



D4.7 Implementation of Advanced CPS-IoT RSM Features v3

Deliverable No.	D4.7	Due Date	29/02/2024
Description	Description of advanced features of the ODIN platform, specifically the Digital Ledger Technologies, for trustworthy Resource Federation, and the Resource Choreographer, for resource orchestration in use case workflows.		
Type	Report	Dissemination Level	PU
Work Package No.	WP4	Work Package Title	CPS-IoT Resource Management System
Version	1.0	Status	Final



Authors

Name and surname	Partner name	e-mail
Anastasia Blitsi	CERTH	akblitsi@iti.gr
Vasileios Lolis	CERTH	vaslwlis@iti.gr
Tsiobra Maria	CERTH	mtsiobra@iti.gr
Gerovasilis Georgios	CERTH	ggerovasilis@iti.gr
Grypiotis Ioannis	CERTH	igrypiotis@iti.gr
Diamantakis Christos	CERTH	ch.diamantakis@iti.gr
Konstantinos Votis	CERTH	kvotis@iti.gr
Pilar Sala	MYS	psala@mysphera.com

History

Date	Version	Change
10/01/2024	0.1	Table of Contents
15/02/2024	0.2	First Draft
27/02/2024	0.3	Second Draft – content addition
29/02/2024	0.4	Version ready for peer review
26/03/2024	0.5	Peer review feedback incorporated
28/03/2024	0.6	Technical review feedback incorporated
29/05/2024	0.7	Version ready for quality review
03/07/2024	1.0	Final Version

Key data

Keywords	Digital Ledger Technologies, blockchain, resource federation, resource choreography, resource workflow design
Lead Editor	Anastasia Blitsi

Internal Reviewer(s)

UoW, UCBM, M&S

Abstract

This document provides an in-depth exploration of two crucial elements within the ODIN healthcare platform: Digital Ledger Technologies (DLT) and the Resource Choreographer. The integration of Fabric Network in DLT introduces dynamic event handling, enhancing real-time communication and synchronization. The Resource Federation architecture, powered by smart contracts, emphasizes transparency and security in healthcare data exchanges. Simultaneously, the Resource Choreographer undergoes a transformative journey, evolving into a cluster of microservices for improved scalability, fault tolerance, and integration capabilities with emerging technologies. Utilizing Kogito, a cloud-native business automation technology, the Resource Choreographer showcases its flexibility through Serverless Workflow models, supporting low-code service creation, Kafka integration, and external API interactions. Various healthcare use cases illustrate the Resource Choreographer's adaptability, from orchestrating AI-driven rehabilitation sessions to guiding patients via Robot Receptionists. The document also explores Documentation Component and API Gateway workflows, shedding light on their potential contributions to platform documentation and API management.

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of contents

- TABLE OF CONTENTS 4
- LIST OF TABLES 5
- TABLE OF FIGURES 6
- ACRONYM GLOSSARY 7
- 1 ABOUT THIS DELIVERABLE..... 9
 - 1.1 DELIVERABLE CONTEXT 9
- 2 INTRODUCTION 10
- 3 RESOURCE FEDERATION 11
 - 3.1 RESOURCE FEDERATION ARCHITECTURE 11
 - 3.1.1 *Connectivity in Architecture*..... 11
 - 3.1.2 *DLT API*..... 11
 - 3.1.3 *Blockchain*..... 12
 - 3.2 DEVIATIONS AND CORRECTIVE ACTIONS 14
 - 3.2.1 *Data persistence*..... 14
 - 3.2.2 *Communication* 14
 - 3.3 UPDATED TECHNOLOGIES: RECENT ADDITIONS AND ENHANCEMENTS 14
 - 3.4 USE CASES 15
 - 3.4.1 *Use Case for Federated Learning*..... 15
 - 3.4.2 *Use Case for Health Inventory Traceability framework based on blockchain*..... 16
- 4 RESOURCE CHOREOGRAPHER..... 18
 - 4.1 EVOLUTION OF THE COMPONENT 18
 - 4.2 MAIN TECHNOLOGIES & IMPLEMENTATION PROCESS 18
 - 4.2.1 *Kogito*..... 18
 - 4.2.2 *How to design and implement a workflow* 19
 - 4.2.3 *Architecture of the workflow* 20
 - 4.3 USE CASES 22
 - 4.3.1 *Robot Receptionist*..... 27
 - 4.4 DOCUMENTATION COMPONENT WORKFLOW 27
 - 4.5 API GATEWAY WORKFLOW 27
 - 4.6 OTHER FEATURES 28
- 5 CONCLUSION 29

List of tables

TABLE 1. DELIVERABLE CONTEXT.....	9
TABLE 2 INPUT EXAMPLE FOR HIT COMPONENT	13
TABLE 3 USE CASES ANALYSIS	23

Table of figures

FIGURE 1 RESOURCE FEDERATION CONNECTION.....	11
FIGURE 2 REPRESENTATION OF BLOCKCHAIN.....	13
FIGURE 3 RESOURCE FEDERATION WORKFLOW.....	16
FIGURE 4 INTEGRATION OF HIT WITH ODIN PLATFORM.....	17
FIGURE 5 RUC A2 UC4 COMPONENT AND DATA FLOW DIAGRAM.....	19
FIGURE 6 BASIC ARCHITECTURE.....	21
FIGURE 7 EXAMPLE COMPLEX WORKFLOW.....	22
FIGURE 8 REHABILITATION AI WORKFLOW ARCHITECTURE.....	26
FIGURE 9 KOGITO WORKFLOW DESIGNER TOOL.....	28

Acronym Glossary

Acronym	Definition
AI	Artificial Intelligence
API	Application Programming Interface
BFT	Byzantine Fault Tolerance
BMN	Business Modeling Notation
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
BRE	Business Rule Engine
BRMS	Business Rules Management System
CA	Certificate Authority
CPS	Cyber Physical System
dApp	Decentralized Application
DLT	Digital Ledger Technologies
DMN	Decision Model and Notation
ESB	Enterprise Service Bus
EVM	Ethereum Virtual Machine
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HIT	Health Inventory Traceability framework
IDE	Integrated Development Environment
IoT	Internet of Things
JMX	Java Management eXtensions
JVM	Java Virtual Machine
KER	Key Enabling Resource
OAUTH	Open Authorization
OMG	Object Management Group
RC	Resource Choreographer
REST	Representational State Transfer

RMS	Resource Management System
UI	User Interface
UUID	unique identifier
XML	eXtensible Markup Language
WS	Workflow Service

1 About this deliverable

This document is the third and final edition and delves into two vital elements of the ODIN healthcare platform: Digital Ledger Technologies (DLT) and the Resource Choreographer. It explains how Fabric Network enhances real-time communication in DLT and discusses the security aspects of Resource Federation. The Resource Choreographer evolves into microservices for scalability and integrates with Kogito for flexibility. Use cases highlight its adaptability in healthcare, while also mentioning Documentation Component and API Gateway workflows.

The final architecture and updates about the final version of Resource Federation and Resource Choreographer are outlined in this deliverable.

This document is linked with D4.1 (CPS IoT Resource Management System Specification), D4.2, D4.3, D4.4 (Implementation of Local CPS-IoT RSM Features (version 1-3) and D4.5, D4.6 (Implementation of Advanced CPS-IoT RSM Features version 1 and 2).

