FLEX INDUSTRIES

D2.6: Architecture of the Dynamic Energy & Process Management Platform

T2.5 - Definition of the architecture of the overall Dynamic Energy & Process Management platform
WP2: Flexibility assessment and project framework definition

Authors: LINKS

Contributors: RINA-C, CERTH, CAR, CIRCE, UoP, SIM, STAM, FORD, MIL, THD, PRO, TITAN, INTEC

Funded by the European Union

Technical references

Project Acronym	FLEXIndustries
Project Title	Digitally-enabled FLEXible Industries for reliable energy grids under high penetration of Variable Renewable Energy Sources (VRES)
Project Coordinator	RINA CONSULTING SPA coordinator@flexindustries.eu
Project Duration	June 2022 – May 2026 (48 months)

Deliverable No.	D2.6
Dissemination level*	PU
Work Package	WP2 - Flexibility assessment and project framework definition
Task	T2.5 - Definition of the architecture of the overall Dynamic Energy & Process Management platform
Lead beneficiary	5 (LINKS)
Contributing beneficiary/ies	1 (RINA-C), 2 (CERTH), 3 (CAR), 4 (CIRCE), 6 (Up), 7 (SIM), 8 (STAM) , 14 (FORD), 20 (MIL), 21 (THD), 23 (PRO), 24 (TITAN), 26 (INTEC)
Due date of deliverable	31/01/2024
Actual submission date	06/02/2024

* PU = Public

PP = Restricted to other programme participants (including the Commission Services)

RE = Restricted to a group specified by the consortium (including the Commission Services)

CO = Confidential, only for members of the consortium (including the Commission Services)

v	Date	Beneficiary	Author
0.1	02/09/2022	LINKS – Document Draft – TOC	Enrico Ferrera, Francesco Aglieco, Lino Mediavilla
0.2	17/01/2024	LINKS – Chapter 1 and 2	Nicolò Bertozzi, Anna Geraci

Funded by the European Union

0.3	18/01/2024	LINKS – Chapter 3	Anna Geraci
0.4	19/01/2024	LINKS – Revision of Chapter 1 and 2	Nicolò Bertozzi, Anna Geraci
0.4	22/01/2024	LINKS – Chapter 4	Nicolò Bertozzi
0.5	23/01/2024	LINKS – Chapter 4	Nicolò Bertozzi
0.6	29/01/2024	LINKS – Chapter 4	Nicolò Bertozzi
1.0	31/01/2024	LINKS – Chapter 4 – Overall Revision	Nicolò Bertozzi, Anna Geraci
1.1	02/02/2024	TUB – Internal Revision	Mustafa Aktaş
2.0	06/02/2024	LINKS – Comments Inclusion	Nicolò Bertozzi, Anna Geraci

Table of contents

Funded by the European Union

Funded by the European Union

1.1. List of Tables

1.2. List of Figures

Funded by the European Union

1. Introduction

This Deliverable presents the work done and the results obtained in the initial phase (M5-M20) of Task 2.5 (T2.5), titled “Definition of the architecture of the overall Dynamic Energy & Process Management platform” of the FLEXIndustries project. This deliverable serves as the first version of the task report, with the second version to be submitted at M31.

The objective of T2.5 is to define the architecture details, building upon the high-level schema presented in Grant Agreement Document, Figure 1. The following paragraphs delve into the correlation between the components within the schema and the role that T2.5 plays for each of them. In the next paragraph there is a brief description of this correlation, that was deeply described in the next chapters.

The components of the Digital Twin Backend are structured in four different layers: Resources and Data Management Layer, Modelling Layer, Perception Layer and Planning and Operational Layer.

As for the Resources and Data Management Layer (RDML) plays a pivotal role in ensuring seamless interoperability at the field level and with external services. The role of T2.5 is defining the interface between the backbone and the RDML, through the Data Handler component, and definition of the mechanism to retrieve and store data.

The Modelling Layer encompasses the Simulation Engine, a crucial component that acts as a unifying force, harmonizing these diverse modelling elements into a holistic framework. The Perception Layer incorporates AI algorithms that utilize Modelling Layer engines and real-time data for simulations, providing forecasts for energy, material costs, consumption, renewable energy availability, and safety-related information.
Task 2.5 aims to establish an action plan for the development of infrastructure that facilitates communication between the Agents of both Layers and, specifically, the backbone and Multi-Agent System. These Agents correspond to the modules developed in WP5.

The Planning and Operational Layer contains modules for optimizing processes and managing energy. It addresses offline optimal process and energy planning and online optimal process and energy management. Task 2.5 clarifies how the information from the Perception Layer guides the reasoning engines in defining steps needed to achieve the goals set by the application layer, through the Multi-Agent System.

Task 2.5 is also engaged in defining communication not only among the components of the Digital Twin Backend but also with all external elements interfacing with it.
Indeed, the Dynamic Energy & Process Management Platform architecture is completed by the Digital Twin Frontend (T4.4) and two vertical layers: the Security Layer that contains the Security, Privacy and Trust component; and the Deployment Layer that contains the Distributed Deployment and Configuration component. In this deliverable were clarified some aspects for what concerns this last Layers, and which tools will be used to develop the concept behind them.

Funded by the European Union

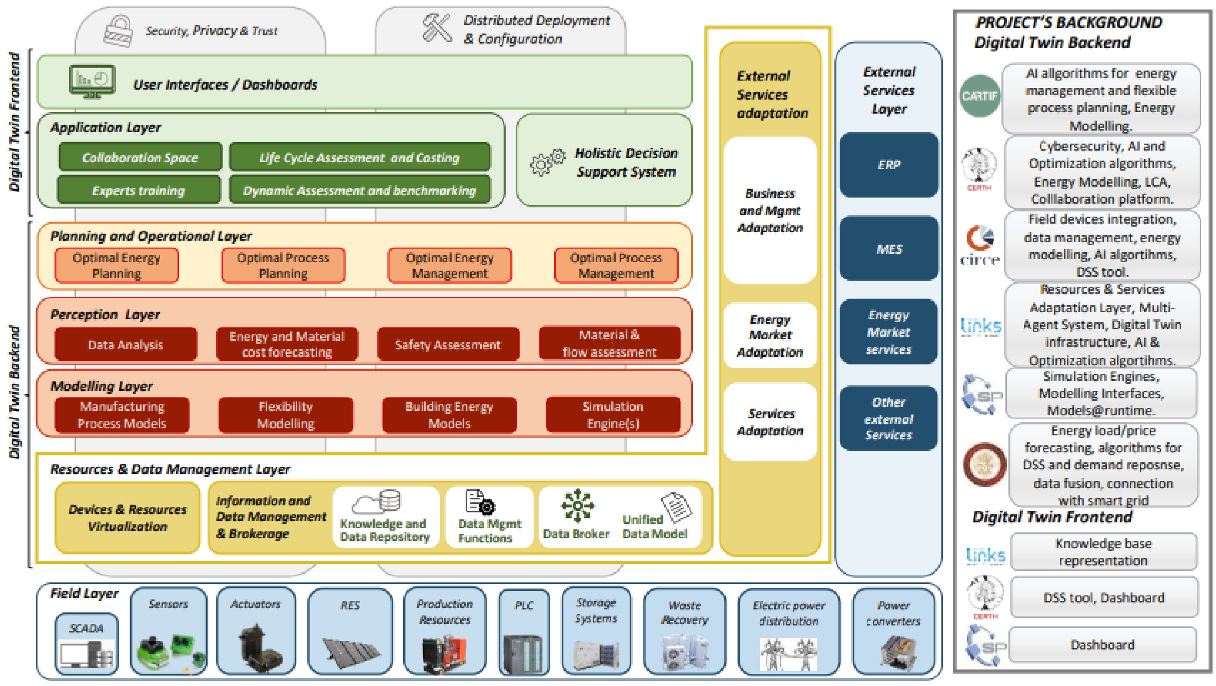

Security, Privacy & Trust
Distributed Deployment & Configuration
Digital Twin Frontend
User Interfaces / Dashboards
Application Layer
Collaboration Space
Life Cycle Assessment and Costing
Experts training
Dynamic Assessment and benchmarking
Holistic Decision Support System
Planning and Operational Layer
Optimal Energy Planning
Optimal Process Planning
Optimal Energy Management
Optimal Process Management
Perception Layer
Data Analysis
Energy and Material cost forecasting
Safety Assessment
Material & flow assessment
Modelling Layer
Manufacturing Process Models
Flexibility Modelling
Building Energy Models
Simulation Engine(s)
Digital Twin Backend
Resources & Data Management Layer
Devices & Resources Virtualization
Information and Data Management & Brokerage
Knowledge and Data Repository
Data Mgmt Functions
Data Broker
Unified Data Model
External Services adaptation
Business and Mgmt Adaptation
Energy Market Adaptation
Services Adaptation
External Services Layer
ERP
MES
Energy Market services
Other external Services
Field Layer
SCADA
Sensors
Actuators
RES
Production Resources
PLC
Storage Systems
Waste Recovery
Electric power distribution
Power converters
PROJECT'S BACKGROUND
Digital Twin Backend
CARTIF
AI allgorithms for energy management and flexible process planning, Energy Modelling.
CERTH
Cybersecurity, AI and Optimization algorithms, Energy Modelling, LCA, Colllaboration platform.
circe
Field devices integration, data management, energy modelling, AI algortihms, DSS tool.
links
Resources & Services Adaptation Layer, Multi-Agent System, Digital Twin infrastructure, AI & Optimization algortihms.
SP
Simulation Engines, Modelling Interfaces, Models@runtime.
Energy load/price forecasting, algorithms for DSS and demand reposnse, data fusion, connection with smart grid
Digital Twin Frontend
links
Knowledge base representation
CERTH
DSS tool, Dashboard
SP
Dashboard

Figure 1: Technical architecture of the FLEXIndustries Dynamic Energy & Process Management Platform.

The following document is structured into these chapters.

The first chapter introduces the purpose and scope of T2.5, providing a brief description of the project overview and its interconnection with other tasks.

The second chapter, about the Architecture Design part, presents the initial architecture design of the Dynamic Energy & Process Management platform. This chapter explores the pivotal role of architecture design in software-intensive systems, focusing on the widely adopted ISO/IEC/IEEE 42010:2011 standard. Evolving from IEEE 1471, it provides a structured methodology for creating architectural descriptions (ADs). We delve into the step-by-step workflow, emphasizing the identification of stakeholders, concerns, and the use of viewpoints to ensure architectural consistency.

The third chapter defines the methodology used to design the architecture. The methodology follows the one defined in the previous chapter. In this chapter the different stakeholders and requirements were analyzed, to produce a list of macro-components that define the whole platform in a generalized way. Details about the specific elements of the macro-components are provided in Chapter 4.

The fourth chapter enters in detailed exploration about 4 + 1 architectural views applied to the platform, as introduced in chapter 2.2. The clear definition of these four views and the scenario enables a comprehensive visualization of the future implementation of the platform from different perspectives.
In this context, the scenario needs to be generalized, considering that the use cases of the different pilots could be highly customized for each of them. Instead, the four views provide different information about the platform, and aim to achieve different goals. The primary

objective of the logical view is to illustrate the functionalities that the platform provides to its users with a brief description of the components involved in the platform. The deployment view describes the physical arrangement of the system's components, outlining the infrastructure and environment in which they operate. It includes a brief description of the tools used, or proposals tools, for component deployment. The process view depicts the dynamic aspects of the system, using UML sequence diagrams to explain the sequence of operations that the platform must be able to perform in order to achieve its objectives. It focuses on the interactions between components during runtime. The development view elucidates how the system's components are created, structured, and organized during the development process. It provides insights into the reasoning behind the design decisions and development methodologies employed.
The chapter ends with a small schema of the role of the partner involved in this architecture design process.

1.1. FLEXIndustries Project Overview

The FLEXIndustries project is dedicated to improving energy efficiency and process flexibility in seven energy-intensive industrial sectors, aiding the transition to better energy sources and operational enhancements. With five key objectives, the project aims to assess industrial flexibility, enhance information and technology integration for improved energy monitoring, establish a secure platform for energy and process management, encourage sustainable energy practices, and validate solutions in real industrial settings. Work Package 2 specifically focuses on a market analysis of flexibility potential in intensive industries and to identify a comprehensive set or parameters of the energy-intensive industries. Which will help develop a monitoring and evaluation plan based on a set of performance KPIs to assess the requirements on flexibility management tools and DSS modules. Finally, to define and establish the architecture of the overall FLEXIndustries platform, that corresponds to the part in which this deliverable will focus on.

1.2. Interconnection with Tasks

Task 2.5 plays a crucial role in the project’s development by designing the Dynamic Energy & Process Management platform, a component to be developed in WP4 and WP5, and then deployed in the demo sites. The inherent connection with both WP4 and WP5 is evident, as the design must align with the functionalities and specifications set forth in these work packages.

This Task is not strictly connected with the other tasks in WP2, except for Task 2.4 (T2.4), and, to a lesser degree, Task 2.3 (T2.3).

T2.4 involves collaborating in the definition of the project architecture, by providing the requirements and specifications for flexibility management tools and Decision Support System (DSS) modules.
The scope of T2.4, according with RTO guidelines and Pilot, is to gather information regarding the DSS modules required for each pilot in the project and how they integrate with the overall platform. Once all necessary information has been obtained, the platform will be designed at the level of setting their specifications and functionalities.

