This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nº 101017331

D4.6 Implementation of Advanced CPS-IoT RSM Features v2

Deliverable No.	D4.6	Due Date	28/02/2023
Description	D4.6 describes 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
Ilias Kalamaras	CERTH	kalamar@iti.gr
Vasileios Lolis	CERTH	vaslwlis@iti.gr
Konstantinos Flevarakis	CERTH	kostisfl@iti.gr
Konstantinos Votis	CERTH	kvotis@iti.gr
Pilar Sala	MYS	psala@mysphera.com
Pablo Lombillo	MYS	plombillo@mysphera.com

History

Date	Version	Change
14/11/2022	0.1	Initial draft containing table of contents.
9/01/2023	0.2	First draft
31/01/2023	0.3	Content addition
20/02/2023	0.4	Content addition
28/02/2023	0.5	Peer Review Version
23/03/2023	0.6	Quality Check Version
08/05/2023	1.0	Final Version

Key data

Keywords	Digital Ledger Technologies, blockchain, resource federation, resource choreography, resource workflow design
Lead Editor	Anastasia Blitsi (CERTH)
Internal Reviewer(s)	UoW, UMCU

Abstract

Deliverable D4.6 outlines the advanced features of the ODIN platform, which include two key components: Digital Ledger Technologies and the Resource Choreographer. Together with D4.2, these components form the core of ODIN's resource management and interaction capabilities. The Digital Ledger Technologies component is designed to facilitate trustworthy resource federation by leveraging blockchain technology to securely share resources between different ODIN instances. The Resource Choreographer, on the other hand, allows users to create operational pipelines and workflows that can coordinate the actions of various resources to achieve specific goals.

This deliverable provides an in-depth description of the architecture and key technologies used in both components, as well as some use cases where they can be applied.

Overall, this document serves as a comprehensive guide to understanding the Digital Ledger Technologies and Resource Choreographer components of the ODIN platform, providing insight into their operation and implementation, as well as offering practical applications for both components.

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

List of tables

List of figures

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
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
UI	User Interface
XML	eXtensible Markup Language

1 Introduction

This deliverable describes advanced features of the ODIN platform, specifically the Digital Ledger Technologies, used for trustworthy resource federation, and the Resource Choreographer, used for resource composition into high-level operational workflows. Together with D4.3 "Implementation of Local CPS-IoT RSM Features v2", it completes the description of the core ODIN platform components responsible for resource management and interaction. While D4.3 focused on low-level resource management, such as semantic resource abstraction, resource communication and metric collection, D4.6 focuses on advanced features that enable the expansion of the ODIN platform into larger distributed ecosystems and custom high-level applications and domains.

The deliverable describes two components of the ODIN platform:

- The Digital Ledger Technologies (DLT) component. This component enables resource federation across ODIN instances in a trustworthy manner. Resource federation is a key idea of the ODIN platform, allowing its implementation to be distributed, with resources of one instance (e.g. a hospital, a cloud machine, etc.) being shared by other remote instances. Resources are valuable assets; therefore, ODIN instances need to ensure that they are shared in a trustworthy manner. DLT technologies, powered by a blockchain infrastructure, allow all resource sharing requests and permissions to be handled and all transactions between ODIN instances to be securely audited, ensuring immutability and non-repudiation. Section 2 is devoted to the ODIN DLT components.
- The Resource Choreographer component. This component allows the composition of resources into operational workflows that implement high-level logic. Based on the semantic abstractions of each resource (i.e. the resource descriptors, described in D4.3, constructed according to the corresponding entities of the ODIN ontology, described in D3.3), each resource will possess, among others, a number of input and output data channels and types. The Resource Choreographer allows the user to design workflows by connecting the outputs of one resource to the inputs of another. These workflows are meant to represent high-level business logic for implementing hospital operations, such as the ones described in the ODIN use cases, by specifying how resources should communicate with each other. The Resource Choreographer component is described in Section 3.

For each of the above components, the current deliverable presents the relevant information in the following structure:

- Requirements review, i.e. which ODIN requirements are fulfilled by each component;
- Architecture review, i.e. how each component fits in the ODIN platform architecture;
- Applicable technologies for the implementation of each component;
- A comparison of the applicable technologies to select the most appropriate ones;
- A list of features to implement in the first version of each component.

At the current stage of the project (end of first year), the development of both components is in the design phase, with the specification of the requirements, architecture and overall functionality of each component. Implementation will start from the beginning of the second year.

1.1 Deliverable context

Table 1 provides an overview of the context of the current deliverable, in relation to the project objectives and foreseen results.

Table 1: Deliverable context.

PROJECT ITEM	RELATIONSHIP		
Objectives	The deliverable is relevant to ODIN's Objective 1, as it describes components that enable the secure and trustworthy decentralized and federated implementation of the ODIN platform, as well as components that facilitate the usage of the Key Enabling Resources in hospital applications, bridging the gap between KER suppliers and healthcare organizations.		
Exploitable results	• The ODIN Digital Ledger Technologies infrastructure for federated KERs • The Resource Choreographer component for resource composition and orchestration		
Workplan	D4.6 is attributed to the tasks of WP4 "CPS-IoT Resource Management System". Specifically, the tasks involved in the preparation of this deliverable are T4.4 "Digital Ledger Technologies Resource Federation and management framework" and T4.5 "Resource Choreographer module".		
Milestones	D4.6 is a key deliverable of the PROCUREMENT PROCEDURE SIMULATION (MS2) and IMPLEMENTATION (MS3) phases of the project.		
Deliverables	D2.3	ODIN Platform catalogue	Regarding the available resources that can be potentially shared and orchestrated
	D3.3	Hospital Knowledge Base and ODIN semantic ontology	Regarding the semantic resource abstractions needed for the Resource Choreographer
	D3.4 – D3.6	Privacy Security and Trust report	Regarding security and trust concepts related to resource federation
	D3.7 – D3.9	Technical Support Plan and Operations	Regarding component documentation
	D3.10 – D3.12	ODIN platform	Regarding the integration of the components in the ODIN platform
	D4.1	CPS-IoT Resource Management System Specification	Regarding the requirements for the DLT and RC components
	D4.2 – D4.4	Implementation of Local CPS-IoT RSM	Regarding the resource descriptors and resource

	Features	communication channels needed by the RC and DLT components
D7.9	Pilot Studies Evaluation Results and sustainability	Regarding component evaluation results of unit/integration testing.