1.1 Deliverable Context

Table 1. Deliverable context

PROJECT ITEM IN THE DOA	RELATIONSHIP
Project Objectives	The final architecture and updates about the Resource Federation and the Resource Choreographer.
Exploitable results	All components described in the current deliverable are potential exploitable results.
Workplan	D4.7 is attributed to the tasks of WP4, Resource Federation and Resource Choreographer.
Milestones	D4.7 is key deliverable for the description of WP4 components
Deliverables	<ul style="list-style-type: none"> • D4.1 CPS IoT Resource Management System Specification • D4.2 Implementation of Local CPS-IoT RSM Features version 1 • D4.3 Implementation of Local CPS-IoT RSM Features version 2 • D4.4 Implementation of Local CPS-IoT RSM Features version 3 • D4.5 Implementation of Advanced CPS-IoT RSM Features version 1 • D4.6 Implementation of Advanced CPS-IoT RSM Features version 2
Risks	Risk of time-consuming integration due to multiple technologies used (by providing means for resource abstraction, facilitating communication between diverse resources).

2 Introduction

This document delves into the functionality of two component that are crucial for the performance and effectiveness of ODIN platform: Resource Federation and the Resource Choreographer. These components are essential for making the platform flexible and responsive:

Resource Federation: The ODIN platform integrates Resource Federation to establish a resilient and secure infrastructure. [Section 3](#) of this document navigates through the enhanced features and recent advancements in Digital Ledger Technologies (DLT). Noteworthy is the incorporation of Fabric Network into Resource Federation, introducing a dynamic and efficient means of event-driven communication. The real-time event-driven architecture, powered by Fabric's event mechanism, synchronizes the platform with the latest updates, ensuring seamless integration with the broader network ecosystem. Within this context, blockchain events within the Fabric Network serve as vital triggers, representing significant transactions. The implementation strategically incorporates blockchain event listeners that actively monitor and interpret emitted blockchain events, allowing the system to respond promptly and appropriately. This event-driven paradigm enhances system responsiveness, adaptability, and synchronization with the Fabric Network.

The document further explores the architecture of Resource Federation, emphasizing connectivity of components such as the blockchain and DLT API, which is integrated into the ODIN platform. This section provides insights into the communication flow, transaction recording, and the role of smart contracts within the blockchain component. Smart contracts for Access, Resource, and Open caller's Health Inventory Traceability framework (HIT) components (more information about the HIT component can be found in [section 3.4.2](#)) ensure transparency, accountability, and security in the ODIN ecosystem, fostering trust and integrity in data and resource exchanges. The implementation details include mechanisms for data persistence and communication, addressing potential disruptions and ensuring a dependable platform. Two compelling use cases, a use case for Federated Learning and a use case for HIT, illustrate the practical application and advantages of the Resource Federation in real-world scenarios.

Resource Choreographer: The Resource Choreographer takes centre stage as a service orchestrator in the ODIN platform. [Section 4](#) unfolds the evolution and implementation process of this dynamic component, emphasizing its role in orchestrating interactions between Key Enabling Resources (KER) and services. Originally conceived as a singular component, the Resource Choreographer evolved into a swarm of microservices. This evolution enhances scalability, eliminates single points of failure and allows for the seamless integration of new technologies. The integration with Kogito, a cloud-native business automation technology, forms the backbone of the Resource Choreographer's implementation. Leveraging Serverless Workflow models, the Resource Choreographer enables the creation of services with minimal code, seamless integration with Apache Kafka for event-driven communication and interaction with external services through API calls. Diverse use cases exemplify the Resource Choreographer's versatility. From orchestrating rehabilitation sessions with AI analysis to guiding patients with QR codes using Robot Receptionists, the Resource Choreographer demonstrates its adaptability in diverse healthcare scenarios. Additionally, the document sheds light on the Documentation Component and API Gateway workflows, showcasing their potential impact on platform documentation and API management.

Together, Digital Ledger Technologies and the Resource Choreographer facilitate secure, adaptive, and efficient healthcare solutions. The integration of advanced technologies and dynamic service orchestration ensures that ODIN remains at the forefront of innovation in distributed ledger systems and healthcare service delivery.

3 Resource Federation

3.1 Resource Federation Architecture

3.1.1 Connectivity in Architecture

The Resource Federation comprises two main components:

- the blockchain,
- the DLT API.

As illustrated in [Figure 1](#), the component within the ODIN platform, referred to for convenience as the Resource Federation, is connected to the instance's Enterprise Service Bus (ESB). It is linked to a specific topic on the ESB and receives information (e.g. requests, messages) that needs distribution via the blockchain (the blockchain will be located in an ODIN server at CERTH as described in D4.6 “Implementation of Advanced CPS-IoT RSM Features v2”).

The Resource Federation processes this information and submits a transaction to the blockchain. Acting as middleware, the DLT API (Resource Federation) serves as the client API of the blockchain, enabling the submission of transactions.

Subsequently, the chaincode of the blockchain (more information can be found in D4.6 “Implementation of Advanced CPS-IoT RSM Features v2”) emits a blockchain event (more information about this kind of event can be found in [section 3.3](#)) and disseminates this event. All the ODIN instances, that are connected to the blockchain through their Resource Federation (which is a DLT API), will have access to the information.

In each ODIN instance, the Resource Federation also includes a blockchain listener that monitors every blockchain event created. This allows the Resource Federation in another instance to be aware when a request is initiated. Further details about the blockchain event listeners and this workflow can be found in [section 3.3](#) and [section 3.4.1](#) accordingly.

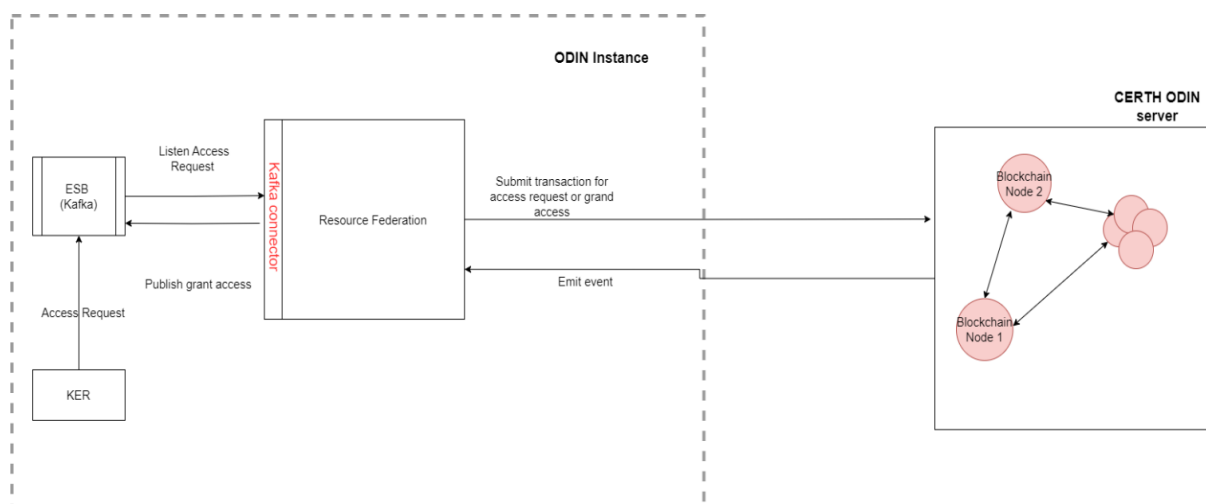


Figure 1 Resource Federation connection

3.1.2 DLT API

The Resource Federation component embedded within the ODIN platform plays a crucial role in managing and processing ESB events and blockchain events originating from the blockchain ecosystem. Its primary function is to receive and handle all blockchain events emitted within the blockchain network. Acting as a central hub for blockchain event monitoring, the component

functions as a DLT API, ensuring seamless real-time access to all blockchain events. Upon receiving these blockchain events, the Resource Federation component initiates a series of procedures to determine the appropriate course of action.