These requirements, furnished by T2.4, serve as the foundation for T2.5 to define an architecture that meets all the needs of the pilots and facilitates interaction between components within the platform. This collaborative effort ensured that the platform's design aligns seamlessly with the flexibility requirements outlined in the project, fostering a cohesive and effective system.

Task 2.3 provides the requirements and specifications that must be taken into consideration in the design of the overall platform architecture, ensuring that the platform meets the needs of energy and process management identified during the analysis of data and technologies.

Furthermore, the interconnection extends to WP6, where the platform is tested using operational data gathered from the demonstration sites. This testing phase is critical for validating the effectiveness and functionality of the designed platform in real-world scenarios.

The collaborative efforts across WP4, WP5 and WP6 create a cohesive workflow, ensuring that the platform not only meets design specifications, defined in Task 2.4, but also proves its practical utility, defined in Task 2.5, through rigorous testing with operational data from the demonstration sited.

Funded by the European Union

2. Architecture Design

In this chapter, an overview of the ISO/IEC/IEEE 42010:2011 standard is provided as a fundamental methodology for developing architectural descriptions (ADs) in software-intensive systems. The chapter emphasizes the application of this standard to the 4+1 Architectural View Model, incorporating perspectives such as Logical, Physical, Process, Development, and Scenarios views. This approach aims to address a variety of stakeholder concerns and foster a comprehensive understanding of software architecture.

2.1. ISO/IEC/IEEE 42010:2011 Standard

The current standard for architecture design, ISO/IEC/IEEE 42010:2011 [1], is an evolution of previous standards, notably IEEE 1471 [2], published in 2000 as the first prevailing approach to software architecture design follows the guidelines outlined in ISO/IEC/IEEE 42010:2011, entitled "Systems and software engineering - Architecture description".

This standard provides a structured methodology for creating architectural descriptions (ADs) for software-intensive systems, including a step-by-step workflow.
The process includes identifying and documenting stakeholders, identifying their architecture-related concerns, selecting and detailing architecture viewpoints to address these concerns, generating architecture views with corresponding models, ensuring view consistency, and documenting the rationale for architectural decisions.

The ISO/IEC/IEEE 42010:2011[1] standard relies heavily on viewpoints and views to capture different facets of a software system, allowing focused examination of specific concerns and issues while maintaining overall architectural consistency. This approach has proven successful in several cases, delivering higher quality artifacts and deliverables than less structured methods that attempt to address all issues in a single pass. Viewpoints are essentially collections of patterns, templates, and conventions tailored to construct specific types of views, such as the functional viewpoint, which contains all the necessary functions for architecture description. IEEE 1471 aimed to establish best practices for the software and systems communities, addressing the fragmented landscape of architectural practices and description languages prevalent at the time. The adoption of this standard led to the recognition of the benefits of a common conceptual foundation and terminology within the software architecture community.
Over the years, the standard has evolved based on a decade of use of the first edition in industry, academia, and government. The 2011 version includes additional features such as modeling architecture decisions, managing correspondence between views, and

[1] The ISO/IEC/IEEE 42010:2011 has been updated and improved in the ISO/IEC/IEEE 42010:2022.

Funded by the European Union

supporting Architecture Description Languages (ADLs) and architecture frameworks. It remains the benchmark standard for software architecture design.

In its current iteration, the standard defines a comprehensive methodology for the architectural description of software-intensive systems that includes a workflow with steps such as identifying stakeholders, identifying architecture-related concerns, selecting architecture viewpoints, creating architecture views with models, and analyzing their consistency. The standard continues to use viewpoints and views to document different aspects of a software system, highlighting specific concerns and issues while ensuring a coherent architectural design.
Rozanski and Woods revised the ISO/IEC/IEEE 42010:2011 in [2], where these key definitions are introduced. In Figure 2 the relationships between these concepts are graphically represented.

2.1.1. System of Interest

The system whose architecture is being studied or considered.

2.1.2. Stakeholder

An individual, group, organization, or similar entity with a vested interest in the realization of the system.

2.1.3. Architecture

The basic ideas or characteristics of a system within its environment, expressed through its components, connections, and the principles that guide its design and growth. Essentially, it's how a system is structured, its properties, how it interacts with its environment, and so on.

2.1.4. Architectural Description

A document or output used to convey the structure of an architecture, which may include diagrams such as component diagrams or data flow diagrams.

2.1.5. Concern

An interest in a system that is important to one or more stakeholders. This can be a requirement, either functional or non-functional, or an objective that a stakeholder has with respect to the system.

2.1.6. View

A collection of models and explanations that represent a system or part of it, focusing on a particular set of concerns.

2.1.7. Viewpoint

A set of patterns, templates, and conventions that guide the creation of a particular type of view.

Funded by the European Union

2.1.8. Model

A simplified representation of a facet of the architecture, often in the form of a UML diagram[2].

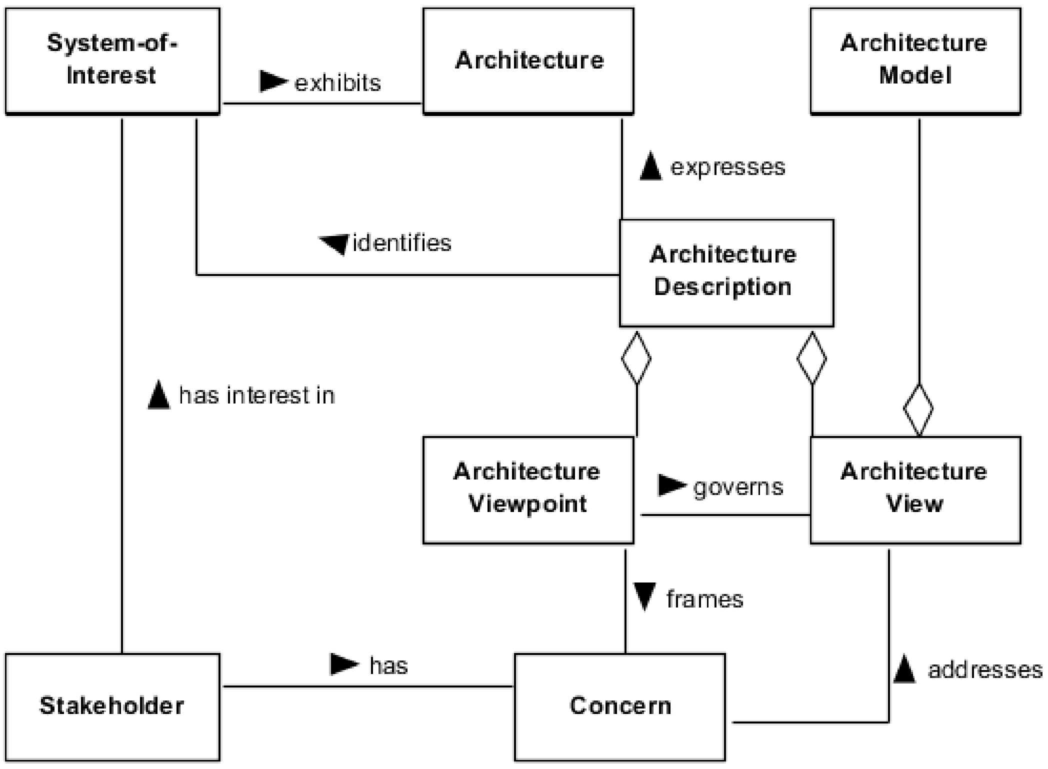


Figure 2: Architecture concepts and their relationships.

2.2. 4+1 Architectural View Model

The 4+1 model aims to address the concerns of different stakeholders and provides a holistic understanding of the software architecture.
The "+" in the 4+1 model represents the 'Scenario' view, which complements the four architectural views by providing specific use cases or scenarios that demonstrate the system's behavior in various situations.
The 4+1 Architectural View Model, developed by Philippe Kruchten [3], extends the traditional software architecture viewpoints and includes the following perspectives:

2.2.1. Logical View (Functional Viewpoint)

This view focuses on the functional requirements of the system and describes the system in terms of key abstractions or objects. It outlines the interactions, responsibilities, and

[2] In the context of this deliverable, it was preferred not to use UML diagrams in order to achieve a simpler knowledge explaination. In addition, some technicalities required by this illustration standard were not defined by the partners.

Funded by the European Union

relationships between these objects. The main purpose was to capture the high-level design of the system’s functionality.

2.2.2. Physical View (Deployment Viewpoint)

The physical view describes the deployment and distribution of the system components across the hardware. It includes details about the network connections, and how software components are mapped to the physical environment. The goal is to specify the system's deployment architecture, addressing hardware and network considerations.

2.2.3. Process View

This view addresses the runtime aspects of the system. It illustrates how the system will be structured to handle concurrent execution, including processes, threads, and their interactions. The objective is to show how the system components will execute concurrently and how they will coordinate and communicate.

2.2.4. Development View

This view focuses on the software development aspects and provides guidance on how to organize the software development process. It includes information about coding standards, development tools, and version control. It is designed to help developers implement the system and ensure consistency and efficiency.

2.2.5. Scenarios (Use Cases) View

The scenarios or use case's view illustrates how the system behaves in response to different scenarios or use cases. It highlights the interaction between the system and external entities under various conditions. Its purpose is to provide a realistic, user-centric perspective on the system's operation.

Funded by the European Union

3. Methodology

In this section, we delve into the methodology employed by FLEXIndustries for architecting its software, aligning with the ISO/IEC/IEEE 42010:2011 standard. This discussion can help to understand the fundamental concepts guiding FLEXIndustries' design choices, aiding both the seasoned practitioner and the occasional reader in comprehending the rationale behind the project consortium's decisions.

3.1. Architecture Design Process

Rozanski and Woods [4] have formulated the architectural design process based on the following definition: "Architecture Definition is a process by which stakeholder needs and concerns are captured, an architecture to meet these needs is designed, and the architecture is clearly and unambiguously described via an architectural description." The foundation for this process is the ISO/IEC/IEEE 42010:2011 standard. The FLEXIndustries project adopted the process proposed by [4], aligning it with the mentioned standard (see Figure 3).

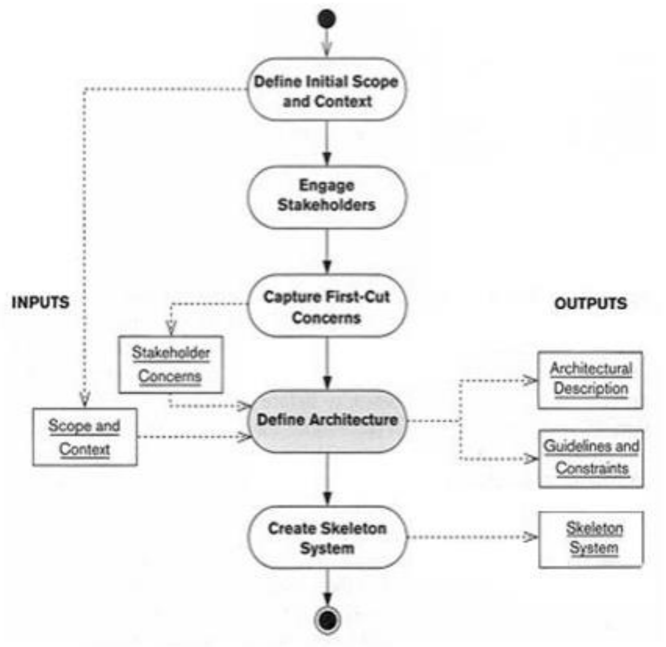


Figure 3: Activities supporting architecture definition.

The Architecture design process follows the pipeline outlined in Figure 3.

Initially, the Scope and the Context of the Architecture were defined, as documented in D4.5. This document presents an overview of various tools that could be utilized for the platform's purpose. The deliverable's objective was to provide a comprehensive view of potential tools, including their pros and cons. Based on requirements, scope and context it is possible to determine which tool best aligns with the application's customization needs.

Subsequently, a stakeholder's analysis was conducted in the next subchapter, to identify and understand the individuals or entities involved in the organizational structure and production processes of different factories. These actors, representing end-users, have varied expectations for platform functionality, resulting in different access rights and restrictions.

The input from both stakeholders and definition of scope and context converges in the definition of the architecture, encompassing both architectural description and Guidelines and Constraints for the platform.

The final phase involves the output described in chapter 4, which is the creation of a system skeleton. This ensures that all information gathered in the previous steps is accurately collected and implemented in a well-structured platform.

3.2. Analysis of Stakeholders and Requirements

As was previously mentioned, different stakeholders are involved in the organizational structure, and particularly in the production processes of the factories with different roles and information needs. From this set of stakeholders, a list of actors of the FLEXIndustries platform can be extracted in the form of different end-users expecting different functionality of the solution, and in consequence with different access rights and restrictions to the services offered by the Platform.

In Table 1 was presented the list of stakeholders involved in the use of the platform.

Table 1: Stakeholders

Platform end-users	Role
Planning Director	Responsible for planning and execution of strategic plans, according to the analysis of the DSS modules.
Managing Director	Responsible for all activities related to the management process from different point of view, raw materials, use of energy, production efficiency.
Purchasing Director	Responsible for all activities related to the purchase of raw materials and machinery. (e.g., In case of predictive maintenance they should buy the necessary materials in advance.)
Industrial Manager	Responsible for ensuring efficient management of materials and workers within a production environment. Their primary responsibility is to optimize production processes to enhance energy efficiency.
Maintenance Manager **Maintenance Technician**	Responsible for ensuring the optimal functioning of equipment, machinery, and facilities to prevent downtime and maintain operational efficiency.