Risks

The following risks are relevant to this deliverable:

- Technical problems during component/module development
- Risk of time-consuming implementation and integration due to multiple technologies involved.

Both the DLT and Resource Choreographer components presented in this deliverable deal with communication between resources from a high-level perspective, either for sharing or for workflow composition. For this reason, the risk of delays due to the interconnection of diverse technologies is always present. The semantic resource abstraction and common communication through the resource gateway (see D4.3) offer mechanisms to mitigate this risk.

On the other hand, this deliverable offers components that help mitigating the following risks:

- Loss of compliance due to a low user acceptance of the system (by providing means that facilitate the creation of custom high-level applications and providing a trustworthy substrate for resource sharing)
- Legal restrictions imposed in the execution of the trials (by providing secure and trustworthy means for resource sharing)
- Failure to attract proposals for open call (by providing trustworthy means for sharing resources from open calls with ODIN instances, and connecting open call resources with native ODIN resources)

1.2 Overview

An overview of the ODIN platform architecture, with the two components described in this deliverable, the Digital Ledger Technologies and the Resource Choreography, highlighted, is presented in Figure 1. At the bottom layer lie the Key Enabling Resources, i.e. the robots, IoT devices, AI services, databases, front-end and back-end services, etc., as well as the existing Hospital Information System. These are the elements that empower the design of high-level applications. Resources communicate with each other and with other platform components in two communication layers: either through the Enterprise Service Bus (ESB), after being semantically and syntactically transformed to a common data model and a common bus protocol, or through direct communication with each other, e.g. in intra-robot communication, or for large data transfer.

The DLT and Resource Choreographer components lie above this layer, at the platform service level, together with core components such as the ODIN ontology, resource directory, metric collection and platform management. Both components are connected to the Enterprise Service Bus, for communicating with resources. Communication with the outer world always passes through the ODIN Application Programming Interface (API) gateway, which in turn passes through proper user/service authentication. Details about specific architectural elements of each component are provided below, in sections 2.2 and 3.2.

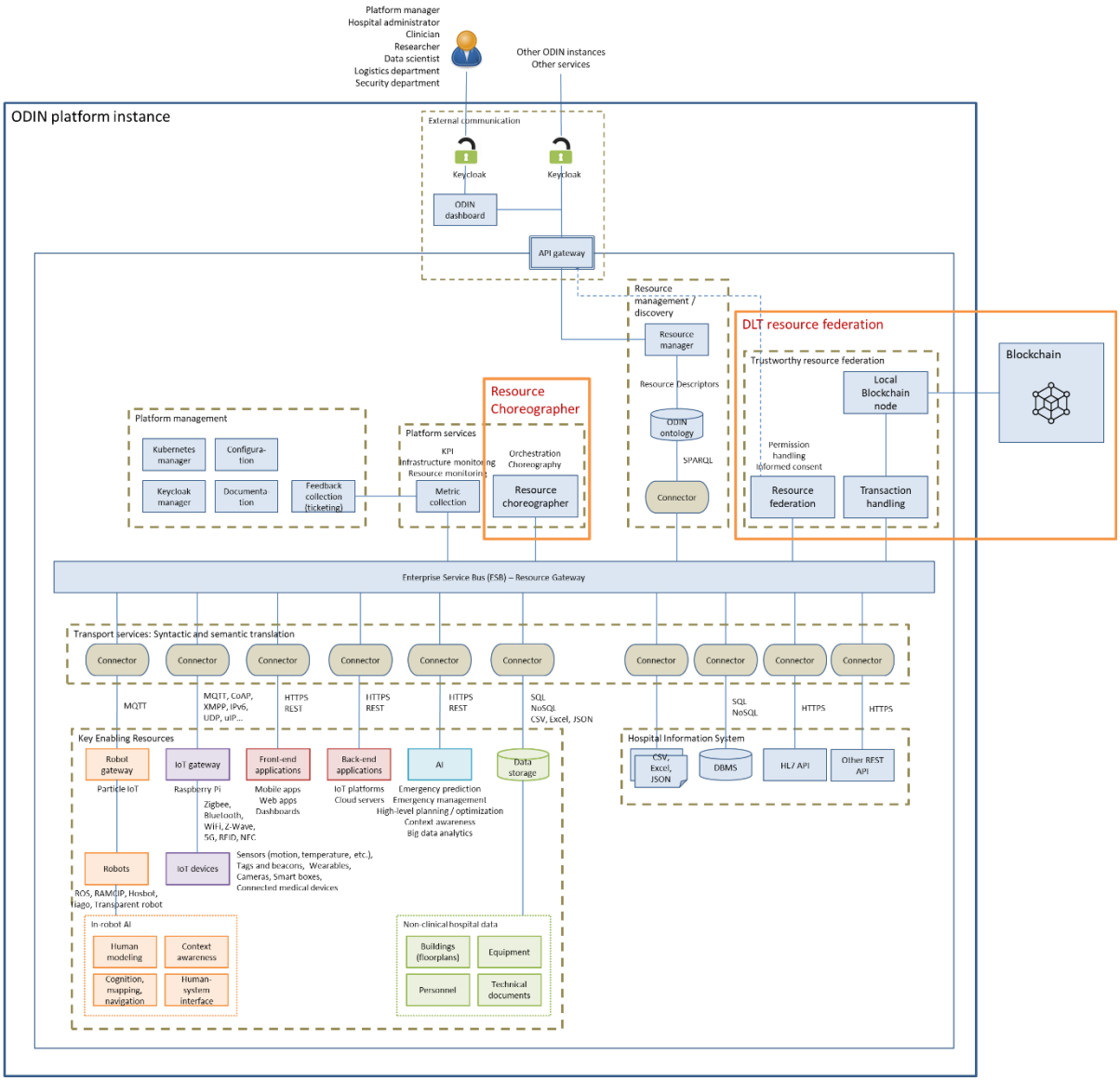

Figure 1: Positioning of the Digital Ledger Technologies and Resource Choreographer components in the ODIN platform architecture.