In addition to the functionalities described above, the Resource Federation leverages a Kafka Node.js component to further enhance ESB event processing and management capabilities. Serving as a consumer, the Kafka Node.js integrates with the ESB, specifically listening to a designated topic where requests are posted. The Resource Federation component receives this data and emits blockchain events to the blockchain network.

Furthermore, a pivotal step involves querying a configuration file to determine the federated components within the ODIN instance to which it belongs. This examination helps in identifying if the received blockchain event requires access to a federated component within the same instance. In cases where the blockchain event indeed requires interaction with a federated component, the DLT API component proceeds to trigger a transaction on the blockchain, including vital information such as the IP address of the component requiring access.

Once the necessary data is transmitted to the DLT API, a transaction is generated on the blockchain network. This transaction encapsulates the request and serves as a formalized channel for communication between different ODIN instances within the ODIN platform. The transaction created on the blockchain network is not restricted to the originating ODIN instance alone. Instead, it is transmitted across the blockchain infrastructure, reaching the DLT API component located in the targeted ODIN instance, which initiated the blockchain event initially. This seamless transmission ensures efficient information exchange and facilitates the synchronization of federated resources across ODIN instances within the ODIN ecosystem.

Beyond its role as middleware, the DLT API features an essential component known as the blockchain event listener. This blockchain listener constantly monitors the blockchain for any transaction-related activity. When a transaction initiation request is received from the Resource Federation component, the blockchain event listener analyzes the transaction details and transmits them to the blockchain network. This integration of the blockchain event listener ensures that every transaction is promptly registered and processed within the decentralized ledger ecosystem, enhancing the reliability and transparency of the entire transactional process.

Additionally, the DLT API sends the DLT block, where the transaction proposals are then shared with peers chosen by organizations according to the details outlined in the smart contract endorsement policy (more information can be found in D4.6 “Implementation of Advanced CPS-IoT RSM Features v2”). The transaction proposal serves as input for the smart contract, generating an endorsed transaction response, which the peer node then returns to the DLT API.

These transaction responses are combined with the transaction proposal to create a fully endorsed transaction, ready for distribution across the entire network. The blockchain comprises three smart contracts, which are described in the following paragraph.

3.1.3 Blockchain

The Smart Contracts (more information can be found in D4.6 “Implementation of Advanced CPS-IoT RSM Features v2”) within the blockchain component are pivotal for ensuring transparency, accountability, and security in the ODIN ecosystem. The smart contracts that interact with the ODIN platform can, also, emit blockchain events when a transaction is occurred.

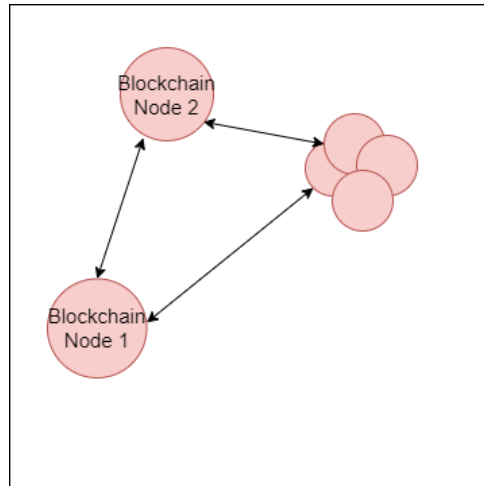


Figure 2 Representation of blockchain

There are three smart contracts:

- Access component smart contract:** governed by its dedicated smart contract, logs significant actions during data exchange among ODIN stakeholders. This encompasses the recording of access requests and security events, creating an auditable trail for non-repudiation and conflict resolution. The blockchain ledger captures requests to access components, capturing both the initial request and the subsequent access transaction. An illustrative JSON input details sender and receiver IDs, the nature of the request (e.g., "GPU"), and a timestamp. If needed, the DLT API can facilitate filtering options.
- Resource component smart contract:** managed by its own smart contract, maintains a record of key activities during resource exchanges. It carefully logs resource requests, fostering a transparent and accountable environment. Transactions involving resource sharing, like a department seeking access to another's resource, are permanently recorded in the blockchain ledger. The JSON input for resource requests includes sender and receiver IDs, the specific resource (e.g., "lorazepam"), and a timestamp. This robust system ensures a history of resource interactions, allowing stakeholders to provide digital evidence and efficiently resolve issues. This smart contract serves as foundational element in strengthening the ODIN blockchain, promoting trust and integrity in data and resource exchanges.
- Health Inventory Traceability (HIT) component smart contract:** assumes the role of a meticulous recorder for crucial actions during health data exchange among stakeholders. Its primary objective is to ensure transparency, auditability, non-repudiation, and accountability for actions occurring within the health information ecosystem.

A representative example of the input the HIT Component receives, formatted in JSON, is as follows:

Table 2 Input example for HIT Component

Device_id:	XYZ123
Type_of_event:	Maintenance

Location:	Surgery
Ownership:	HIT
timestamp:	1675418872

This structured input captures key details, including the device ID, type of event, location, ownership specifics, and timestamp. More details about the use of this smart contract are in [section 3.4](#).

3.2 Deviations and corrective actions

3.2.1 Data persistence

Within the DLT API component, serving as a mediator to the blockchain, and throughout the blockchain infrastructure, a mechanism for data persistence has been implemented to address scenarios involving machine restarts. When confronted with a machine restart, the system's essential data is preserved, and the integrity of the blockchain is maintained. Through the orchestration of this passive data persistence mechanism, an uninterrupted user experience is ensured, solidifying the system's reputation as a dependable platform in the presence of potential disruptions.

3.2.2 Communication

In the previous deliverable (D4.6 “Implementation of Advanced CPS-IoT RSM Features v2”), the Resource Federation was outlined, emphasizing the necessity for other components to send HTTP requests to interact with it, due to its external positioning to the ODIN platform. However, owing to the platform's constraints on direct communication with external components, a strategic decision was made. To address this limitation, the DLT API was restructured to be part of the ODIN platform, serving as an intermediary between the instances' components and the blockchain. A detailed description of this crucial intermediary component can be found in [section 3.3](#). This adjustment ensures a more integrated communication flow within the ODIN ecosystem while maintaining compliance with platform restrictions.

3.3 Updated Technologies: Recent Additions and Enhancements

Hyperledger Fabric Network is a permissioned blockchain framework implementation and was described in deliverable D4.6 “Implementation of Advanced CPS-IoT RSM Features v2”. The difference to the Resource Federation implementation from the deliverable D4.6 is the addition of the event listener. In Hyperledger Fabric Network, an event listener is a component or mechanism that allows applications to subscribe to and receive notifications about specific events that occur within the blockchain network. The technology behind the event listener is similar to ESB but the crucial difference is that this is a mechanism specifically for the blockchain network.

The importance of this addition came up when the architecture of the Resource Federation changed after the deliverable D4.6. This change was that, instead of the Resource Federation and the blockchain being outside the ODIN platform, the Resource federation was included in the ODIN platform. For the Resource Federation to be always up to date with the data that are in the ledger (more information about the ledger can be found in D4.6 “Implementation of Advanced CPS-IoT RSM Features v2”) of the blockchain the only solution is to add in the Resource Federation an event listener.

In the context of Resource Federation utilizing Hyperledger Fabric Network, blockchain events are generated as a result of transactions within the distributed ledger. To seamlessly capture and respond to these blockchain events, our implementation incorporates blockchain event listeners. These blockchain listeners are designed to actively monitor and interpret the emitted blockchain events, enabling Resource Federation to be always up to date. The blockchain listener mechanism operates on a publish-subscribe model, where the interested components subscribe to specific types of blockchain events. When a relevant blockchain event occurs, the blockchain listener captures and processes the information, facilitating the execution of predefined actions.