Funded by the European Union

Management Engineer	Responsible for analysis and searching possible inefficiencies and proposing optimization changes: working methods, workforce, equipment.
Quality Manager	Responsible for guaranteeing that the product complies with legal requirements and expectations of the customers. Indeed a faulty/outdated machinery could produce low quality product, with consequent waste of energy.
Production Manager	Responsible for planning and managing material resources, supervising the production facilities, machinery, equipment and daily performance of each worker.
Energy Manager	Responsible for planning and managing the energy use in the plant, according to the analysis done by the HDSS modules of the platform.

According to the role and responsibilities that every stakeholder has in the production process of the factory, the functionalities expected by every user of the platform is different, resulting in the specification of a series of desired requirements for the FLEXIndustries Platform. These requirements can be deduced from the information provided by the requirement of the Modules and are translated into a set of functional blocks, modules or components in the Platform able to offer the expected functionality.

Regarding the Functional Requirements, they could be classified according to the components to which they relate, as outlined below in Table 2.

Table 2: Platform Requirements

Requirement ID	**Functional/ Non-Functional**	**Component involved**	**Description**
R1-001	Functional	General Platform	The platform must support the execution of modules.
R1-002	Functional	Module Integration	The platform must facilitate interaction and data exchange between modules.
R1-003	Functional	Data Management	The platform must provide access to all available historical process data required for module execution

Funded by the European Union

R1-004	Functional	Execution Level	The platform must be designed with a scalable architecture to handle various activation of the modules.
R1-005	Functional	Data Management	The platform must be designed with a scalable architecture to handle various data requests, deterministic and non.

The R1-001, related to the Platform, means that the platform must be able to support the execution of modules for energy-related data forecasting (M1), energy price forecasting (M2), energy consumption and flexibility forecasting (M3), process planning (M4), energy and flexibility markets (M5), and real-time process optimization (M6). The execution of these modules occurs within a cooperatively and organized workflow.

The R1-002, related to the modules integration means that for example, results from M1 should be accessible by M3, M4, and M6.

The R1-004, related to the fact that some modules have a deterministic activation, (e.g., M1, M2, M3, and M6), while M5 could be executed in both deterministically and on-demand. As for M4, the execution of the module is solely on-demand. The platform must be able to handle the activation request for modules, considering that some executions are deterministic and periodic, while others are not.

The R1-005 is connected to the previous requirement. The platform must be able to retrieve data collected by T4.1. Data retrieval aligns with the execution of the module, meaning that if a module is executed deterministically (e.g., every hour), the data retrieval process is also determined. Instead, for modules that allow the on-demand request, data retrieval is on-demand. Therefore, the platform must be capable of fetching input data when the module is executed.

Additional details can be added, or these requirements can be tailored, based on the specific project needs and implementation context.

Funded by the European Union

3.3. Required Components

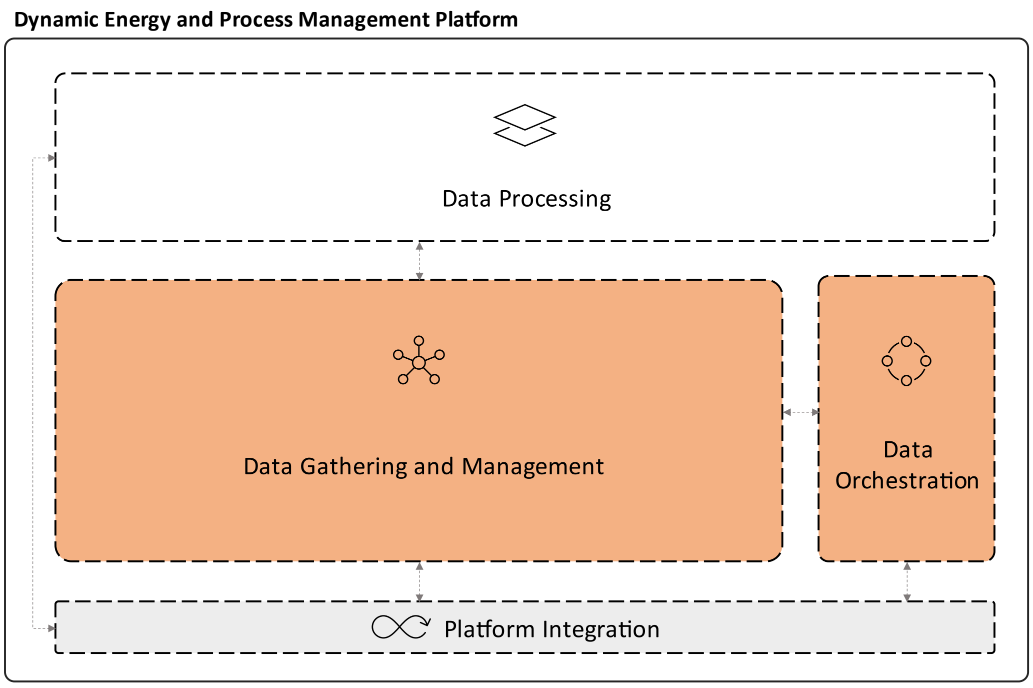

Dynamic Energy and Process Management Platform
Data Processing
Data Gathering and Management
Data Orchestration
Platform Integration

Figure 4: Macro-level modules composing the FLEXIndustries Platform.

In Figure 4 is possible to see all the components that are required for the FLEXIndustries Platform. The components were chosen based on the requirements listed before and fully discussed.

3.3.1. Data Gathering and Management

Data Gathering and Management was composed of a data broker, as central element, that manages the data streams among the platform's component and with the other entities within the FLEXIndustries architecture. It facilitates the interconnection of modules by leveraging the publish/subscribe paradigm.

The data handler, another integral part of this component, processes requests from the data broker, utilizing information from data sources sent in the payload. The data source information includes the APIs through which the pilot exposes its data. Subsequently, the data handler retrieves and stores data in a specific data-storage accessible by the agents.

Consequently, this component includes also the Metadata Storage which stores the data collected from the field and the external sources, offering to the agents the ability to retrieve large data without imposing overhead on the data broker.

3.3.2. Data Orchestration

The data orchestration component aims to configure, deploy, monitor and orchestrate the platform components. It is replicated by the Multi-Agent System defined in T4.3 for the Planning and Operation Layer. Its primary role is to efficiently distribute and execute agents based on the data they need to retrieve from the database. It can be viewed as

Funded by the European Union

orchestrating data rather than services, since it determines which agent can run based on the availability of the required data.

Indeed, the Data orchestration communicates with the Data Processing component to coordinate their execution, and interfaces with Data Gathering and Management to request the necessary data from the agents.

3.3.3. Data Processing

The "Data Processing" component in FLEXIndustries is comprised of three main elements.

Firstly, it involves the development of a specific task aimed at supporting a holistic modeling and simulation approach. This task (T4.2) enabling online modeling and simulation with a focus on model energy demand according to production objectives, integrating various layers of modeling, and extending simulation engines to support value-stream, process, and discrete-event simulation.

Secondly, it includes contributions from components developed within Work Package 5 (WP5). Tasks 5.1 to 5.4 within WP5 include the analysis and forecast of process, material, and energy-related data, the development of a Process and Energy Planning tool, energy efficiency, and flexibility monetization for energy markets, and the creation of a Process.

In the "Data Processing" component, is present also the Frontend of the platform developed by T4.4. This involves establishing the structure and framework to host the Application Layer and the Decision Support System.

3.3.4. Platform Integration

Platform integration is a component that aims to supervise the ongoing system integration of the WP4 infrastructure with the modules developed in WP5 and refined in WP6. It also aims to address security requirements by implementing authorization and authentication tools customized for the application. The entire process will undergo testing facilitated by a central code repository and a Continuous Integration/Continuous Deployment (CI/CD) infrastructure.

Funded by the European Union

4. Architecture Views

4.1. Scenario

The scenario serves as the foundation for the logical, process, physical, and development views. It is the reference point for the entire project, as it outlines the platform's intended features and the technical steps necessary to implement them.

For FLEXIndustries, the abstract use case is predetermined by the Grant Agreement in the form of work packages and tasks. From a high-level perspective, this project is aimed at helping industries optimize their energy consumption by interacting flexibly with the power grid. To achieve this optimization, data is crucial. The FLEXIndustries Platform is designed to work with a set of data sources that characterize the scenario itself.
In summary, regardless of the specific details that distinguish one pilot from another, there is a need from an architectural perspective to manage information flows between modules efficiently.

4.2. Logical View

The primary objective of the logical view is to illustrate the functionalities that the platform provides to its users. These functionalities are closely related to the modules that make up the architecture, as shown in Figure 5, which presents the main services of the FLEXIndustries platform. It's noteworthy that certain modules are not directly derived from WP4 and WP5. This deliverable aims to ensure effective communication between all the building blocks produced in these phases, relying on additional services for completion.

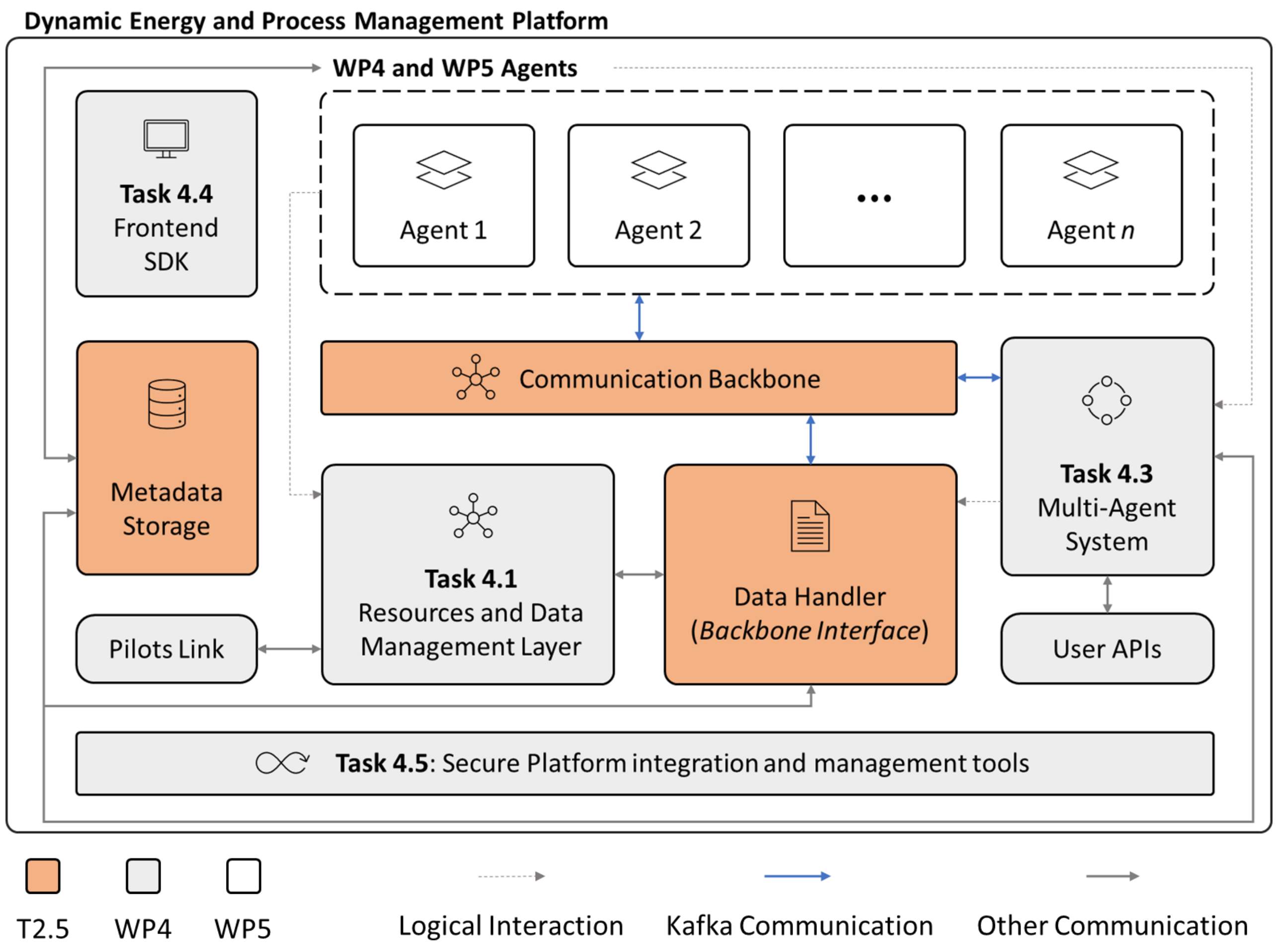


Figure 5: Logical View

In the following sections, the roles of each macro-thematic building block will be discussed, and it will be shown why certain services are essential for fulfilling user-requested plans and maintaining a seamless architecture.