2 Digital Ledger Technologies

Distributed Ledger Technologies (DLTs) are systems for securely recording and sharing data and transactions in a decentralized manner. They use decentralized networks, cryptographic algorithms, and consensus mechanisms to create a secure, transparent, and tamper-resistant digital ledger that can be shared among multiple participants. Examples of DLTs include blockchain-based systems such as Bitcoin and Ethereum, as well as Hyperledger Fabric, which is going to be used in the ODIN platform. DLTs have the potential to transform a wide range of industries by providing a secure and efficient way of tracking and exchanging digital assets and information, while reducing the need for intermediaries and increasing trust and transparency.

2.1 Introduction

This chapter provides an overview of the key concepts and technologies involved in building ODIN platform using Distributed Ledger Technologies (DLTs). It includes information about Hyperledger Fabric, the CCKit, and a description of the DLT components that will be used to create a secure and efficient system for managing patient data and medical records. The chapter serves as an introduction to the technical foundations of ODIN platform and provides a deeper understanding of the underlying technologies that enable the platform to achieve its goals of improved security, transparency, and efficiency in healthcare.

2.1.1 Privacy and Trust

Several strategies are offered to provide security, privacy, and trust management in health care networks, as mentioned in D3.5 Privacy Security and trust v2. Recently, blockchain technology has been advocated as an important component of several methods to establish a safe, private, and trustworthy environment. Blockchain is an immutable ledger that is often executed without a central repository and without a central authority. It is a new trend in the field of safeguarding shared information across many organizations in a network.

The primary benefits of blockchain are decentralization, trust, transparency, anonymity, and non-repudiation. The ledger is maintained by various network nodes, resulting in a distributed network architecture. Through the consensus process, trust can be established without the involvement of a trusted third party. Once uploaded to the blockchain, its segments, or blocks, are difficult to edit, resulting in non-repudiation.

Blockchain validates and stores data using cryptographically linked block structures, produces and updates data with consensus methods, and programs and alters data using smart contracts. The whole network is working to achieve an agreement on the most recent block to be put to the blockchain.

2.1.2 Data pre-screening

Privacy, trust, and decentralisation will be addressed via the implementation of DLT-related components in ODIN. DLT is well-known for (i) decentralizing IoT network operations, (ii) providing a stable state for IoT devices, (iii) being resistant to change, (iv) providing anonymity, and (v) creating data immutability.

A permissioned blockchain will run among a set of known, identified, and verifiable participants and will be managed by a system that generates a certain amount of trust. The permissioned network offers flexibility since the business case can determine the number of participants, driving the adoption of a private blockchain with one organization or a consortium blockchain with several organizations. Each participant may have varying levels of access, so only authorized people may

publish. Furthermore, in such a permissioned environment, the likelihood of a participant purposely inserting dangerous code via a smart contract is lowered. The elimination of potentially damaging occurrences is achieved, since all DLT transactions are recorded on the blockchain in line with an endorsement policy specified for the network and relevant transaction type. Rather than being completely anonymous, the perpetrator may be easily identified, and the matter treated in accordance with the standards of the governance model. Additionally, data will be pushed to the blockchain via transactions, ensuring that all acts are recorded on the ledger. Finally, data must follow a precise format established by the smart contracts or they will be rejected.

Data pre-screening is a crucial step in the process of logging data on a distributed ledger technology (DLT) system. It involves thoroughly checking and verifying the data to ensure that it adheres to the predetermined format and meets the specific requirements set by the smart contract. This process helps to ensure the integrity and accuracy of the data that is recorded on the DLT, as well as to prevent any errors or inconsistencies from being introduced into the system. By conducting thorough data pre-screening, instances can maintain the trust and transparency of the DLT, and ensure that all recorded data is secure, reliable, and accessible for future use. Thereby, data must conform to a predetermined format in order to be recorded on the distributed ledger technology (DLT). This format is established beforehand so that the smart contract is configured to receive and process only that specific format.

2.2 Main Technologies

2.2.1 Hyperledger Fabric

Hyperledger Fabric (HLF) is an open source permissioned blockchain developed by the Linux Foundation that is distinguished from other systems by its innovative transaction architecture, known as the Execute-Order-Validate architecture. This new design replaces the old order-execute model utilized by all previous platforms. In order-execute architecture, transactions are first arranged depending on the consensus protocol. Then, in the execution phase, each peer processes transactions consecutively and in the same sequence. Furthermore, another feature that distinguishes HLF is its support for general-purpose programming language smart contracts, or chaincodes as HLF refers to them. Smart contracts in platforms that follow the order-execute architecture must be deterministic, which is why some networks need smart contracts to be written in a Domain-Specific Language (DSL) to reduce non-deterministic processes.

In addition, Hyperledger Fabric is a permissioned blockchain framework that provides numerous advantages concerning the environment. One of the most significant benefits is its energy efficiency compared to other blockchain platforms that use PoW consensus mechanisms. Hyperledger Fabric's consensus mechanism is less energy-intensive, reducing the carbon footprint associated with blockchain operations. Additionally, Fabric's scalability enables it to handle a large number of transactions simultaneously, reducing energy consumption and carbon footprint. Its modular architecture further enhances its flexibility and customization options, allowing users to optimize their systems for energy efficiency and further reduce their carbon footprint. Overall, these advantages make Hyperledger Fabric a sustainable and eco-friendly choice for blockchain applications.

Hyperledger Fabric Chaincode Kit (CCKit) is an open-source toolkit for developing and deploying chaincode (smart contracts) on the Hyperledger Fabric platform. It provides developers with a set of abstractions and libraries that simplify the process of writing and deploying chaincode.

CCKit helps to reduce the amount of boilerplate code needed for chaincode development, making it easier for developers to focus on the business logic. Additionally, it offers a suite of testing tools to ensure the reliability and security of chaincode, as well as support for multiple programming languages such as JavaScript, Go, and TypeScript. CCKit streamlines the chaincode development process and helps to enhance the overall performance and security of Hyperledger Fabric-based applications.