3.4 Use Cases

The following section presents specific use cases that demonstrate the practical applications of Resource Federation. These use cases offer insights into how seamless interactions and data management are facilitated in various scenarios.

3.4.1 Use Case for Federated Learning

In RUC A UC 3: Automated Sleep Scoring of Sleep Studies, federated learning was implemented to improve the automatic classification and diagnosis of sleep data. Anonymized sleep apnea data from a European database and insomnia PSG data from a previous clinical trial are provided for training purposes. The training process, which involves creating a model capable of automatically diagnosing sleep disorders, utilized federated learning techniques. Additionally, resource federation is utilized for transferring information between instances, further enhancing the training process.

The workflow for the [Figure 3](#) is, as follows:

1. KER1 initiates a request and publishes it to a designated topic on ESB1 within ODIN Instance 1.
2. Resource Federation 1(DLT API) monitors this access request, facilitating the submission of a corresponding transaction to the blockchain.
3. The blockchain triggers an event that is listened to by the Resource Federation in every ODIN instance.
4. In ODIN Instance 2, Resource Federation 2 listens for this event and verifies in its configuration whether KER2 is federated.
5. If KER2 is indeed federated, Resource Federation 2 proceeds to submit a transaction for granting access (including the URI of KER2 for access).
6. The blockchain's chaincode emits an event for the access grant message,
7. which Resource Federation 1 attentively listens to.
8. Subsequently, Resource Federation 1 publishes the access grant message to ESB1.
9. As a result, KER1 and KER2 now possess all the necessary information to establish a connection, enabling them to connect in the end.

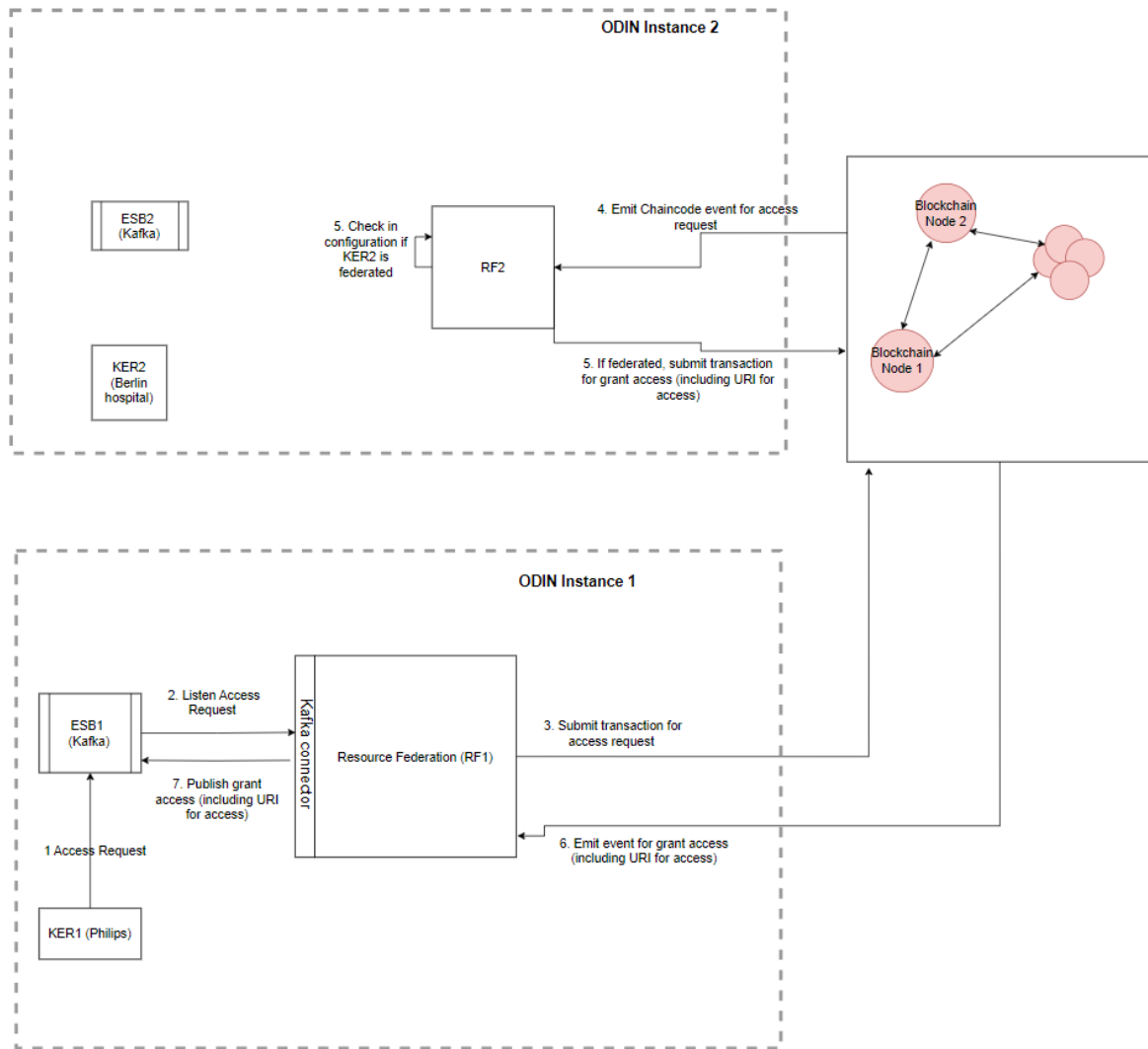


Figure 3 Resource Federation workflow

3.4.2 Use Case for Health Inventory Traceability framework based on blockchain

The open callers HIT - Health Inventory Traceability framework based on blockchain, requested access to the blockchain and the creation of a smart contract with the following functions.

HIT (Health Inventory Traceability framework based on blockchain) open-callers project, provides a complete framework for decentralized inventory management of hospital goods and medical devices. The solution proposed leverages the blockchain network provided by CERTH to trace and track assets in healthcare facilities along their entire life, i.e. from acquisition to decommissioning. Tracing involves determining the historical status of these goods based on the traceability data, and tracking refers to the resolution of the current condition. Traceability data includes data related to inventory management, ownership change and movement history recorded in detail throughout the asset life cycle. To provide accurate and real-time data, HIT leverages current KERs of ODIN platform, like HOSBOT for delivery of goods around the hospital or the mobile app created within the project to scan QR codes and enter the current status of a certain asset.