4.2.1. WP4 and WP5 Agents

The operational components coming from WP4 and WP5 are treated as agents of this platform. These agents practically perform computations requested by the user and whose results are needed by the user himself. Each of these agents is responsible for a small part of the overall canvas. The responsibility of the Multi-Agent System is then to coordinate the flow of information between these modules in order to respond to the user with an aggregate view of their contribution. The Dynamic Energy and Process Management Platform ultimately attempts to seamlessly ensure that these core components are able to converge on a specific outcome in a cooperative manner.

The success of this mechanism is due to the abstracted communication layer seen by each service. In other words, each WP4 and WP5 Agent receives as input only the task it has to perform. It does not have the visibility of the whole platform, since it does not need it to proceed with the execution of its tasks.
The activities that can be carried out by an agent towards the system as a whole are two: to communicate the completion of a task or to request some data from the field.

Funded by the European Union

This second activity causes the individual agents to also communicate with the Metadata Storage and, logically, with the Resource and Data Management Layer. This means that if new data is required to complete a task, it is possible for the agent to pause its execution and request its need from the Multi-Agent System. Once the data is available, the orchestrator notifies the agent that it can access the repository to retrieve what it needs[3].
Therefore, the arrow in Figure 5 between the WP4 and WP5 Agents and the Metadata Repository implies a physical communication, while the connection between the agents and the Task 4.1 components is ephemeral. The former is actually used by the agent to access the data, while the latter is logical and only indicates that the information retrieved by the Metadata Storage comes from the Resources and Data Management Layer. Technically, the agents access the data from the Metadata Storage, while this one actually receives the data from RDML in the background.

This decoupling between the Communication Backbone and the WP4 and WP5 Agents ensures that these ones are not responsible for the communication or the orchestration mechanism that is responsible for the macro blocks described in sections 4.2.2 and 4.2.3. Thus, their operation is independent of how these other services are implemented, which allows the platform to be flexible to changes or improvements.

4.2.2. Communication Backbone

The Communication Backbone is the central event manager of the Dynamic Energy and Process Management Platform. The core services of the architecture communicate through this component, as the input coming from the user is also shared within the communication backbone.
The name *event manager* derives from this crucial point: all messages exchanged between the core services are events and therefore follow a fully asynchronous paradigm. The same is true for user input, which is asynchronous and can only be ingested and managed by the architecture when the source provides it.

Thus, the communication backbone is not just a marshalling yard that acts as a network switch, capable only of redirecting the messages coming from a source to a specific destination labeled with its internal IP address. In fact, the communication backbone is also able to manage and keep separate the different data flows, while ensuring a high throughput between the components.

In conclusion, whenever a new input request comes from the user, the Multi-Agent System is the first component to be triggered by this event. As a consequence, it elaborates the sequence of steps necessary to carry out the planning by publishing a set of events addressed to the specific agents called in cause by the particular workflow. Since they are in turn triggered by the Multi-Agent System, their activity is strictly limited in time, since it ends when the assigned task is completed.

[3] By isolating the data flow from the control flow, the platform achieves higher throughput. These features are described in more detail in the following subchapters.

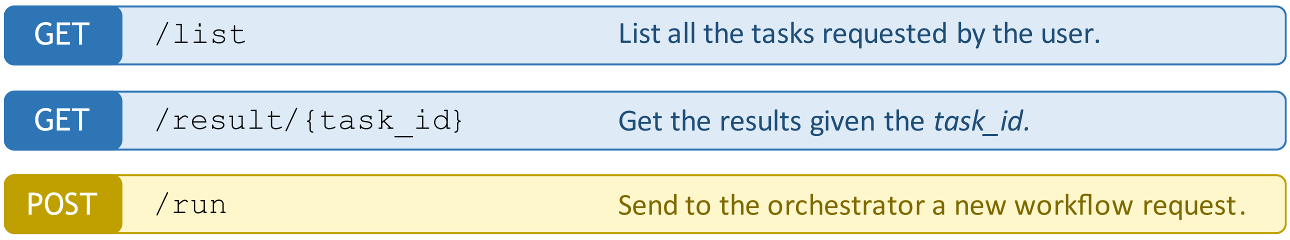
Funded by the European Union

4.2.3. Multi-Agent System

4.2.3.1. Component

The Multi-Agent System is the data orchestrator of the platform. It is responsible for triggering the agents involved in the workflow requested by the user whenever they are needed.
Since these involvements are asynchronous, the orchestrator is able to continuously manage possible further inputs coming from the user while executing the workflows requested in the past.
The interoperability achieved by this component depends not only on the asynchronous paradigm, but also on Metadata Storage. This is used by the orchestrator to store information about its current pending activities in such a way that it can easily pause and resume a task without losing control over it.
Logically, the Multi-Agent System is connected to the WP4 and WP5 Agents and the Data Handler. These are the only two recipients of events published by the orchestrator over the communication backbone. For obvious reasons, the first connection is used to distribute the computational load among the agents to complete a user request. The second, on the other hand, is needed whenever an agent realizes that it cannot complete a task because some data preparatory to the execution of the task is missing.

4.2.3.2. User APIs



GET	`/list`	List all the tasks requested by the user.
GET	`/result/{task_id}`	Get the results given the *task_id.*
POST	`/run`	Send to the orchestrator a new workflow request.

Figure 6: APIs exposed by the platform to the user.

The list of APIs exposed by the Dynamic Energy and Process Management Platform are listed in Figure 6. The user can run the FLEXIndustries workflows by contacting the `/run` POST API. This request requires some information from the user, such as the type of workflow they want to run and some optional data for the agents. At this point, the server assigns a unique identifier to the request so that the user can retrieve the result when it is complete.

The `/list` endpoint can be used to retrieve the list of workflows run by the user. This set includes not only the completed ones, but also the schedules currently in execution. To retrieve the result associated with a particular task, the user should use the `/result/{task_id}` endpoint, which returns in output the overall result obtained by aggregating the contributions of the individual agents.

Funded by the European Union

4.2.4. Data Handler

All data requests coming from the WP4 and WP5 agents are managed by the Data Handler. The role of this component is to provide an asynchronous interface from the Resources and Data Management Layer to the other core components.

This interface is responsible for managing the data requests coming from the Multi-Agent System[4] to the Resources and Data Management Layer.
From a general perspective, the Data Handler is able to interact with all of the interfaces that the Task 4.1 component exposes through its Historical Data Service and Real-Time Data Service[5]. In any case, the responses provided by these two interfaces are formatted according to the data model adopted by the RDML itself, which differs from the format with which these data tasks are shared by the communication backbone.
In order to place core components capable of understanding the events exchanged between them, it is necessary to ensure that the data schema is the same.
The data handler intervenes in this situation by decoupling the format of the RDML from the format of the data tasks exchanged between this module and the orchestrator. In this way, it is possible to modify the two standards separately over time, guaranteeing the platform that it can adapt to new formats by correcting only the data models encapsulated in the Data Handler.
Indeed, it would not be advantageous for the RDML to have a format that completely mirrors the one adopted by the Multi-Agent System. These two schemas serve two completely different purposes and merging them would only result in a huge amount of time being wasted.

In addition, the Data Handler also acts as a data endpoint for the WP4 and WP5 Agents. This means that if they need some special input data that cannot be handled within the Task 4.1 component[6], it is possible to make available this further data source by designing and developing the corresponding Data Handler.

Once the RDML responds to the request made by the Data Handler, the corresponding data is published by the Data Handler to the Metadata Storage so that the agent that logically generated the data task can read the result.

4.2.5. Resources and Data Management Layer

This component is the aggregated data source that the WP4 and WP5 Agents can use to retrieve the data they need.

[4] As introduced in subchapter 4.2.1, in a data request pipeline the Multi-Agent System is just an intermediary between the agent which originated the request and the Resources and Data Management Layer.
[5] For more information about these two modules, please refer to D4.3.
[6] For example, if a particular highly customized protocol cannot be integrated into the Resource and Data Management component for some reason.

Funded by the European Union

4.2.5.1. Component

As already anticipated in the previous subchapter, this module responds only to the data requests that come from the Data Handler. Depending on the type of request, the Data Handler can use the services exposed by this component.
In order to store the data in the internal data store and then offer it to the Data Handler, this module must be connected to the real world by a dedicated link.

4.2.5.2. Pilots Link

In fact, this document has not specified whether the entire Dynamic Energy and Process Management Platform will be deployed at the edge or in the cloud. Therefore, it is difficult to highlight the characteristics of the connection between the platform and the pilot plants.
What can be said today is that in the case of a cloud deployment, the data transfer must be kept encrypted to ensure the security of the payload.
In addition, the platform should be able to handle heterogeneous situations. For example, it may be possible to run part of the platform, including data storage, in the cloud, while the rest of the architecture runs on the edge.
Since this information is missing or only partially available at the time of writing, the second version of this deliverable will provide more appropriate specifications in this regard, with particular attention to the specific implementation requirements of each pilot.

However, the Resources and Data Management Layer can be deployed in the cloud or at the edge, depending on the specific need. If it will run on the edge, it will be equipped with export services (e.g. MQTT export) so that the collected data can be transferred from the edge to any machine in the cloud.
When running in the cloud, the sensors need to send data to the machine in the cloud. For example, the sensors could send data directly to an MQTT broker in the cloud. Another alternative in the example of running in the cloud is that instead of sending data directly to the cloud, the sensors in the factory can send it to an RDML installed in the factory, and from there the data can be sent to the cloud using export services. In this case, the RDML in the factory is only responsible for sending the data and no other services are used.

4.2.6. Frontend SDK

Although it is technically wrong to include an SDK in a platform architecture, in this case the name of this component depends only on the official title of Task 4.4. This task aims to design the structure and frameworks needed to build the front end of the DSS modules developed in WP5. For illustrative purposes only, this software development kit is treated as a separate component in Figure 5, even though it should be replicated across all agents that will expose a GUI composed starting from this framework. In addition, this module is not linked to any other service in the FLEXIndustries platform. This choice depends on the fact that it is necessary to understand the options and configurations that these frontends offer to the user. Once the whole picture is complete, it is possible to connect this component to the modules it needs.

4.2.7. Security Platform Integration and Management Tools

This module is composed of several tools that are useful to monitor what is happening inside the platform and to ensure the security between the internal and external actors.

Funded by the European Union

Monitoring is essential in any data ingestion platform, as it allows to check the status of each service in order to apply the appropriate resolution actions.
In this particular case, it is possible to use different monitoring techniques that can be applied to control the architecture from different perspectives.
Regarding the different services, the health check mechanism is probably the best option to guarantee that all modules are running as expected. Instead, regarding the amount of load assigned to the core modules, it is possible to continuously analyze some metrics about the communication backbone to know if the running instances are able to handle all the incoming requests or not.
These two methods can be used to both ensure that the actual platform setup is healthy and able to handle the requests coming from the user[7].

The security goal can be achieved by implementing secure communication within and with the platform. Secure communication does not just mean using SSL on top of TCP to ensure that the APIs used by the user are not subject to man-in-the-middle attacks[8]. In fact, a centralized identity and access management service might be the best choice for implementing authentication and authorization of users and services.
Another possible option is to use a key management module to secure communication between the microservices that make up the architecture.

Finally, this component allows the dynamic deployment of the different modules in a DevOps and unified manner.

[7] The set-up is the actual state of the architecture. Simply speaking, the state is composed of the list of services that should be in execution and the number of instances of each service.
[8] However, HTTPs instead of HTTPS is a crucial and strict requirement to guarantee security from the user's point of view and interest.

Funded by the European Union

4.3. Deployment View

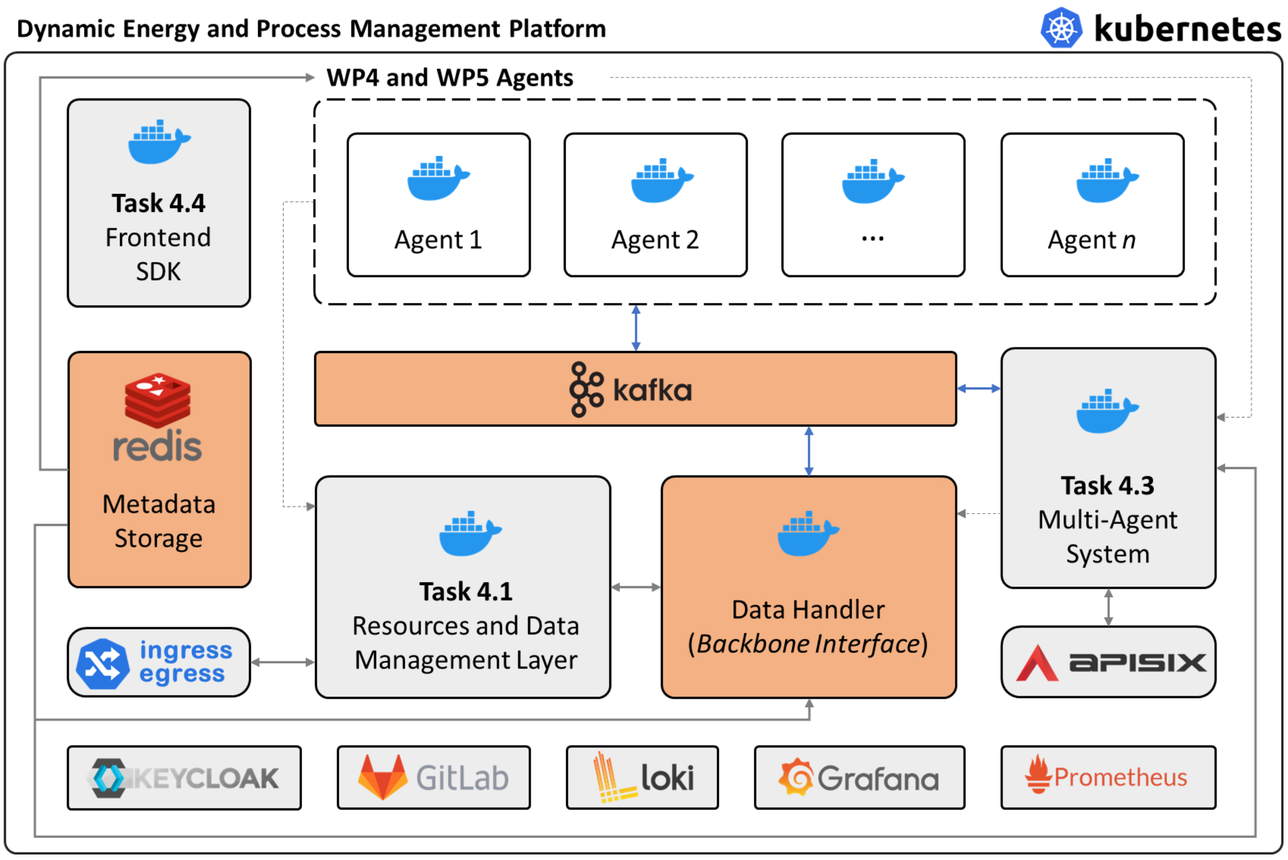


Figure 7: Deployment View.