An example of chaincode about cars, in Golang, is as follows:

```go
// Package cars simple CRUD chaincode for store information about cars
package cars

import (
	"time"

	"github.com/s7techlab/cckit/extensions/owner"
	"github.com/s7techlab/cckit/router"
	p "github.com/s7techlab/cckit/router/param"
)

const CarEntity = `CAR`
const CarRegisteredEvent = `CAR_REGISTERED`

// CarPayload chaincode method argument
type CarPayload struct {
	Id    string
	Title string
	Owner string
}

// Car struct for chaincode state
type Car struct {
	Id    string
	Title string
	Owner string

	UpdatedAt time.Time // set by chaincode method
}

// Key for car entry in chaincode state
func (c Car) Key() ([]string, error) {
	return []string{CarEntity, c.Id}, nil
}

func New() *router.Chaincode {
	r := router.New(`cars`) // also initialized logger with "cars" prefix

```

Figure 2:This is a Go source code example, that defines a Chaincode for a car-related application. The code is written using the "cckit" library which provides a set of tools to simplify building Chaincode applications in Go. (1/2)

The Chaincode is defined as a Router, which allows the application to handle multiple methods (endpoints) for reading and writing data on the blockchain network.
The Chaincode has four main components:

- Constants for the entity name (CarEntity) and event name (CarRegisteredEvent)
- A struct type (CarPayload) that represents the argument for the Register method
- A struct type (Car) that represents the data for the cars stored in the chaincode state
- A function named "New" that creates and returns a new Router (Chaincode) object with defined methods:
  - Init: initializes the chaincode's state to contain the identity of its creator as the owner
  - Query Car: retrieves information about a car by its ID

- Query Cars: retrieves a list of all cars
- Invoke Car Register: registers a new car, inserting its information into the chaincode state, and triggers two events (CarRegisteredEvent and CarRegisteredEvent)

```
// ======= Init ==================
func invokeInit(c router.Context) (interface{}, error) {
        return owner.SetFromCreator(c)
}

// ======= Chaincode methods =====

// car get info chaincode method handler
func queryCar(c router.Context) (interface{}, error) {
        // get state entry by composite key using CarKeyPrefix and car.Id
        //  and unmarshal from []byte to Car struct
        return c.State().Get(&Car{Id: c.ParamString(`id`)})
}

// cars car list chaincode method handler
func queryCars(c router.Context) (interface{}, error) {
        return c.State().List(
                CarEntity, // get list of state entries of type CarKeyPrefix
                &Car{})    // unmarshal from []byte and append to []Car slice
}

// carRegister car register chaincode method handler
func invokeCarRegister(c router.Context) (interface{}, error) {
        // arg name defined in router method definition
        param := c.Param(`car`).(CarPayload)

        t, _ := c.Time() // tx time
        car := &Car{     // data for chaincode state
                Id:        param.Id,
                Title:     param.Title,
                Owner:     param.Owner,
                UpdatedAt: t,
        }

        // trigger multiple event
        if err := c.Event().Set(CarRegisteredEvent+`First`, car); err != nil {
                return nil, err
        }

        if err := c.Event().Set(CarRegisteredEvent, car); err != nil {
                return nil, err
        }

        return car, // peer.Response payload will be json serialized car data
                //put json serialized data to state
                // create composite key using CarKeyPrefix and car.Id
                c.State().Insert(car)
}
```

Figure 3: This is a Go source code example,that defines a Chaincode for a car-related application. The code is written using the "cckit" library which provides a set of tools to simplify building Chaincode applications in Go. (2/2)

2.3 DLT components

A Distributed Ledger Technology (DLT) component is a building block or a software module that contributes to the overall functionality and structure of a DLT system. DLT components can include consensus algorithms, data storage systems, cryptographic algorithms, network communication protocols, and smart contract execution engines. These components work together to enable secure and transparent record-keeping, processing, and transfer of data and assets in a decentralized manner.

From the platform/consumer perspective, DLT components in the ODIN platform provide the infrastructure necessary for secure, transparent, and efficient record-keeping, processing, and transfer of data and assets in a decentralized manner. These components enable the creation of decentralized networks that can provide greater security, efficiency, and scalability compared to traditional centralized systems. In the context of ODIN, DLT components are used to enable secure and efficient collaboration among hospitals and healthcare organizations.

The main problem that DLT components aim to solve in the ODIN platform is the lack of interoperability and trust between different healthcare systems and organizations. This can result in inefficiencies, delays, errors, and reduced patient outcomes. By using DLT components such as consensus algorithms, data storage systems, cryptographic algorithms, network communication protocols, and smart contract execution engines, ODIN can create a decentralized network that enables secure and transparent record-keeping, processing, and transfer of data and assets.

These components work together to enable resource federation, which allows for the sharing of computing resources across different hospitals and organizations. This can help to reduce costs, increase efficiency, and improve patient outcomes. Additionally, DLT components can enable privacy-preserving smart contracts that allow hospitals to share data while keeping sensitive information confidential. This can help to protect patient privacy and comply with data protection regulations such as HIPAA.

Overall, the use of DLT components in the ODIN platform aims to solve the problem of interoperability and trust among different healthcare systems and organizations, and to enable secure and efficient collaboration that can improve patient outcomes.

Resource federation will have two subcomponents (DLT components): the access component and the resource component. The main functions of these components are similar, but there is a crucial difference: the access component records access requests which are requested to get access to a database of another hospital for example, while the resource component records resource requests which are requests to use a resource, like equipment of another department for example. However, in the access component, if an access request is granted and the transaction is recorded in the blockchain, this transaction can be used for future reference. For example, if someone requests access to a GPU for a week and the request is granted, the GPU administrator can check the blockchain transaction the following week to see if access should still be granted. If the week has passed, the administrator will be able to see from the transaction that access should no longer be granted, so the request will be denied.

The architecture of DLT components is as follows: a DLT component consists of an ordering service, certificate authorities (CAs), smart contracts, and a gateway for communication with the outside world. The ordering service and CAs are part of the blockchain network, and the functions of these components are defined in the smart contracts.