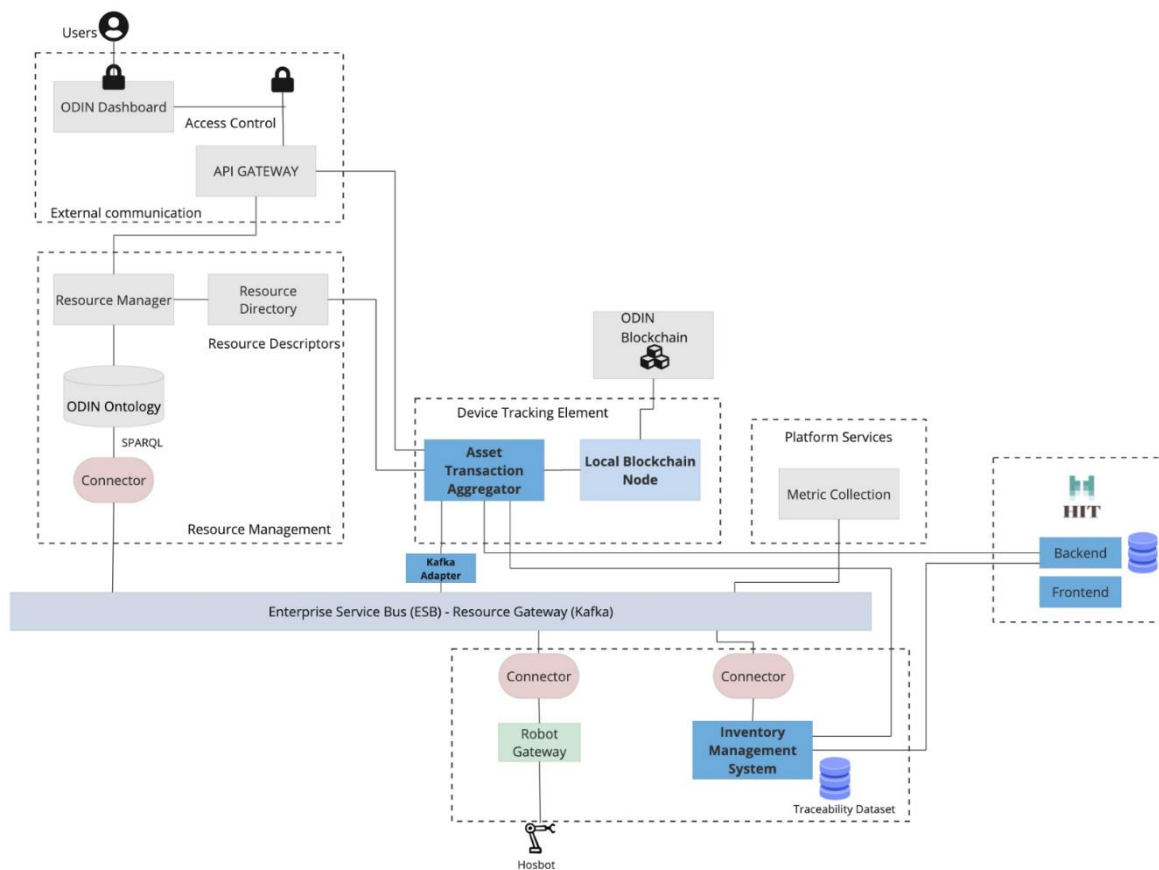


Figure 4 Integration of HIT with ODIN platform

Figure 4 illustrates the interaction of the HIT component with the blockchain. HIT has developed both the backend and frontend, along with the Inventory Management System and an Asset Transaction Aggregator, featuring its own Kafka adapter. In this use case, the primary emphasis lies on the Asset Transaction Aggregator and its integration with the blockchain.

The smart contract of the blockchain facilitates two primary functions: data posting and retrieval. In the data posting process, users initiate submissions by crafting requests that include device details, event types, locations, ownership information, and timestamps. These requests undergo secure processing within the system, which generates unique identifiers (UUIDs) upon successful submission, ensuring the secure storage of information on the blockchain and preserving the integrity of recorded data. Data retrieval offers access options, allowing users to retrieve all entries stored within the blockchain for a comprehensive overview of the data. Additionally, specific entry retrieval is supported through two methods: by UUID or by device ID. Users can access particular entries by utilizing the assigned UUID received during data posting or target specific data by submitting requests with associated device IDs.

4 Resource Choreographer

4.1 Evolution of the component

The Resource Choreographer was conceived as the service facilitating the orchestration of KER and services to achieve new business objectives. Originally, it was created using a traditional BPMN approach, which tends to be more static and less adapted to heterogeneous cloud computing scenarios. Also, while BPMN is mainly used to define business logic, Odin needed a component that was able to orchestrate the different services and scenarios at a technical level. For this reason, a serverless approach is better fitted to achieve this goal. The use of Kogito as a serverless workflow technology has allowed a smooth integration with Kubernetes used in Odin for container orchestration and is specialised to live in cloud-native environments. Through the project, this vision has been realized, resulting in the development of several microservices containing workflows designed to handle various tasks and support the ODIN platform in completing its use cases.

This shift represents a significant enhancement in scalability, as it eliminates reliance on a single component in favour of a swarm of services. Consequently, this approach minimizes the risk of single points of failure, contributing to a more reliable platform overall, which is a critical requirement in the hospital context and therefore for ODIN, too. With this evolution, the Resource Choreographer concept enables hospitals to introduce new technologies more seamlessly, as these services can be repurposed for broader scenarios. Integration becomes less problematic due to the automated task flows and the ability to reuse specific results across different contexts.

4.2 Main technologies & Implementation process

4.2.1 Kogito

Kogito, a cloud-native business automation technology, provides a flexible and scalable platform for developing intelligent business applications. Its features include rule engines, process automation, and decision management, empowering developers to efficiently build and deploy event-driven applications. It supports serverless architectures and offers tools for creating workflows, integrating with external services, and managing business rules and decisions.

Kogito is emphasized as a cloud-native business automation technology tailored for building cloud-ready business applications. While it offers various options for creating workflows, such as serverless workflows for service orchestration and BPMN definitions for business cases requiring task approvals, a thorough analysis during implementation revealed that all use cases, due to their middle to low complexity and requirements, could effectively utilize the serverless workflow model.

During the implementation phase, Kogito has proven to meet the project requirements effectively:

- enabling the creation of new services with minimal or no code. In cases where Kafka messaging is utilized, and simple orchestration without complex logic is needed, the workflow can be modeled by parameterizing the project and workflow states through the JSON descriptor (`applicationworkflow.wf`).
- seamlessly integrating with Apache Kafka to read and write messages from the bus. Topics for interaction can be defined in the "application.properties" file, while events can be specified in the "applicationworkflow.wf" file.
- facilitating interaction with external services through API calls. Serverless workflows provide integration with REST APIs, allowing workflows to call services using the OpenAPI

Specification. For services utilizing GraphQL, although direct integration is not yet available, an intermediate service can be created to interface with it using a GraphQL client.

- offering cloud integration capabilities. In the final implementation, a Docker approach was utilized, with deployment to Kubernetes for seamless integration with cloud environments.

4.2.2 How to design and implement a workflow

After the high-level workflow is defined using Lean process improvement methodology¹, Kogito facilitates the creation of a new service with serverless workflow. This entails a first analysis of the Use Case and the components involved.

To do so, component diagrams and data flow diagrams were used.

RUC A2 UC4: Robot Receptionist

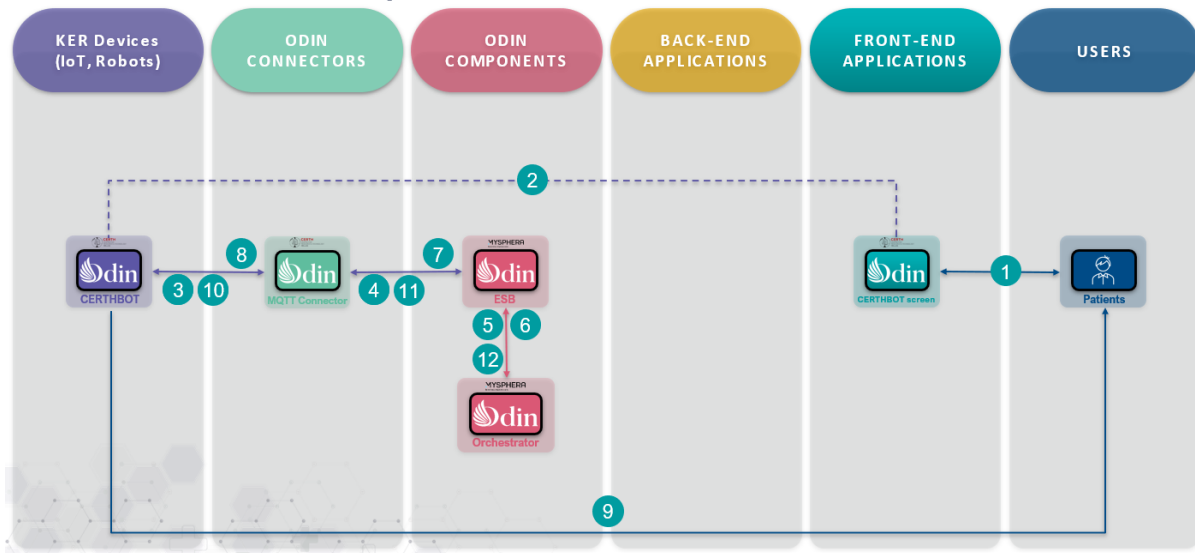


Figure 5 RUC A2 UC4 Component and data flow diagram

After this, the steps can be summarized as:

¹ <https://www.atlassian.com/agile/project-management/lean-process-improvement>

- Create an architecture of the workflow and the data using UML methodology. The architecture usually follows one or a mix of several cases. The architecture section explains the typical cases found.
- With this architecture, the events, states and the topics to be used can be declared and named.
- The next action is to the concrete the message content for each event.
- Finally, the creation of the workflow can be done with the following steps.
 - Create the Kogito project. The Kogito documentation² describes how to create a project, but it can take some advantage of currently implemented ODIN workflows in the Gitlab, or use the examples in Kogito Github³
 - Define the channels to be used in the “application.properties” file. Each channel is associated with one or more topics.
 - Define the states, events, actions, API calls and Services in the “applicantworkflow.sw” file.

Create a Service (usually a resource) to manage intermediate actions referenced in the “applicantworkflow.sw” file.

4.2.3 Architecture of the workflow

All ODIN cases could be solved using two serverless workflow architectures as presented in the following paragraphs.

² <https://kogito.kie.org/get-started/>

³ <https://github.com/kiegroup/kogito-examples/tree/main-apache/serverless-workflow-examples>

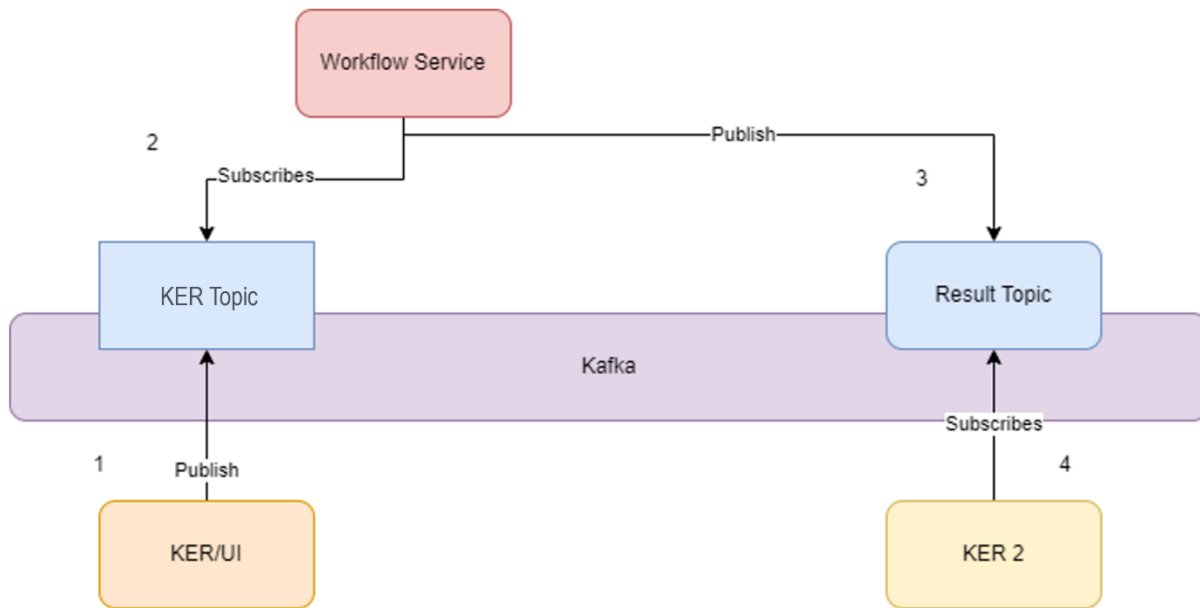


Figure 6 Basic architecture

This first architecture depicted in [Figure 6](#) is the most basic:

1. a KER or UI sends messages to a topic “KER topic” when an interesting event happens.
2. The Workflow Service (WS) is connected to Kafka to receive those messages. The WS then processes the message and starts following the internal states defined.
3. In this case, the Workflow service processes the message and puts the result in the “Result topic”.
4. This result can be read by another KER.

Usually, this schema can be used to monitor for example, a sensor and send an alarm when the logic decides to do so, or a UI starting a process.

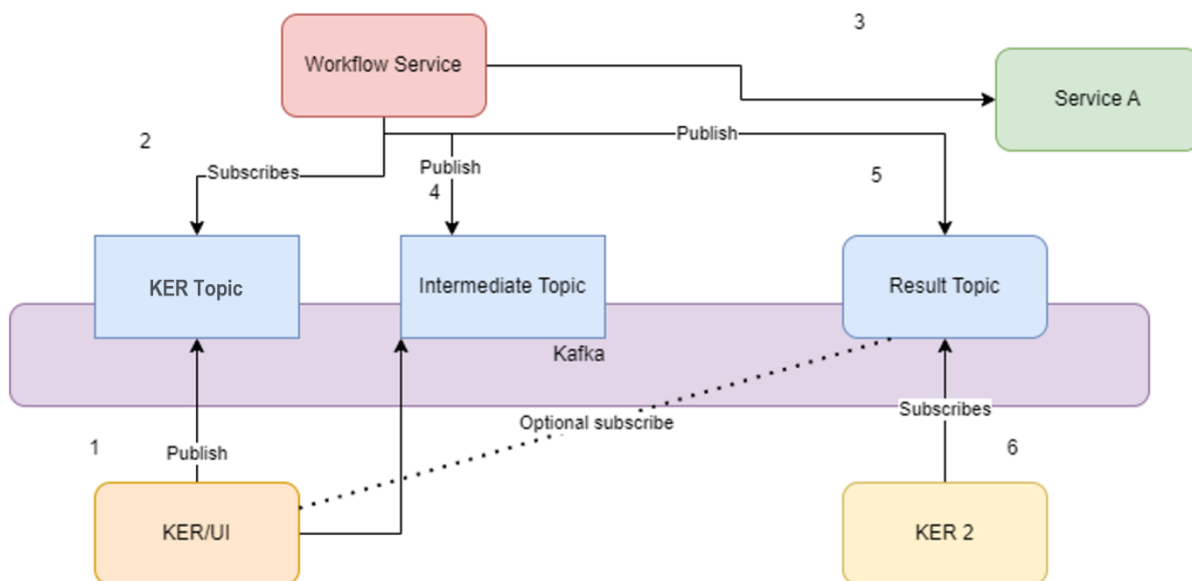


Figure 7 example complex workflow

A more complex process builds upon the previous one depicted in [Figure 7](#) and, is described as follows:

1. A "KER topic" serves as a feed for messages generated by a KER or a User Interface whenever an event of interest occurs.
2. The WS is connected to Kafka to receive these messages. Subsequently, the WS processes the message and initiates the internal states as defined within the workflow.
3. As part of the specific workflow process for each use case, the WS might need the interaction with another Service (Service A), which could potentially be a KER exposing an API. This interaction occurs through an API call.
4. The outcome of the interaction is then relayed to the KER using additional specified topics termed "Intermediate topics."
5. In most scenarios, the microservice forwards a result message to a specific topic labeled "Result Topic."
6. Another KER may subscribe to this topic to access and utilize the received result.