The Logic View of Figure 5 can be realized by implementing and using a set of tools that are widely used in distributed architecture designs, like it depicted in Figure 7.
This deployment view is still general, since it is simply a practical version of what was described abstractly in subchapter 4.2.
From a standard point of view, a Deployment View should be integrated with the more technical features specific to this type of architectural designs, such as load balancing, secret management, SSL implementation, high availability replication mechanism, disaster recovery, dynamic instantiation, and more.
However, since this document describes the first version of the platform, and since not all the tools used to achieve these goals have been definitively identified, they will not be considered in this section.

Finally, it is worth noting that Keycloak and GitLab, and their alternatives, mentioned in subchapter 4.3.7 are more the subject of Task 4.5 and its deliverable. Consequently, these solutions will only be mentioned here since they could be easily integrated into the platform. On the other hand, the corresponding deliverable will define the final choices and the rationale behind them.

4.3.1. Docker Containers

Docker's importance in deploying applications across multiple platforms, including edge and cloud, is due to its integration of container tools and technologies. Docker is built on

Funded by the European Union

an open foundation, leveraging components such as runc, an implementation of the Open Container Initiative (OCI) specifications, and containers, a high-level container runtime. This integration provides a robust framework for encapsulating applications and ensuring consistent deployment. In addition, Docker provides developer-friendly features such as a CLI for container lifecycle management and access to a public container registry. This seamless orchestration is complemented by high-level tools such as Docker Compose and Kubernetes, which improve application management, scalability, and resiliency in dynamic environments.

Therefore, the deployment that will be used relies on the Docker image building process. Then the Multi-Agent System will logically interact with all these services, but actually it is not clear whether a service registry is needed or not. Of course, with this kind of component, it is easier for the platform players to discover their neighbors and exchange data with them. On the other hand, it is also true that some features of Kubernetes[9] and Apache Kafka[10] features should partially solve this issue. Therefore, the choice depends on the effort required to deploy and manage this additional service, which should simplify interaction within the platform.

4.3.2. Kubernetes

Kubernetes[11] is a popular tool for managing container-based workloads in a horizontally scalable and distributed manner. It uses a declarative configuration approach that automates deployment, scaling, and operational tasks while abstracting from the hardware resources of the underlying instances. In a symbiotic relationship with Docker, Kubernetes seamlessly integrates Docker containers as its primary packaging format, combining the benefits of Docker containerization with the robust orchestration capabilities of Kubernetes. This dynamic duo, complemented by tools like kubectl and Helm, creates a comprehensive ecosystem for efficiently deploying, scaling, and managing containerized applications across diverse environments.

Kubernetes uses various service components, including Kubernetes Service, kube-proxy, and Ingress controller, to distribute loads efficiently across services in a cluster. Kubernetes Service offers a dependable endpoint for load balancing, kube-proxy maintains network rules for routing traffic to pods, and the Ingress controller manages external access and routing strategies. These components can improve resource utilization, scalability, and performance of containerized applications in a Kubernetes cluster. They can also be used with Kafka Consumer Groups to distribute the load among multiple modules. It is important to note that the only difference between components is whether they need to be connected to the communication backbone. Consumer groups may benefit the WP4 and WP5 Agents, as well as the Multi-Agent System and the Data Handler[12]. Instead, the others would communicate using other protocols like HTTP or

[9] More about Kubernetes Service in subchapter 4.3.2.
[10] More about the Consumer Groups in subchapter 4.3.3.
[11] https://kubernetes.io
[12] The replication of the agents, Multi-Agent System and Data Handler requires these components to be stateless. For instance, the Agent *n* can be replicated, and the load distributed across its replicas solely and only if the actions performed by the module does not depend on the history of it.

Funded by the European Union

gRPC. It is still under evaluation the exploitation of proxy and service meshing tools like Istio[13], Envoy[14], or Traefik[15].

4.3.3. Apache Kafka

Apache Kafka[16] is an open-source, distributed event streaming platform that is renowned for its ability to handle real-time streaming data. It has a central broker that oversees the organization of data streams into individual records encapsulated in topics. This design ensures data persistence across multiple server instances, which increases reliability. Kafka's main components consist of producers, which generate and publish data, topics that categorize data streams, consumers that subscribe to and process these streams, and brokers that facilitate communication between producers and consumers. This framework is designed for speed, scalability, and efficient distributed operations, making it a robust solution for real-time data streaming applications.

For these reasons, Kafka seems perfectly capable of managing the messages coming from the platform components connected to it. As described in Section 4.4, this data is split into different topics to increase throughput while trying to balance the load between multiple services. This mechanism can be achieved using Kafka consumer groups. Multiple Kafka clients can be grouped together to distribute topic messages evenly among interested consumers.

Other tools besides Kafka can also be used for this purpose, as listed in the Table 3.

Table 3: Alternative tools for Apache Kafka.

Tool	Description
RabbitMQ[17]	RabbitMQ is an open-source message broker that implements the Advanced Message Queuing Protocol (AMQP). It provides reliable message queuing, supports various messaging patterns, and is highly extensible. RabbitMQ is suitable for distributed systems and facilitates communication between different components.
Mosquitto[18]	Mosquitto is an open-source MQTT broker that enables efficient communication especially for the Internet of Things. It supports the MQTT protocol, providing a publish-subscribe model for messaging. Mosquitto is simple, resource-efficient, and easy to integrate into IoT scenarios.

[13] https://istio.io
[14] https://www.envoyproxy.io
[15] https://traefik.io
[16] https://kafka.apache.org
[17] https://rabbitmq.com
[18] https://mosquitto.org

Funded by the European Union

Zenoh[19]	Zenoh is an IoT middleware that prioritizes efficient and decentralized communication. It supports various communication patterns, such as publish-subscribe and request-reply, and seamlessly integrates multiple protocols like MQTT, CoAP, and WebSocket. Zenoh's protocol-agnostic approach provides flexibility and scalability in dynamic and resource-constrained environments.

4.3.4. Redis

Redis[20] is an open source, in-memory data structure store that serves as a high-performance and versatile key-value database. It provides fast data retrieval and caching capabilities. In addition, it can also be used as a publish/subscribe message broker and as a backend for applications that require fast data access..

Unfortunately, there are not that many tools that can be used to replace Redis. The most popular is Memcached, which can be used as a caching system by some NoSQL databases, although this is not a core feature for them. This category includes tools such as MongoDB and Couchbase.

4.3.5. Ingress and Egress

Inside a Kubernetes environment[21], the ingress manages external access to services within the cluster, providing HTTP and HTTPS routing. It acts as an entry point, handling traffic routing, load balancing, and SSL termination. Conversely, the egress governs outgoing traffic, controlling pod access to external services or the internet. Ingress and Egress together regulate how external requests enter the cluster and how internal requests leave, contributing to secure and controlled communication for Kubernetes applications.

In the deployment view of the Dynamic Energy and Process Management Platform depicted in Figure 7, it seems these two communication gates are basically associated to the data flow that is connected to the pilots' premises. Of course, this is only a simplification. The API Gateway managed by Apache APISIX is also an endpoint for a potential FLEXIndustries user and, therefore, it is possible to also consider it as an ingress and egress point for the architecture.
The association of this block to the pilots is only a graphical notation that is exploited for focusing the attention of the part of the architecture where a lot of data will flow. In addition, this entry point will in case be accompanied by also a VPN Gateway in order to ensure the security of the link between the on-premises data source and the cloud platform[22].

[19] https://zenoh.io
[20] https://redis.io
[21] In reality, the ingress and egress concept are adopted generally adopted also to indicate the incoming and outcoming data of a cloud environment.
[22] At the time of this writing, it is not clear which use cases will provide data access directly from the field with a link to their on-premises data storage, or how many already have cloud subscriptions that are also used for data storage purposes.

Funded by the European Union

4.3.6. Apache Apisix

Apache Apisix is an open-source API gateway that provides high-performance and scalable API management. Some key features of Apisix are scalability, dynamic load balancing and dynamic configuration through REST APIs.

Other tools that are comparable to Apache Apisix are described in Table 4.

Table 4: Alternative tools for Apache Apisix.

Tool	Description
Traefik[23]	Traefik is a modern, open-source reverse proxy and load balancer designed for dynamic environments like Docker and Kubernetes. It excels in automatic service discovery, offers support for multiple protocols, and simplifies SSL/TLS certificate management.
Tyk[24]	Tyk is an open-source API gateway that provides comprehensive API management capabilities, including rate limiting, authentication, analytics, and a developer portal. It is suitable for securing, monitoring, and controlling access to APIs and supports both on-premises and cloud deployments.
Kong[25]	Kong is a scalable API gateway built on NGINX with features such as load balancing, authentication, traffic control, and over 80 extensible plugins. Suitable for various environments, Kong manages the entire API lifecycle and supports RESTful and GraphQL APIs in microservices architectures.

4.3.7. Keycloak

Keycloak is an open-source identity and access management solution. Developed by Red Hat, it provides single sign-on (SSO), user authentication, authorization, and social login capabilities. Designed to secure applications and services, Keycloak supports multiple protocols, including OAuth 2.0 and OpenID Connect.
This makes Keycloak suitable for centralized identity and access management (IAM), but not the best choice for securing communication between microservices. Instead, SPIFFE (Secure Production Identity Framework for Everyone) and SPIRE (SPIFFE Runtime Environment) is a specialized framework focused on providing secure identities for microservices in dynamic and distributed environments.

As a result, platform security could be assigned to multiple services and tools. Keycloak could manage the authentication and authorization of FLEXIndustries users, while

[23] https://traefik.io/traefik
[24] https://tyk.io
[25] https://konghq.com

Funded by the European Union

SPIFFE/SPIRE could secure the communication between the modules that make up the Dynamic Energy and Process Management Platform.

4.3.8. GitLab

GitLab is a web-based platform for managing Git repositories, providing a comprehensive set of features for source code management, continuous integration, continuous delivery, security, and collaboration.
As a Git repository, it provides code versioning, control, and collaboration among developers. As a CI/CD tool, it allows automated building, testing, deploying, delivering, and therefore releasing. This is done by instantiating GitLab runners, which are agents that physically execute the CI/CD jobs. Finally, it can also be used as a container and package repository to serve as an artifact repository for Docker images and libraries.

These complete set of features, in addition to the possibility of self-hosting the entire tool, make GitLab one of the most valid choices for repositories and CI/CD pipelines. However, in Table 5 some possible tools are provided.

Table 5: Alternative tools for GitLab.

Tool	Description
GitHub[26]	GitHub is a leading code hosting platform with powerful collaboration features and built-in CI/CD through GitHub Actions, making it the best choice for version control and automated workflows. The downside of GitHub is that cannot be self-hosted.
Bitbucket[27]	Bitbucket by Atlassian supports Git and Mercurial repositories. It offers robust code collaboration features and integrated CI/CD with Bitbucket Pipelines, providing a complete solution for development teams. Bitbucket Server is a self-hosted version of Bitbucket.
Jenkins[28]	Jenkins is a popular open-source automation server. While not a repository, it integrates seamlessly with Git and other VCSs to provide rich CI/CD capabilities for automating software development processes. Jenkins can be easily self-hosted inside an existing infrastructure.
Travis CI[29]	Travis CI is a CI/CD platform built for GitHub repositories. Configured through a simple YAML file, it automates builds and tests, and supports a range of programming languages for seamless integration. Travis CI is only available for a fee.

[26] https://github.com
[27] https://bitbucket.org
[28] https://www.jenkins.io
[29] https://www.travis-ci.com

Funded by the European Union

4.3.9. Loki

Loki is an open-source, horizontally scalable log aggregation system that is part of the Cloud Native Computing Foundation (CNCF) landscape. It is designed to handle and store log data generated by various applications and services in a distributed and efficient manner.