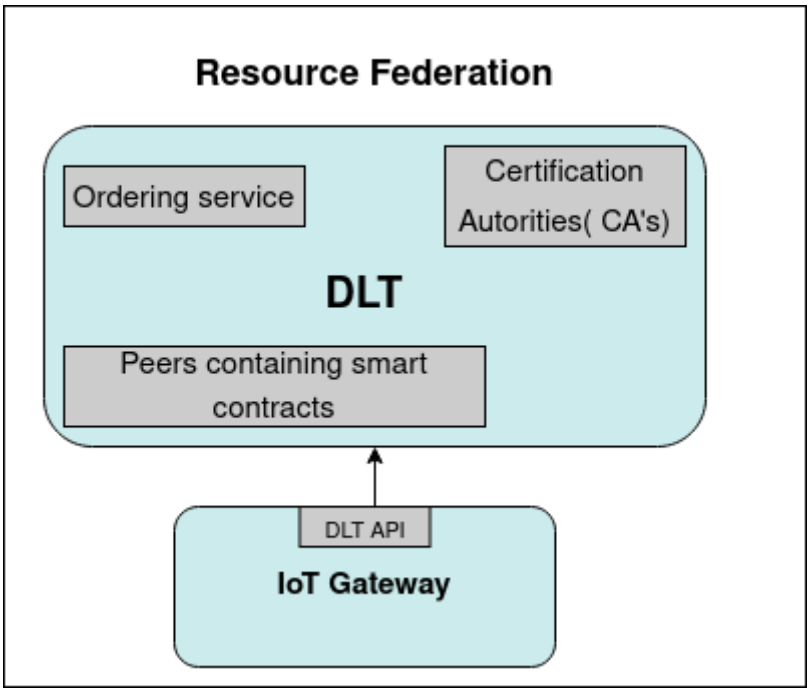

Figure 4: Resource federation component's architecture.

The ordering service in Hyperledger Fabric is responsible for ordering and distributing transaction data to all peers in a network. It is a critical component of the Hyperledger Fabric system and ensures that all transactions are properly ordered and recorded in the ledger.

Certificate Authorities (CAs) are entities that issue digital certificates to validate the identity of participants in a Hyperledger Fabric network. They play an important role in ensuring the security and privacy of transactions on the network by verifying the identity of participants and encrypting communications between them.

Smart contracts in Hyperledger Fabric are also known as chaincode and are used to define the rules and logic for transactions in the network. They are self-executing contracts with the terms of the agreement directly written into code. They automate the process of verifying and executing transactions, reducing the need for manual intervention and ensuring a high level of trust and transparency in the network. In ODIN platform, smart contracts are used to make all the recording of the transactions and the queries to the ledger possible. Both access component and resource component use smart contracts to enable functionalities and interaction with the blockchain network.

2.3.1 Access component

This component will log critical actions that happen during the data exchange between ODIN stakeholders to allow for transparency, auditing, non-repudiation and accountability of actions during the data exchange. In Figure 5 there is an example of this data exchange.

It will also log access requests and identified security events to help to provide digital evidence and resolve conflicts between stakeholders, when applicable. If any requirement of filtering prior to logging, a filtering module will be considered to be deployed. The DLT API is the candidate component for performing any filtering.

When a department wants to access something, e.g. a database, the request as well as the transaction of granting access, are recorded in the ledger of the blockchain.

An example of an input that the access component will receive, in a json format, is as follows:

`sender:`	`"id_of_sender",`
`receiver:`	`"id_of_receiver",`
`request:`	`"GPU",`
`timestamp:`	`"1675418872"`

The decision-making process to determine the specific contents of the ID's (sender and receiver) will be undertaken in the next phase of the project, along with other important processes that are still being finalized. Updates will be provided on the progress of this process, and once the conclusions are finalized, they will also be included in the project documentation regarding the ID requirements.

An example of an access request is shown in Figure 5. An instance wants to make a request to access a pre-trained AI model. The instance's gateway forwards the request to the resource federation, and the request is recorded in the ledger. If the request is granted, the AI can see it through the ESB and provide access. If the request is denied, the instance receives an "ACCESS DENIED" message.

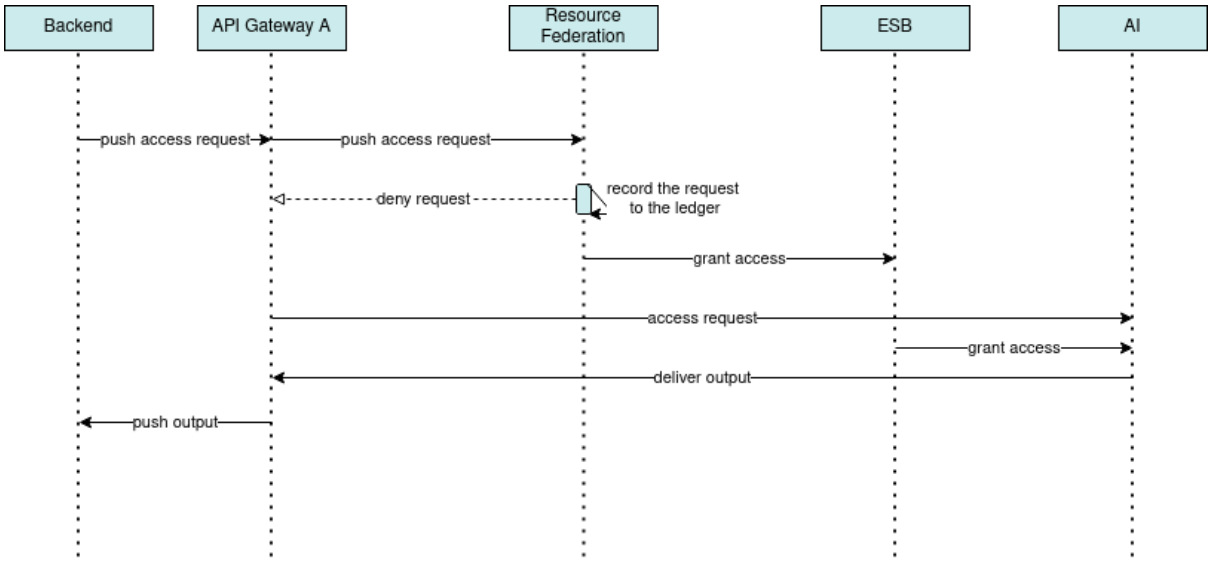

Figure 5: Sequence diagram for an example of resource federation's access component.

2.3.2 Resource component

This enabler will log essential activities that occur amongst ODIN stakeholders during resource exchange to provide for transparency, auditing, non-repudiation, and accountability of actions.

When necessary, it will also track resource requests and recognized security incidents to assist in providing digital proof and resolving issues amongst stakeholders.