Usually, this architecture can happen interacting with more complex KER's such as Robots, or AI that starts with a UI or event trigger, follows KER interaction, other service consumption and finally reports the result. The Documentation component and the API Gateway follows a schema similar to this.

Other schemas may exist, but these two comprehend all the use cases reviewed for ODIN.

4.3 Use Cases

Previous version of this deliverable (D4.6 "Implementation of Advanced CPS-IoT RSM Features v2") presented several use cases where the Resource Choreographer could be used.

[Table 3](#) covers the 12 workflows proposed for the pilots use cases and other components and the reason to implement them or not (more information on pilot use cases can be found in

Deliverable 7.1 “Pilot Studies Use Case Definition and Key Performance”). In total 3 of them were implanted but only one included, which is highlighted in blue in table below.

Table 3 Use cases analysis

Use Case	Description	Proposed	Included
SER RUC B1 UC1	Stent Administration	Yes	No. The RC was not adding value, as it was just a message dispatcher, and it was not its duty.
SER RUC B2 UC2	Stent Transportation	No	No. Robots have implemented their own human interface, and every message is managed among them.
SER RUC C UC7	Disaster Preparedness	No	This use case did not require RC.
CUB RUC A1 UC3	Sleep Scoring	Yes	No. The RC was not adding value, as it was just a message dispatcher, and it was not its duty.
CUB RUC A2 UC4	Robot Receptionist	Yes	Yes
CUB RUC C UC7	Patient Monitoring/ Evacuation/ Intruder	No	This use case did not require RC.
UCBM RUC A2.1 UC4	Food consumption monitoring	Yes	No. The RC was used as messaging broker.
UCBM RUC A2.2 UC4	Rehabilitation monitoring	Yes	First version implemented, but final use case was not using the RC for orchestration purposes, so it was removed.
UCBM RUC A3 UC4	Oxygen monitoring	Yes	No. The RC was used as messaging broker.
UCBM RUC B1 UC1	Food Delivery Process	No	Use case was planned to use choreography without RC.
MUL RUC A2 UC4	Blood sample transportation	No	No. Robots have implemented their own human interface,

			and every message is managed among them.
MUL RUC B2 UC2	Medical Equipment Location	No	The RTLS provides the location through a map and no RC is needed
MUL RUC C UC7	Biohazard Transportation	No	No. Robots have implemented their own human interface, and every message is managed among them.
UMCU RUC A1 UC3	CVD Diagnosis Workflow	No	Not integrated with RC as it was not required for the use case.
UMCU RUC A2 UC4	CVD Patient Identification	No	Not integrated with RC as it was not required for the use case.
UMCU RUC C UC7	Disaster Preparedness	No	Not integrated with RC as it was not required for the use case.
UMCU RUC B1 UC3	Robotic Transportation: mock-up environment	No	Not integrated with RC as it was not required for the use case.
HUVR RUC B2 UC4	Perioperative Pathway	No	Not integrated with RC as it was not required for the use case.
HUVN RUC B2 UC4	TAVI pathway	No	Not integrated with RC as it was not required for the use case.

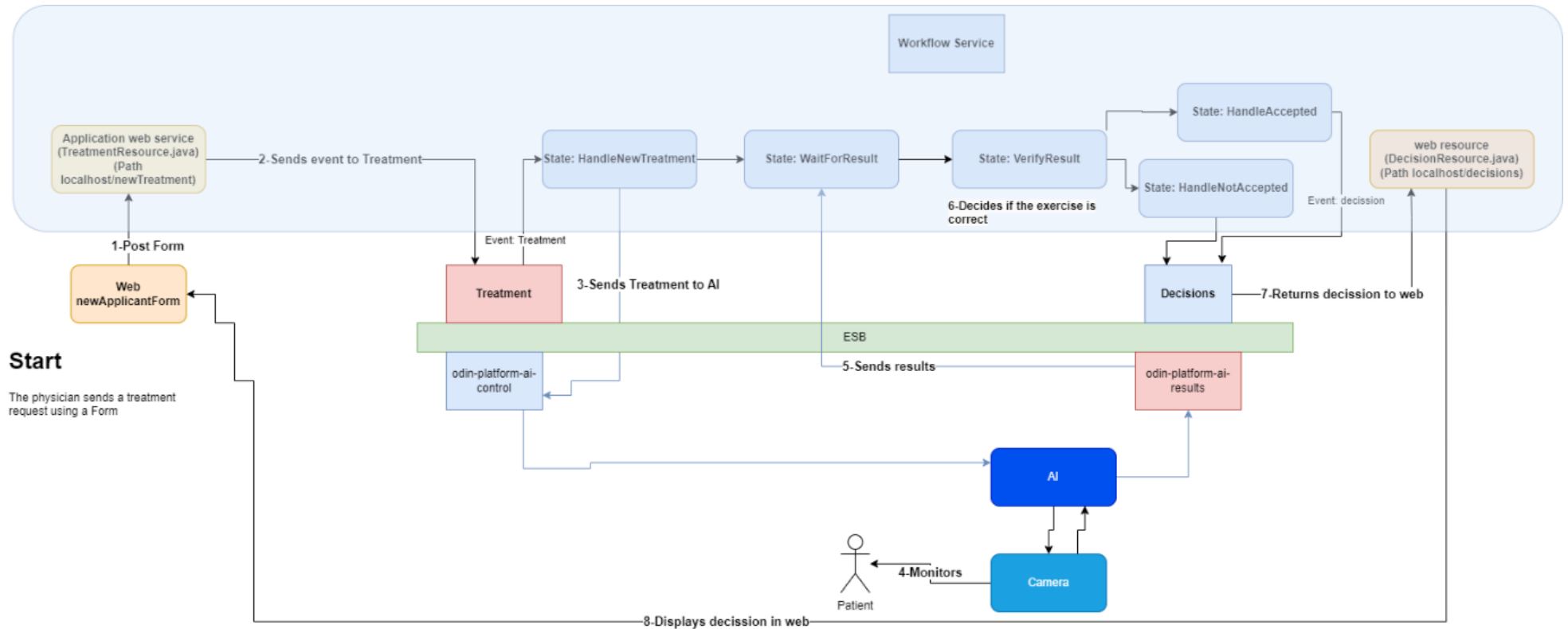
In conclusion, the implementation phase was primarily focused on developing specific versions of workflows for the different use cases, namely the Rehabilitation AI for UCBM and the Robot Receptionist for CUB.

The Rehabilitation AI use case focused on conducting rehabilitation sessions with patients whose exercises were monitored via camera. The camera captured images of the exercises, which were then analysed by an AI algorithm to determine if each movement was performed correctly.

The Resource Choreographer was implemented to manage the initiation of exercise recording. This allowed doctors to use a web page to set up the exercise parameters, including the patient's details and exercise parameters. When triggered by the web page, the workflow processed the request and sent a start signal to the AI algorithm. Subsequently, the camera began recording the patient's movements, sending images to the AI for analysis. Once the patient completed the required repetitions, the AI ceased analysis and transmitted the results to the Resource

Choreographer. The Resource Choreographer then determined whether the exercise was performed correctly based on predefined rules within its states. Finally, the web page was updated with the decision, enabling the doctor to either request another exercise or conclude the session.

Figure 8 covers all the architecture for this case, with the events, states, topics, and data flow.