Loki is also part of the Grafana Labs, and therefore it is completely integrable with the Grafana Dashboard. However, in Table 6 are reported some possible alternatives to it.

Table 6: Alternative tools for Loki.

Tool	Description
Fluentd[30]	Fluentd is an open-source data collector that efficiently aggregates logs from multiple sources, parses them, and forwards them to different destinations using a unified logging approach. Its lightweight architecture and extensive plugin support enhance flexibility in log processing workflows. Fluentd is a graduated project of the Cloud Native Computing Foundation.
Logstash[31]	Logstash is a tool part of the ELK stack. Hence, it is an Elastic product for log processing. It allows you to collect, parse, and transform logs before passing them to Elasticsearch. It supports a wide variety of input sources and output destinations.
Apache Flume[32]	Apache Flume is a distributed and reliable service designed for collecting, aggregating, and transporting large amounts of log data. It is commonly used in conjunction with Apache Hadoop, making it suitable for big data environments. It is written in Java, and the latest version, 1.11.0, was released on October 10th, 2022.
Prometheus	Prometheus is best known for its metrics monitoring capabilities, but it can also do log collection and query with PromQL. It is particularly useful in environments where metrics and logs are closely integrated, providing a unified approach to monitoring and troubleshooting.

4.3.10. Grafana

Grafana is an open-source analytics and monitoring platform that excels in visualizing time-series data. It integrates with various data sources, including popular databases, cloud services, and monitoring systems, enabling users to create interactive and customizable dashboards. Grafana's rich set of features includes support for different chart

[30] https://www.fluentd.org
[31] https://www.elastic.co/logstash
[32] https://flume.apache.org

Funded by the European Union

types, alerting, and the ability to share dashboards, making it a widely adopted tool for monitoring and observability.

Inside the FLEXIndustries Platform, Grafana can also be exploited for its complete integration with Prometheus. In other words, it is very simple create a Grafana dashboard and select as data source for this GUI the Prometheus Service. However, in Table 7 some alternatives to Grafana are provided.

Table 7: Alternative tools for Grafana.

Tool	Description
Kibana[33]	Kibana is an open-source data visualization platform primarily used with the Elasticsearch stack. It provides powerful analytics and visualization capabilities.
Chronograf[34]	Chronograf is part of the TICK stack (Telegraf, InfluxDB, Chronograf, Kapacitor) and is designed to visualize and explore time series data.
Metabase[35]	Metabase is an open-source business intelligence tool that allows users to create charts and dashboards and provides an easy-to-use interface for exploring data.
Apache Superset[36]	Apache Superset is an open-source business intelligence web application that supports data exploration, visualization, and interactive dashboards.

4.3.11. Prometheus

Prometheus[37] is an open-source monitoring toolkit which ensures reliability and scalability in dynamic environments. It was originally developed by SoundCloud but is actually widely used in the cloud-native ecosystem. The tool collects time-series data, making it easy to query and visualize performance metrics across applications and infrastructure. Using a multi-dimensional model, it supports dynamic service discovery, robust querying, and rules-based alerting. Integration with container orchestration, especially Kubernetes, is seamless, often coupled with Grafana for visualization. With its adaptability, Prometheus is ideal for monitoring diverse systems and applications.

Within the FLEXIndustries platform, Prometheus is used in conjunction with Grafana to create a monitoring dashboard that aims to continuously analyze the status of the core components. In other words, the idea is to know at any moment the actual overload of the

[33] https://www.elastic.co/kibana
[34] https://www.influxdata.com/time-series-platform/chronograf
[35] https://www.metabase.com
[36] https://superset.apache.org
[37] https://prometheus.io

Funded by the European Union

platform by monitoring the percentage of completed tasks in relation to the scheduled and paused ones.

Table 8: Alternatives to Prometheus.

Tool	Description
Zabbix[38]	Zabbix is a flexible and scalable open-source monitoring solution that supports metrics, performance, and availability monitoring. Using a server-agent architecture, it provides auto-discovery, alerting, reporting, and a web-based interface for configuration and analysis in enterprise-class environments.
Datadog[39]	Datadog is a cloud-based monitoring platform built for modern cloud and microservices environments. It provides comprehensive infrastructure, application and log monitoring solutions with real-time dashboards, anomaly detection, APM and collaboration tools. It is widely used for holistic observability across complex, distributed systems but, a Monitoring-as-a-Service, it does not provide self-hosted versions.
Graphite[40]	Graphite is an open-source tool for real-time graphing and monitoring of time series data. It provides scalable tracking using the Carbon daemon, the Whisper database, and a web application. It is often integrated with other tools and is often used in larger monitoring solutions.

4.4. Process View

The main objective of this subchapter is to show the basic message exchange that happened between the core components and the user.
Hence, the process which is triggered when a FLEXIndustries user starts a new optimization pipeline is described in subchapter 4.4.1. Then, the interaction between the three core components is illustrated in subchapters 4.4.2 and 4.4.3.

[38] https://www.zabbix.com
[39] https://www.datadoghq.com
[40] https://grafana.com/oss/graphite

Funded by the European Union

4.4.1. User – Multi-Agent System Communication

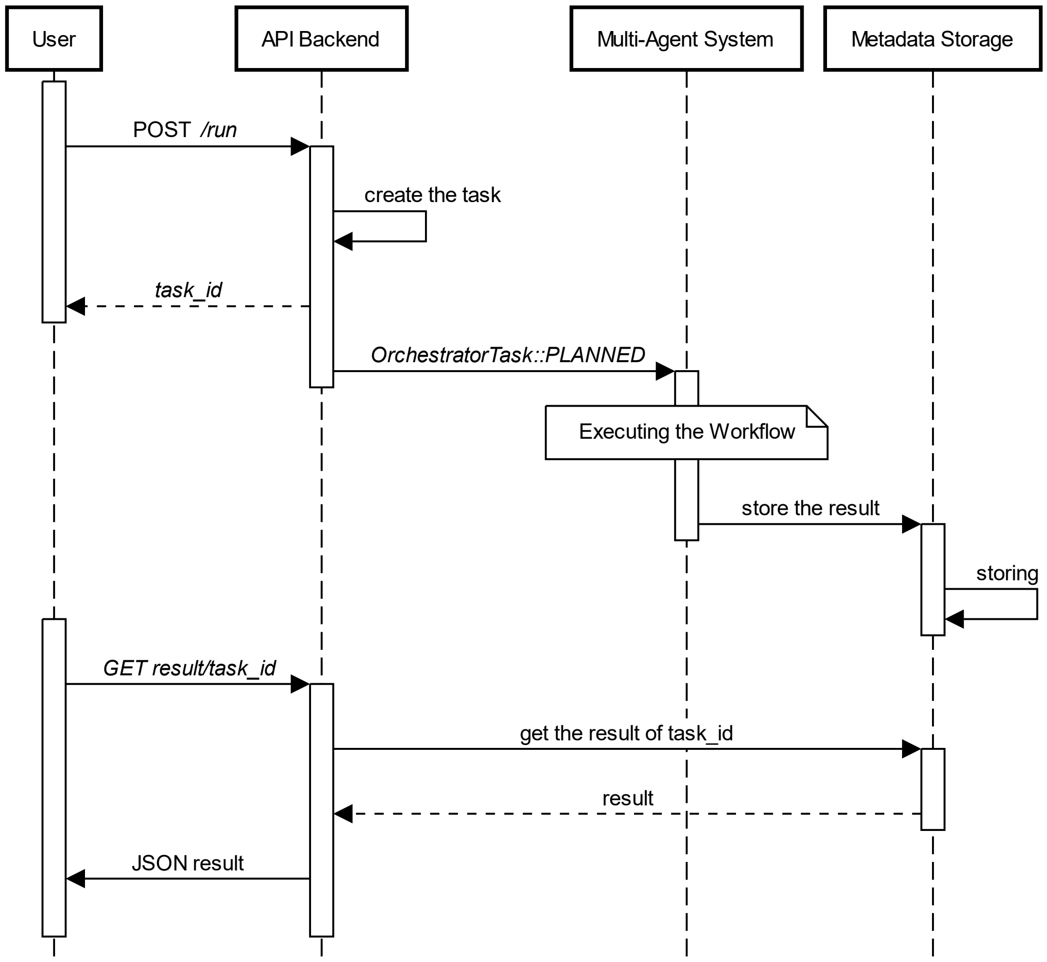


Figure 8: Communication between the User and the Multi-Agent System through the API Backend.

The interaction depicted in Figure 8 is the standard procedure that can be used by a generic user to interact with the Dynamic Energy and Process Management Platform, through the API Backend.

To start a new FLEXIndustries schedule, the user can send a *POST* request to the API backend. As already described in chapter 4.2.3.2, the user can also specify the workflow to be executed. At this point, the backend creates the task for the Multi-Agent System and inserts into the message an identifier generated using the *UUID4* standard specified in RFC4122. This *id* is also returned to the user so that it can check the status of the request.

The delivery of the Orchestrator Task from the API backend is completely asynchronous. This means that the task is sent to the communication backbone in such a way that the Multi-Agent System can process it whenever possible. As a result, the API server is not aware of the execution. In other words, from the backend's point of view, the task may or may not be handled by the Multi-Agent System. The downside is the completely decoupled way in which these core components interact with each other, which leads the platform architect to manage them separately, simplifying monitoring in the long run. Of course, this requires the user to keep track of the status of the task he or she has just created.

For this reason, the GET API shown in Figure 8 can be used to obtain the progress of the FLEXIndustries schedule.
For the API backend, the verification method is very simple. Since the Multi-Agent System publishes the result of the task in the Metadata Storage as soon as it is completed, if the result with an id equal to the previously quoted task id exists, it means that all is well. Otherwise, the task is still in progress, or some problem occurred during the workflow.

4.4.2. Multi-Agent System – Agent Communication

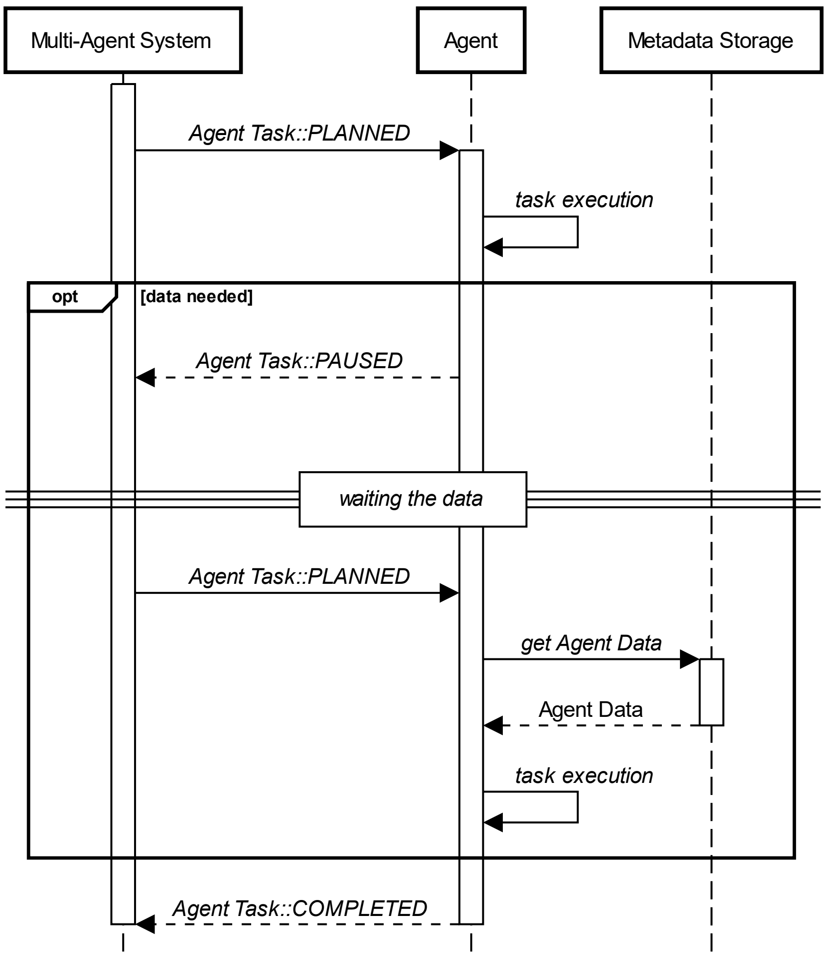


Figure 9: Communication between the Multi-Agent System and the Agent without and with optional data.

Once the Multi-Agent System receives an Orchestrator Task, it executes all the steps specified by the workflow selected by the user. This results in the creation of a sequence of Agent Tasks for the agents invoked by the specific workflow. If the task requested by the Multi-Agent System does not require any additional data, and since each agent is treated as an asynchronous stateless service, the data exchange between these two actors is simple and telegraphic. As shown in Figure 9, when this scenario occurs, the agent returns an Agent Task of type `COMPLETED` and publishes the result to the Metadata Storage. This makes it accessible from the API Backend, as shown in the previous subchapter.