When a department wants to use a resource of another department or hospital, the request as well as the transaction of granting the resource, are recorded in the ledger of the blockchain.

An example of an input that the resource component will receive, in a json format, would be as the following:

sender:	"id_of_sender",
receiver:	"id_of_receiver",
resource:	"lorazepam",
timestamp:	"1675418872"

An example of a resource request is following the same process as in **Error! Reference source not found.**. An instance wants to make a request to access an IoT sensor. The instance's gateway forwards the request to the resource federation, and the request is recorded in the ledger. If the request is granted, the IoT sensor can see it through the ESB and provide access. If the request is denied, the instance receives an "ACCESS DENIED" message.

2.3.3 DLT API

In order to log a transaction, an instance has to make a post request to the API of the resource component. The API then will send the DLT block, where the transaction proposals will be sent to peers, owned by the organizations, specified by the smart contract endorsement policy. The transaction proposal serves as input to the smart contract, which uses it to generate an endorsed transaction response, which is returned by the peer node to the client application (API).

It's these transactions responses that are packaged together with the transaction proposal to form a fully endorsed transaction, which can be distributed to the entire network.

Resource federation will have two components, access component and resource component, to handle the two types of requests. Bellow there are two tables, one for each component's API that presents how each of them can be accessed by other components.

Table 2: Resource component.

Component	Resource component		
Function name	Add resource		
Description	Adds a resource to the resource catalogue.		
Inputs	**Name**	**Type**	**Description**
	sender	string	The sender's ID, whether this is a name, a company name, a department or a combination of the above.
	receiver	string	The receiver's ID, whether this is a name, a company name, a department or a combination of the above.

	resource	string	The name of the resource to add. Also, if the sender wants can add more information here (e.g. a description).
	timestamp	Timestamp	The time and date in timestamp format.
	Name	**Type**	**Description**
	success	boolean	Whether the operation finished successfully.

Table 3: Access component.

Component	Access component		
Function name	Add access request		
Description	Adds a request to the request catalogue.		
Inputs	**Name**	**Type**	**Description**
	sender	string	The sender's ID, whether this is a name, a company name, a department or a combination of the above.
	receiver	string	The receiver's ID, whether this is a name, a company name, a department or a combination of the above.
	request	string	The ID of the item that the sender is requesting to access (e.g. the name of an AI model). Also, if the sender wants can add more information here (e.g. a description).
	timestamp	Timestamp	The time and date in timestamp format.
Outputs	**Name**	**Type**	**Description**
	success	boolean	Whether the operation finished successfully.
	error	string	An error message in case of an unsuccessful operation.

2.3.4 Use cases

DLT can be applied in many use cases. Some of them are listed below:

- Trustworthy data sharing for partners reluctant to share data due to privacy concerns. If a hospital has personal information about a patient's condition and that patient is in a different hospital, the first hospital can share patient's information safely.
- Support federated learning AI data sharing, which is described in more details in D6.4 and D6.6. The main instance has the permission to access a model of another instance. Different departments of the same hospital or even another hospital, can have access to the same AI model.
- Support hospital-to-hospital sharing of data and services. If a hospital has minimum resources and a big event occurs (e.g., a big road accident or a building collapse (RUC c)) needs access to another's hospital resources (e.g., medical equipment, GPUs, etc.), it can request access to these resources.
- Support within-hospital sharing of services:
    - Sharing between separate instances within the same hospital, e.g., medical and research departments. The research department can work on the development of diagnostic and therapeutic modalities associated to vitreo-retinal surgery, so the medical department shares its data during the development and it can benefit from the results after.
    - Domiciliary hospitalization (i.e., at home). A hospital makes a request to access a home's data, that has the ODIN platform installed. A patient that is in a coma condition can be in his family home with an ODIN platform installed to record or help with his condition. The hospital requesting access to the patient's information can detect any changes in patient's health (e.g., lack of nutrients).
    - Support the use of remote (cloud or hospital) resources. If a hospital doesn't have enough processing power, it can make an access request to a cloud service (e.g., for emergency situations).
    - Support data and service sharing of campaign hospitals, e.g., created on the spot during a war.
- Support sharing other kinds of resources (not KERs), e.g., drugs, between hospitals or departments.

3 Resource Choreographer

3.1 The Resource Choreographer

The resource choreographer is the service enabling the orchestration of the KER and services to achieve new business objectives.

Using visual tools or defining workflows and rules with standard BPMN and BMN notation, users with some technical training but also business users, will define new services, workflows and business cases, using currently available resources from the platform.

The Resource Choreographer enables hospitals to adopt technologies in a faster way as new technologies can be reused beyond its specific purpose, being integrated in flows that enable new use cases. Such advantage makes easier the procurement decisions as the money spent in a specific solution can be recovered before.

Resource Choreographer will help automating tasks, acting upon alarms and notifications that will fire the workflows and managing the resources states.

3.2 Main technologies

3.2.1 KOGITO

Kogito is a cloud-native business automation technology for building cloud-ready business applications. Kogito is optimized for a hybrid cloud environment and adapts to your domain and tooling needs. The core objective of Kogito is to help to adapt a set of business processes and decisions into the domain-specific cloud-native set of services.

In concrete ODIN is very interested on using Kogito both as a normal workflow and as a Serverless workflow because the Serverless Workflow enables defining declarative workflows that control event-driven, serverless applications.

Serverless Workflow orchestration is aimed to define logical steps of execution declaratively (no code) for cloud-native services. Service orchestration can be achieved using process diagrams in Business Process Model and Notation (BPMN), but, the Serverless Workflow is more focused on function-level steps of execution, and not in business logic. Serverless Workflow is also perfect for microservice orchestration and allows writing workflows in formats (JSON or YAML) that could be better tailored for developing applications in cloud or container environments with a serverless mindset.

Each of the approaches will be used where needed. For example, an emergency or disaster preparedness case (RUC C) could be implemented using both approaches, but a workflow based on BPMN could manage more complex cases with several roles performing approvals.

Logistic use cases such as RUC B UC1, or Medical location use cases from RUC B UC2 can take better advantage of Serverless Workflows.