Start

The physician sends a treatment request using a Form

Explanation

- 1-The physician sets up a treatment for a patient using the web form and submits it
- 2-The requests is converted to a message so the workflow can read it from Kafka
- 3-The workflow processes the requests and sends it to the AI
- 4-The AI starts recording the exercise, monitoring the patient with the camera
- 5-When it finishes the number of repetitions, sends the results to the workflow
- 6-The workflow computes the result of the treatment using a rule. This gives power so we can change the rule without modifying the AI.
- 7-The result is sent back to the web
- 8-The web displays the result

Figure 8 Rehabilitation AI workflow architecture

4.3.1 Robot Receptionist

The Robot Receptionist use case, corresponding to the CUB (Charité) pilot use case, involves a patient who receives a QR code when scheduled for a medical visit. Upon arrival, the patient presents the QR code to the robot's camera, which initiates a message transmission to the workflow. As the robot lacks knowledge of the QR code's meaning, the workflow assumes responsibility for QR code tracking and room coordinates management. Upon receipt of the QR code, the workflow correlates its content with coordinates and dispatches these coordinates to the robot. Subsequently, the robot computes a route to the designated room and guides the patient accordingly. Upon completion, the robot relays the outcome to the workflow, which in turn disseminates the result to another topic for consumption by other KER.

For brevity, the workflow will not be included in this deliverable. However, upon completion, it can be accessed at: <https://odin-gitlab.iti.gr/odin/resource-choreographer-robot-guide>.

To test the use case without AI, comprehensive instructions are provided in the "Readme.md" file, detailing how to run the workflow and conduct testing using specific messages and steps.

Throughout the development process, this use case underwent testing in three environments. Initial tests were conducted locally by MYSOPHERA, followed by integration testing in an environment provided by CERTH with an ODIN instance. The final round of testing took place at CUB hospital prior to the scheduled pilot dates.

4.4 Documentation Component workflow

The Documentation Component leverages the capabilities of Kogito, particularly its connectivity with Kafka and API integration features. While Kogito currently lacks native support for GraphQL in its configuration files, a GraphQL client module has been developed and seamlessly integrated into the Documentation Component. This enables the Documentation Component to interact with the wiki system⁴ by making API calls using the provided API-Key, facilitating the insertion or updating of new content as needed.

The Documentation Component source code is hosted at <https://odin-gitlab.iti.gr/wp4/documentation-ker>. While the implementation has commenced and the framework is defined, it is important to note that none of the KERs are currently utilizing this component. Following a thorough assessment, the priority of this component has been demoted, as its perceived value to the platform falls below that of other components.

4.5 API Gateway workflow

The API Gateway workflow follows a similar approach to the Documentation component. It listens for changes from KERs seeking to publish an API outside the cluster. The components integrate via REST API with the Proxy Server responsible for exposing the API. Additional details can be found in Deliverable 4.4 "Implementation of Local CPS-IoT RSM Features v3".

⁴ <https://odin-wikis.iti.gr/>

4.6 Other features

Kogito offers a framework for implementing workflows, although currently lacks an API or middleware for dynamic creation or updating of workflows. However, it provides numerous tools for managing workflow status, monitoring, and designing new workflows. While these features are reserved for future development, comprehensive documentation is available to explore the capabilities of each functionality.

The [Figure 9](#) below illustrates the tool offered by Kogito to design workflows in a graphical way:

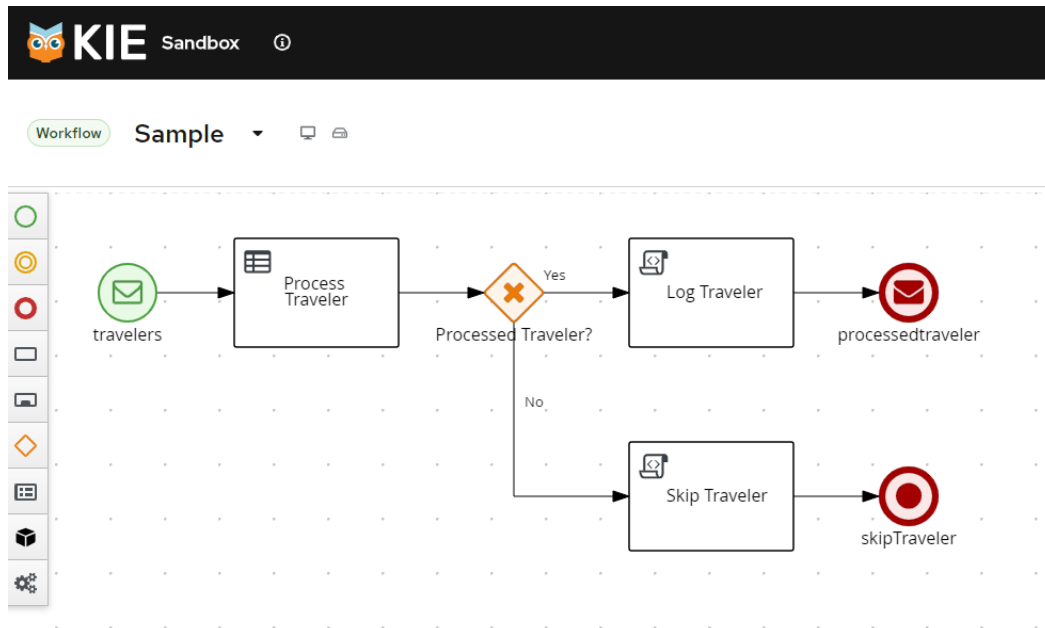


Figure 9 Kogito Workflow designer tool

5 Conclusion

This document has explored the intricate landscape of two essential elements, Digital Ledger Technologies (DLT) and the Resource Choreographer, within the ODIN platform. Through examination, it is evident that these components serve as the foundation of ODIN's operational effectiveness, efficiency, and innovation in healthcare service provision.

The integration of cutting-edge Digital Ledger Technologies, particularly the incorporation of Fabric Network into Resource Federation, underscores ODIN's commitment to reinforce its infrastructure. This integration serves as a shield, enhancing platform resilience and facilitating seamless integration with the broader network ecosystem, thereby promoting flexibility and responsiveness. Additionally, the emphasis on resource federation underscores the importance of enabling components from different instances to share critical information, enhancing collaboration and efficiency.

There's potential for further development, including the ability to query Fuseki (details about fuseki Fuseki can be found in D4.3 "Implementation of Local CPS-IoT RSM Features v2" and D4.4 "Implementation of Local CPS-IoT RSM Features v3") in the future. This future advancement could enable, querying a central repository where KERs register with the resource manager, housing information about federated components. Another idea for future development could involve making Resource Federation accessible through the administration dashboard (more information about the administration dashboard can be found in D3.12 "Odin Platform v3" in the section 5.6.2). An administrator would then be able to access the resource manager from the administration dashboard to register KERs as federated. Additionally, they could configure a specific time period for federation. This approach would enable the Resource Federation to disseminate this information to all ODIN instances, allowing administrators in other instances to view this information on their dashboards.

Moreover, the evolution and implementation of the Resource Choreographer highlights its critical role as a service orchestrator, enabling dynamic interactions between KER and services. The transformation of the Resource Choreographer into a collection of microservices underscores ODIN's commitment to scalability, resilience, and ongoing technological progress. These are key non-functional requirements for ODIN and any platform that aims to serve resilient services and application to health professionals in such a sensitive environment.

In conclusion, the integration of cutting-edge technologies and dynamic service orchestration provided by Digital Ledger Technologies and the Resource Choreographer positions ODIN as a pioneer in distributed ledger systems and healthcare service delivery.