Funded by the European Union

Instead, if new data is needed, the agent can simply pause the current task for the missing information. This operation is encoded by the Agent Task of type `PAUSED`, which is returned to the Multi-Agent System by the contacted agent. In addition, the agent can specify in the message the data source to be contacted to obtain the requested values. Once this procedure is complete, the Multi-Agent System tries to send a new agent task of type `PLANNED`, hoping that the agent will now be able to complete its homework. If all goes well, the agent publishes the result within the metadata store and returns an agent task of type `COMPLETED`.

4.4.3. Multi-Agent System – Data Handler Communication

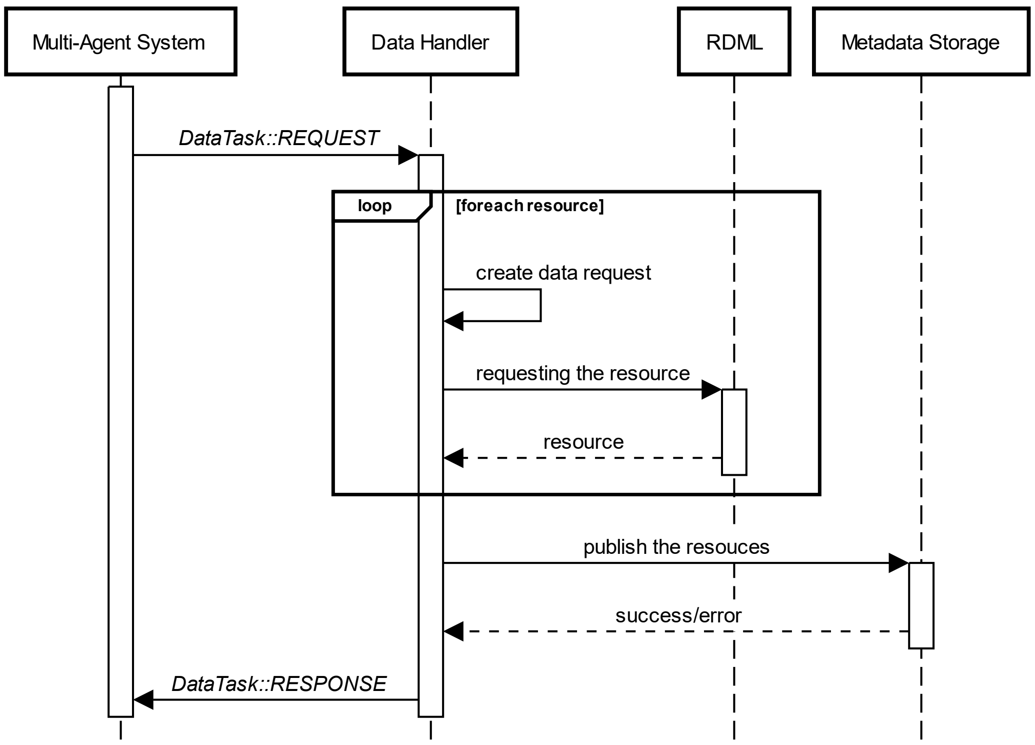


Figure 10: Communication between the Multi-Agent System and the Data Handler, with the Resources and Data Management Layer (RDML).

As anticipated in the previous sequence diagram, if the agent feels compelled to request some data, it suspends its task and communicates the suspension to the Multi-Agent System.
As soon as the orchestrator receives such a message from the agent, it creates a Data Task of type `REQUEST` starting from the data sources specified by the agent, as shown in Figure 10. Then, for each of these resources, the Data Handler makes a separate request to the Resource and Data Management Layer (RDML) developed in Task 4.1. Finally, all partial results are merged by the Data Handler and published to the Metadata Storage. If the write was successful, the Data Handler returns a Data Task of type `RESPONSE` to the Multi-Agent System.

This message is important to the orchestrator because it is, in a sense, the confirmation that the data has been successfully retrieved from the RDML component and that the result has been published to the platform cache. The adjective temporary is due to the

Funded by the European Union

expiration time assigned to this data, which is fixed at 60 seconds. Thus, the Multi-Agent System and the agent that requested the data have 60 seconds to resume the previously paused planning. Otherwise, all the steps just described and shown in Figure 10 must be repeated.

4.5. Development View

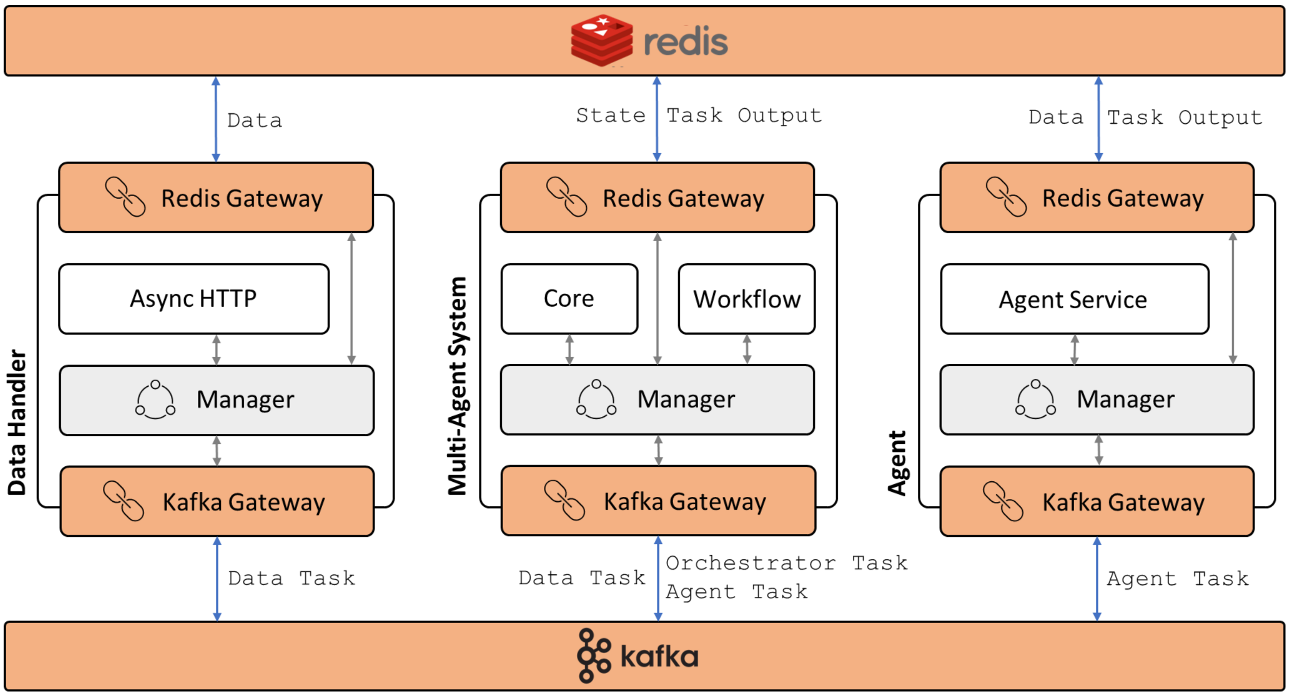


Figure 11: Development View

The previous section presented the communication flow between the core components that make up the FLEXIndustries platform.
This communication between these three modules is guaranteed by a set of interfaces, managers and wrappers that aim at abstracting the underlying data exchange through the logic that governs the functioning of each core component.
Figure 11 shows the composition of the data handler, the Multi-Agent System, and the agent components.

The communication backbone at the bottom manages the control event messages, while the Redis service at the top is responsible for caching results and temporary data, as explained in the previous chapters. Therefore, each core component is equipped with two communication interfaces: one for Redis and one for Kafka. Their names are Redis Gateway and Kafka Gateway, respectively. These two blocks are responsible for serializing and deserializing the messages coming from and going directly to the aforementioned data channels. Therefore, these components also implement the platform's common data model[41]. As a result, and as shown in Figure 11, the Data Handler

[41] In deliverable D2.5 this data model is referred as the Orchestrator Data Model. In reality, it contains the format of all the kind of messages that are sent and received by the Multi-Agent System.

Funded by the European Union

is only able to exchange Data Task with Kafka and Data messages with Redis. The Multi-Agent System, on the other hand, can exchange Data, Orchestrator, and Agent Task with Kafka and uses Redis to read the output published by each agent and as a persistent storage for its state. Finally, the Agent can only exchange Agent Task with Kafka, but is able to read the Data messages written by the Data Handler and write the Task output to and from Redis.

Each message received from the different gateways is then handled by the managers. This module organizes the execution of each operation within the core component. In other words, it practically implements the sequences of operations already illustrated in Figure 8, Figure 9, and Figure 10.

Finally, each core component is equipped with some core code that is used to distinguish it from others.

In Figure 11, the Data Handler contains an Async HTTP block, the Multi-Agent System contains a Core and a Workflow block, while the Agent contains an Agent Service.

The Async HTTP is the implementation of an async HTTP client used to interact with the APIs exported by the Resource and Data Management Layer components described in D4.5. If the endpoints of this module change in the future, this core code will change accordingly[42].

The core implements the entire logic of the Multi-Agent System, such as the management of the workflow, the decoupling management of the messages coming from Kafka, the pausing and restarting of a paused agent task, and the publishing of data tasks for the data handler.

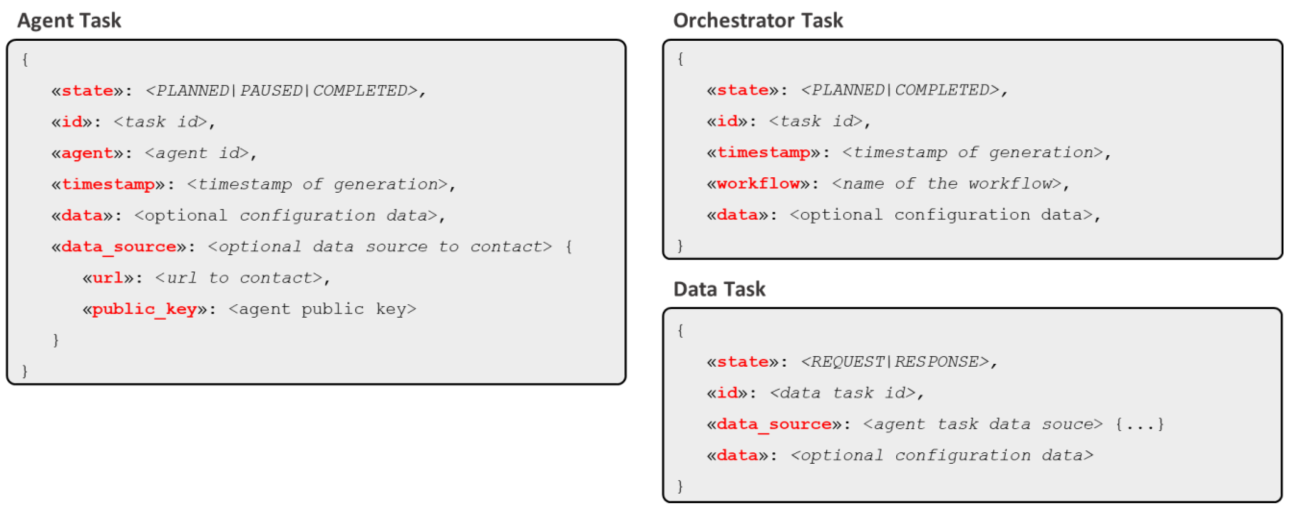
The workflow is the Python program used to characterize a FLEXIndustries pipeline.

The agent service is basically the code that uniquely identifies the operations performed by the agent. It is therefore the code of the WP4 and WP5 modules developed by the partners involved. As you can see, this code can be developed in isolation by the partners and only integrated with the agent wrapper at the end. In this way, this procedure is completely transparent to the partner since the activity is guaranteed by the surrounding functions.

[42] For example, another popular protocol for communicating with other microservices is gRPC, which is also faster with less overhead because it parses messages using protocol buffers rather than JSON as HTTP.

Funded by the European Union

4.5.1. Data Model



Agent Task

```
{
   «state»: <PLANNED|PAUSED|COMPLETED>,
   «id»: <task id>,
   «agent»: <agent id>,
   «timestamp»: <timestamp of generation>,
   «data»: <optional configuration data>,
   «data_source»: <optional data source to contact> {
      «url»: <url to contact>,
      «public_key»: <agent public key>
   }
}
```

Orchestrator Task

```
{
   «state»: <PLANNED|COMPLETED>,
   «id»: <task id>,
   «timestamp»: <timestamp of generation>,
   «workflow»: <name of the workflow>,
   «data»: <optional configuration data>,
}
```

Data Task

```
{
   «state»: <REQUEST|RESPONSE>,
   «id»: <data task id>,
   «data_source»: <agent task data souce> {...}
   «data»: <optional configuration data>
}
```

Figure 12: Data Model adopted for the communication between the core components.

In Figure 12: Data Model adopted for the communication between the core components. are reported the data models of each kind of message sent over the communication backbone. As already described in subchapters 4.3.3, 4.4, and 4.5.3, the Agent Task contains information about the *id* of the agent called to perform the activity, the optional input data required to execute the task, and, in the case of a data request, the pointer to the data source that the data handler must contact. The Orchestrator Task is extremely closed, since the only information required to start a new pipeline is the name of the workflow to be executed and the optional input data. Finally, the Data Task contains only the copy of the data source contained in the Agent Task and some optional configuration data.