3.3 Resource Choreographer components

3.3.1 Architecture

Kogito can use REST API call to use external services, but also can interact with messages coming from and also send messages to Kafka, which is the implementation for the ESB pointed in the architecture, defined in D3.11 ODIN Platform and worked on T4.2. This is very important and to understand how Kogito interacts with the ODIN ESB, the following diagram is included.

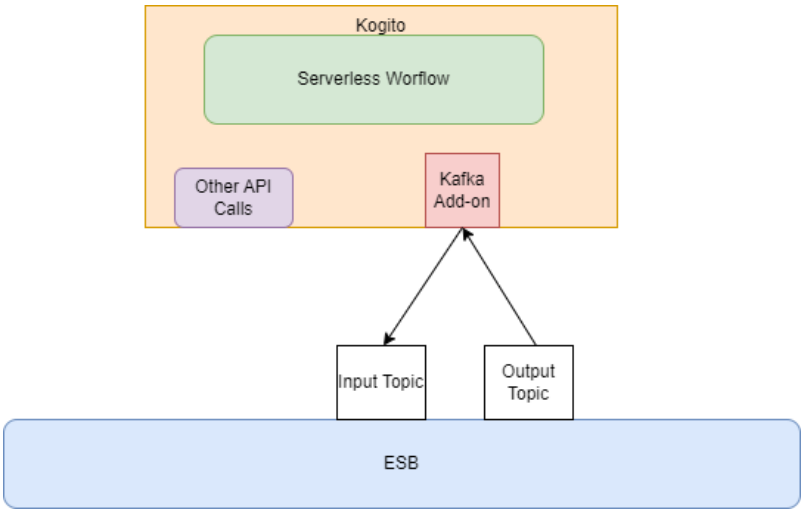

Figure 6: Components of Resource Choreographer.

Kogito will run several workflows that will interact with the rest of ODIN platform using the add-on provided by Kogito to connect to Kafka, which allows publish and subscribe to messages. This interaction drives the behaviour of acting upon events received, the workflow processing and actions taken and followed by components receiving the messages forwarded by Kogito.

It is important to remark that messages coming to and from Kogito, must follow CloudEvents[1] specification. This requirement perhaps is too hard to be followed by all the components, as it forces messages to be less than 64KB. Therefore, to minimize dependencies, a group of Connectors would be placed to transform the messages and interact with more topics and the Resource Choreographer. Messages do not have any other model limitation beyond the memory footprint.

Kogito also could use direct service API calls when available and needed in case no integration through the ESB is available. As an example, under RUC A we have rehabilitation use case, where

[1] https://cloudevents.io/

a patient must perform a several exercises and an AI controls the execution. Once the exercise is performed, the Resource Choreographer gets the results from the AI. Finally, the Resource Choreographer it determines with a rule if the exercise has been performed ok or not. The results then could be sent to the hospital HIS to integrate the data into patient's medical record history. To send the information to the HIS, it could be performed with a direct call using the HIS API, for example using FIHR protocol.

3.3.2 The workflow

The workflow specification can be expressed as a JSON file following this template:

```
{
  "id": "sampleWorkflow",
  "version": "1.0",
  "name": "Sample Workflow",
  "description": "Sample Workflow",
  "functions": [],
  "events": [],
  "states": []
}
```

Figure 7: Serverless Workflow template expressed as JSON.

Functions, events, and states are the minimal components for defining the orchestration behaviour for the services. The workflow definition can be expressed using only one or all type of components. The component are as follows:

- The Functions definitions are reusable components that can be used to define invocation information about services that could be invoked during workflow execution.
- The Events definitions are reusable components that define all consumed and produced events during workflow processing.
- The State definitions allow to define the workflow states and determine what the workflow should do.

Using the definitions, Kogito can generate scaffolds for some services and orchestrate them.

In the following pictures, a workflow to control a rehabilitation session and decide if the exercises were performed correctly is presented using the events and states defined of the use case.

```
{
  "id": "applicantworkflow",
  "name": "Rehabilitation Workflow",
  "expressionLang": "jsonpath",
  "version": "1.0",
  "start": "HandleNewRehabilitation",
  "events": [
    {
      "name": "NewRehabilitationEvent",
      "source": "",
      "type": "Rehabilitations"
    },
    {
      "name": "RehabilitationFinishedEvent",
      "source": "",
      "type": "finish"
    },
    {
      "name": "RehabilitationStartEvent",
      "source": "",
      "type": "start"
    },

    {
      "name": "RehabilitationDecisionEvent",
      "source": "",
      "type": "decisions"
    }
  ],
  "states": [
    {
      "name": "HandleNewRehabilitation",
      "type": "event",
      "onEvents": [
        {
          "eventRefs": ["NewRehabilitationEvent"],
          "actions": [
            {"name": "publishAction",
             "eventRef": {
              "triggerEventRef": "resumeEvent",
              "data": "{exerciseData: \".NewRehabilitation\"}"
          }
        }
          ]
        }
      ],
      "transition": "WaitRehabilitation"
    },
    {
      "name": "WaitRehabilitation",
      "type": "event",
      "onEvents": [
        {
          "eventRefs": ["RehabilitationFinishedEvent"],
          "actions": []
        }
      ],
      "transition": "VerifyNewRehabilitation"
    }
    ,
    {
      "name": "VerifyNewRehabilitation",
```

Figure 8: Serverless Workflow example 1 of 2.

```
      "type": "switch",
      "dataConditions": [
        {
          "condition": "{{ $.[?(@.correlation >= 0.75)] }}",
          "transition": "HandleApproved"
        },
        {
          "condition": "{{ $.[?(@.correlation < 0.75)] }}",
          "transition": "HandleDenied"
        }
      ]
    },
    {
      "name": "HandleApproved",
      "type": "inject",
      "data": {
        "decision": "Exercise passed"
      },
      "end": {
        "produceEvents": [
          {
            "eventRef": "RehabilitationDecisionEvent"
          }
        ]
      }
    },
    {
      "name": "HandleDenied",
      "type": "inject",
      "data": {
        "decision": "Repeat the exercise"
      },
      "end": {
        "produceEvents": [
          {
            "eventRef": "RehabilitationDecisionEvent"
          }
        ]
      }
    }
  ]
}
```

Figure 9: Serverless workflow example 2 of 2.