4.5.2. Workflow Example

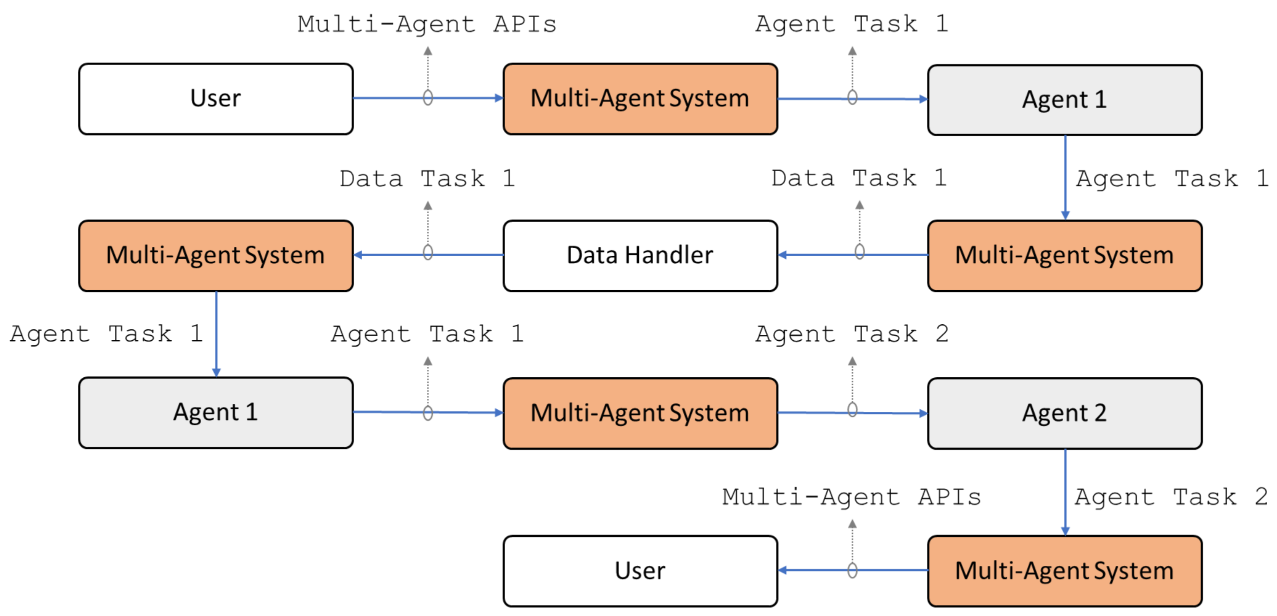


Figure 13: Example of a possible simple workflow that can be requested to the Multi-Agent System.

Funded by the European Union

An example of a classical and demonstrating workflow that can be requested to the Multi-Agent System is reported in Figure 13. This demo set-up includes the presence of one Multi-Agent System, of one Data Handler, and of two agents: Agent 1 and Agent 2.
The flow starts from the *POST* API, which is exploited by a potential user to create a new Orchestrator Task, as described in 4.2.3.2. Then, the Multi-Agent System create an Agent Task for the Agent 1, which however is unable to complete its operations due to some missing data. Hence, the Multi-Agent System create a Data Task for the Data Handler and another Agent Task for the Agent 2.
Finally, the agents can complete the activities and the overall outcome can be returned to the user through the *GET* API.

```python
from core.orchestration import execute, finish, get_result, execute_and_result
from core.agent import Agent
from core.model import TaskRequest, OrchestratorTask

async def transform1(data):
    data["transform1"] = "transform1"

async def run(request: OrchestratorTask):

    module1 = Agent(id="agent-1")
    module2 = Agent(id="agent-2")

    task1 = TaskRequest(
        agent=module1,
        data={"hello": "world"}
    )

    await execute(task1)
    res1 = await get_result(task1)
    await transform1(res1.data)

    task2 = TaskRequest(
        agent=module2,
        data={}
    )
    res2 = await execute_and_result(task2)

    await finish(request, [res1.data, res2.data])
```

Figure 14: Coding implementation of the workflow of Figure 13.

The illustration of Figure 13 can be represented by code using Python in Figure 14. What is extremely important is to notice is the importing of four functions in the first row: `execute`, `finish`, `get_result`, and `execute_and_result`.
These can be thought of as library functions provided by the Multi-Agent System to programmatically describe a FLEXIndustries workflow.
Once the agents and the task are defined using the Agent and TaskRequest data classes, the `execute` method sends an Agent Task over the communication backbone. The optional data requirements are abstracted and handled autonomously by the Multi-Agent System itself. Therefore, it is not necessary to specify this action in the workflow. After all,

Funded by the European Union

it is not the task of the author of this simple program to handle such situations. To use a metaphor, it would be as if the programmer of a high-level language were responsible for managing low-level resources, timings, and hardware. The `get_result` method waits for the results of the tasks to be published within the Metadata Storage. The `execute_and_result` method is just a combination of the two former methods, while the `finish` method determines the end of the workflow and the publication of the aggregated results within the Metadata Storage. At this point, it is possible for the user to retrieve these results using the *GET* API mentioned above.

4.5.3. Asynchronous Paradigm

These executions are completed asynchronous and, therefore, they do not occupy the CPU time for all the time. The gateways and the managers are developed using the asyncio[43] python library. Hence, the asynchronous python loop is responsible of handling the execution of all the asynchronous functions which compose the core components. This is extremely critical in low-latency and high-throughput architectures.
In particular, the strength of asynchronous event handling lies in the lack of waiting. Any interruptions in the code due to waiting of various kinds[44] are temporarily skipped to free up time for handling other functions. In this way, it is actually possible to avoid wasting CPU cycles and thus time without having to spend it waiting for the result of a particular function. Consequently, by eliminating these waits, it is possible to reduce execution time, or alternatively, it is possible to perform more operations per second, thus increasing throughput and decreasing response latency.

4.6. Roles of involved partners

In Table 9, there is a first approach at listing the concrete inputs that we would need from the partners involved in T2.5.

Table 9. Contributions from involved partners

Partner	Related Task(s)	Contribution Needed
CERTH	T5.4	Provide more details about the near real-time constraints of the software to be developed in T5.4
		Confirm if the proposed architecture is compatible with the technology to be used for the development of their ‘online optimal process energy management’ software.

[43] https://docs.python.org/3/library/asyncio.html
[44] Normally due to CPU-intensive and IO-intensive tasks.

Funded by the European Union

CAR	T2.4	Verify the architecture design against the requirements and functionalities specified in D2.5 (outcome of T2.4).
	T5.1	Confirm integration features with the platform for the DSS module developed in T5.1, regarding the process, material and energy related data analysis and forecast.
CIRCE	T2.3	Ensure that the platform meets the needs of energy and process management identified during the analysis of data and technologies in D2.3 (outcome of T2.3).
UoP	T5.3	Confirm if the proposed architecture is compatible with the technology/approach to be used in developing the energy load/price forecasting modules of T5.3.
SIM	T4.2	Confirm if the proposed architecture is compatible with the technology/approach of their Simulation Software.
	T4.4	Confirm if the proposed architecture is compatible with their plan for implementing the Digital Twin Frontend infrastructure and SDK.
STAM		Give support in the definition of the platform according to their role in the project.
FORD		Give support in the definition of the platform according to their role in the project.
MIL		Give support in the definition of the platform according to their role in the project.
THD		Give support in the definition of the platform according to their role in the project.
PRO		Give support in the definition of the platform according to their role in the project.
TITAN		Give support in the definition of the platform according to their role in the project.
INTEC		Give support in the definition of the platform according to their role in the project.

Funded by the European Union

FBK	T5.2	Confirm if the proposed architecture captures correctly the role of the offline process & energy planning digital tool
TUB	T4.1	Confirm if the proposed architecture captures correctly the role of the Resources and Data Management Layer.

In Figure 15, a schema illustrates the component of the platform in which the partners are involved. Consequently, this allows to understand the point of contact between T2.5, led by LINKS, and the other tasks along with their respective task leaders.
All the blocks with an orange background correspond to the component designed and defined in T2.5, those in grey correspond to the components that will be developed in WP4, and finally the white ones to the components that will be developed in WP5.

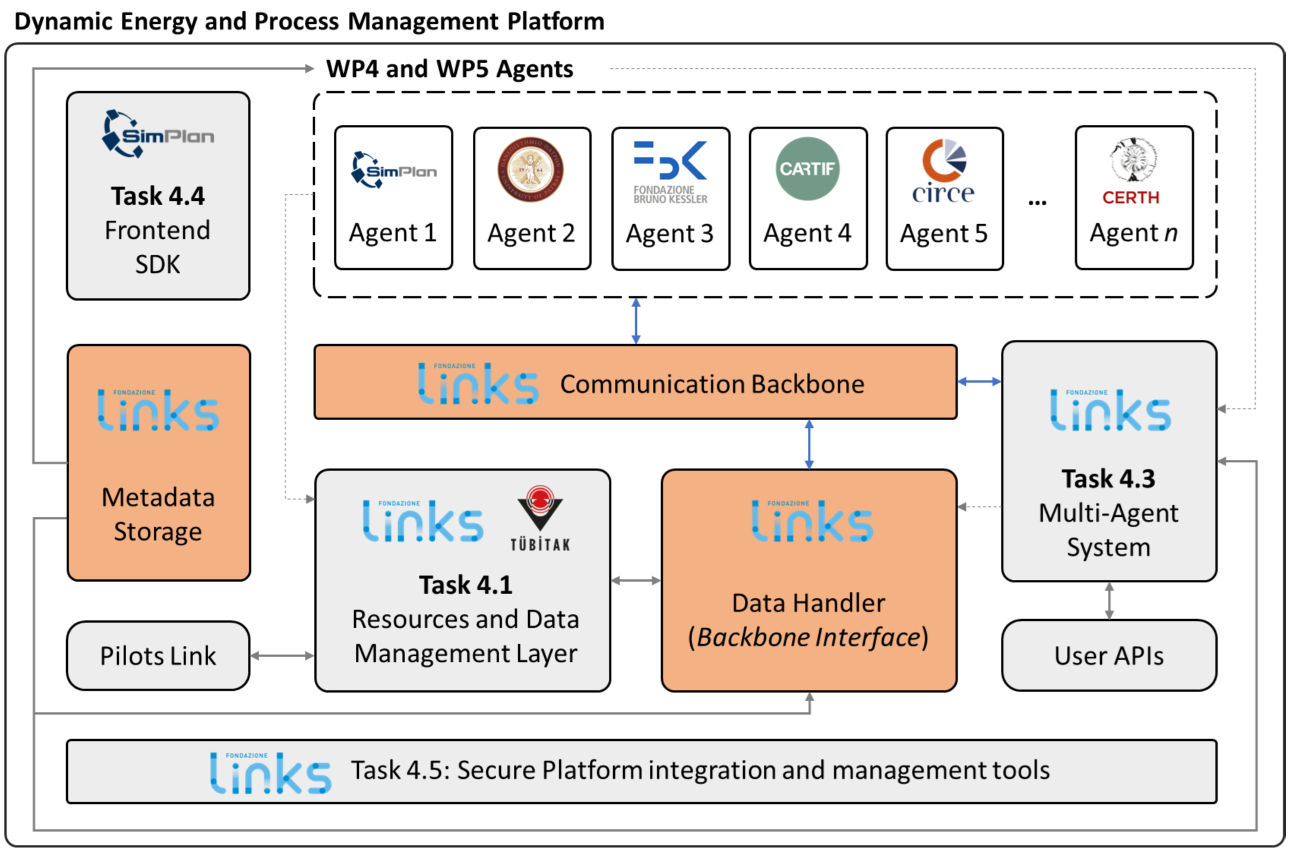


Figure 15: Partner View.

In summary, the main objectives of the partners involved consist in defining the interface and the integration mechanism of their component with the platform, ensuring readiness for the subsequent development of the platform.

5. Conclusions

The Dynamic Energy and Process Management platform is a complex canvas composed of multiple efforts from Work Package 4 and Work Package 5.

This composition requires some adjustments to label Task 2.5 as an integrator of multiple artifacts that attempt to solve multiple goals and return multiple outcomes. The disadvantage is that the requirements of each player must be taken into account in this operation.

In fact, some specific implementation needs have been found and identified. It is obvious that these needs depend on the way all agents have to communicate and cooperate. The Multi-Agent System and the communication backbone will certainly manage heavy data flows between the services composing the architecture. In this scenario, the scaling and the corresponding availability must be as responsive as possible.

Finally, some tools have been identified that can be used to ensure that these requirements are met. The next steps will include a beta version of the platform, including the security constraints that make access to the FLEXIndustries portal safe for users and data providers.

Funded by the European Union

References

[1] “ISO/IEC/IEEE 42010:2011.” Accessed: Jan. 31, 2024. [Online]. Available: https://www.iso.org/standard/50508.html

[2] “IEEE Recommended Practice for Architectural Description for Software-Intensive Systems.” Accessed: Jan. 31, 2024. [Online]. Available: https://standards.ieee.org/ieee/1471/2187/

[3] P. B. Kruchten, “The 4+1 View Model of architecture,” *IEEE Softw*, vol. 12, no. 6, pp. 42–50, 1995, doi: 10.1109/52.469759.

[4] Nick Rosanski and Eoin Woods, “Software Systems Architectures,” 2012.

Funded by the European Union