3.4 Situations that can be solved using the Resource Choreographer

A list of useful tasks that can be driven by the Resource Choreographer (RC) is here to understand more in depth its responsibility. On the other hand, it must be remarked that the RC is capable of solving new UC using KER capabilities as long as they expose the right features to do so.

Table 4: Useful tasks driven by Resource Choreographer.

Use Case	Workflow	Description
Disaster Preparedness	Evacuation plan	In case of fire or any disaster that requires evacuation, the RC could send the right messages to the KERS. Robots and people could be driven to a safe place.
AI Management	Workflow to define an AI behaviour	With AI usual tasks for a processing workflow, the users could define workflows to treat any kind of data and problem. The workflow could indicate when to start, what data to pick, the actions to be taken and the place where the results should be placed.
AI Management	Training Workflow	The resource choreographer can schedule when a new training should be performed and which data use. For example, in case several bad classifications or problems with a model are detected manually or automatically, the RC can call the AI to perform an automated training and enhance the model.
RUC A	Delivery and reposition management	In case there is a lack of any fungible material, such as catheter, towels, etc in a service, a user or an automated alarm could trigger a workflow that automates calling a robot to replenish the material.
General management	Activate/Deactivate KER's	Controlling activation or deactivation of a KER could potentially be managed from the RC

3.4.1 Messages, states, tasks and data types to be used to orchestrate the services with the RC.

In the following table, several states, tasks and events are proposed to tackle some of the previous use cases.

Table 5: Proposed states, tasks and events.

Workflow	Events	Tasks	State	Description
Evacuation plan	securityEvent, securityApproval, performEvacuation	approveEvacuation	WaitSecurityEvent, waitEvacuationApproval, prepareEvacuation	The workflow would be waiting for a securityEvent that could came from any AI or IoT system. Then, it would wait for a human approval, and upon approval it would send messages to performEvacuation.
Delivery and reposition management	newOrder, fungibleAvailable, fungibleNotAvailable, performTask	CheckFungible, checkRobot, sendRobot	waitForOrder, waitForFungible, waitForRobot	The workflow can be waiting for a new order. Then checks for the fungible. Then requests a robot and finally sends the robot to finish the task.
Creating or updating new documentation	newKER	updateDocumentation	waitForNewKer	The workflow would wait to receive a message confirming a new KER. Then would request the Documentation component to update the documentation.
Activate / Deactivate KER's	deactivateKER, activateKER	executeTask	WaitForKerManagement	The admin could request to deactivate or activate a KER,

				sending a deactvateKER message. Then the choreographer would send a task to the KER to disconnect

4 Conclusion and next steps

Two of the major objectives of the ODIN project are decentralization of the platform implementation and narrowing the gap between resource providers and healthcare organizations. The components described in this deliverable attempt to address these two objectives:

- The Distributed Ledger Technologies (DLTs) provide a decentralized and secure method for recording and sharing data and transactions. DLTs, such as blockchain-based systems like Bitcoin and Ethereum, have the potential to transform various industries by offering a tamper-resistant digital ledger that can be shared among multiple participants, thus reducing the need for intermediaries and increasing trust and transparency. In the case of the ODIN platform, DLT-related components are implemented to address privacy, trust, and decentralization. The use of a permissioned blockchain with verified participants and an endorsement policy ensures that data is secure and reliable, while data pre-screening is a critical step to ensure the integrity and accuracy of the recorded data. By adopting DLTs, ODIN platform is able to offer improved security, transparency, and efficiency in healthcare.
- The Resource Choreographer component provides a way for users, technical users but also business users such as medical staff or hospital manager, to design custom application workflows that make use of the available resources and target specific use case scenarios. This narrows the gap between resource providers and healthcare organizations, since new resources can readily be embedded in new applications designed by the healthcare authorities.

This deliverable is following (D4.5) as the second version in a series of three deliverables. While the first version was focusing on the design of the two components by introducing architectural elements, functionality and technology survey, this version is covering the actual development of the components, further analysing their functionalities and architecture, outlining the adopted technologies and introducing the corresponding workflows of the components. The third and final version of the deliverable (D4.7) will conclude the development of the components in their final version with the integration in the RUC's.

This version is also introducing real use cases where the DLT and RC components have a crucial role, by associating the functionalities of the components in a hospital environment with the RUCs that will be implemented in the ODIN project. With regards to the implementation of the project the next steps will be focusing on the adoption and integration of DLT and RC components by the ODIN platform, by putting them in to practice under the RUCs scenarios.

5 References

[1] Nofer, M., Gomber, P., Hinz, O., & Schiereck, D. (2017). Blockchain. Business & Information Systems Engineering, 59(3), 183-187.

[2] Swanson, T. (2015). Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. Report, available online.

[3] Monrat, A. A., Schelén, O., & Andersson, K. (2019). A survey of blockchain from the perspectives of applications, challenges, and opportunities. IEEE Access, 7, 117134-117151.

[4] Theodouli, A., Arakliotis, S., Moschou, K., Votis, K., & Tzovaras, D. (2018, August). On the design of a blockchain-based system to facilitate healthcare data sharing. In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE) (pp. 1374-1379). IEEE.

[5] Azaria, A., Ekblaw, A., Vieira, T., & Lippman, A. (2016, August). Medrec: Using blockchain for medical data access and permission management. In 2016 2nd international conference on open and big data (OBD) (pp. 25-30). IEEE.

[6] Benchoufi, M., & Ravaud, P. (2017). Blockchain technology for improving clinical research quality. Trials, 18(1), 1-5.

[7] Nugent, T., Upton, D., & Cimpoesu, M. (2016). Improving data transparency in clinical trials using blockchain smart contracts. F1000Research, 5.

[8] IBM. What are smart contracts on blockchain? Source. Accessed on 18 March 2022.

[9] Ethereum. Introduction to smart contracts. Source. Accessed on 18 March 2022.

[10] Szabo, N. (1997). Formalizing and securing relationships on public networks. First monday.

[11] Mik, E. (2017). Smart contracts: terminology, technical limitations and real world complexity. Law, Innovation and Technology, 9(2), 269-300.

[12] Nzuva, S. (2019). Smart contracts implementation, applications, benefits, and limitations. School of Computing and Information Technology, Jomo Kenyatta University of Agriculture and Technology, Nairobi, Kenya.