D3.1: Collaboration Governance Ledger and Smart Contracts

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101069782

Project & Document Information

Grant Agreement No	101069782	**Acronym**	URBANE
Project Full Title	UPSCALING INNOVATIVE GREEN URBAN LOGISTICS SOLUTIONS THROUGH MULTI-ACTOR COLLABORATION AND PI-INSPIRED LAST MILE DELIVERIES		
Call	HORIZON-CL5-2021-D6-01		
Topic	HORIZON-CL5-2021-D6-01-08	**Type of action**	IA
Coordinator	INLECOM INNOVATION		
Start Date	01/09/2022	**Duration**	42 months
Deliverable	D3.1	**Work Package**	WP 3
Document Type	OTHER	**Dissemination Level**	PU
Lead beneficiary	Kühne Logistics University gGmbH (KLU)		
Responsible author	Rod Franklin (KLU)		
Contractual due date	29/02/2024	**Actual submission date**	[29/02/2024]

Disclaimer and Acknowledgements

Co-funded by the European Union

This project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101069782

Disclaimer

Copyright message

Authoring, Revision & QA Information

Deliverable Contributors	
Contributor Name	**Organisation (Acronym)**
Rod Franklin	Kühne Logistics University gGmbH (KLU)
Harris Niavis	Inlecom Innovation (INLE)
Malte Spanuth	Kühne Logistics University gGmbH (KLU)
George Misiakoulis	Konnecta Systems Ltd (KON)

Version History				
Version	**Date**	**%**	**Changes**	**Author**
0.1	21/11/2022	5%	ToC and Introduction	Rod Franklin
0.2	09/02/2024	80%	First draft	Rod Franklin
0.3	13/02/2024	90%	Second draft after review	Rod Franklin
1.0	29/2/2024	100%	Final draft after review process	Rod Franklin

Quality Control (includes peer & quality reviewing)			
Date	**Version**	**Name (Organisation)**	**Role & Scope**
[30/11/2022]	0.1	Yasanur Kayikci (VLTN)	ToC Approval
[14/02/2024]	0.3	Xavier Brusset (SKEMA)	Peer Review
[19/02/2024]	0.4	Yasanur Kayikci (VLTN)	QM Approval
[22/02/2024]	0.4	Maria Kampa (INLE)	PM review
[27/02/2024]	0.6	Ioanna Fergadiotou (INLE)	Project Coordinator approval

Executive summary

The URBANE project's approach to last mile delivery operations is based on the Physical Internet (PI) framework. This framework, drawing upon similarities in the movement of freight to the movement of data packets over the Internet, assumes that logistics service providers collaborate and share assets to generate the greatest efficiencies and effectiveness in the delivery of freight for their customers. Because of the commercial nature of freight operations, it is an absolute requirement of these service providers that they can trust their collaboration partners and that any system used in their operations is secure and trustworthy. In addition, these service providers require visibility to the services that their partners execute on their behalf to ensure that service levels are met and, should problems arise, pro-active corrective actions can be taken. To provide the strict security, privacy, and trust services demanded by these actors requires an infrastructure that is as secure as possible, transparent, configurable for contract monitoring, and non-reputable. Given the technological landscape available today, this means that a blockchain infrastructure, deploying dynamic and configurable Smart Contracts, and accessible through Decentralized Identifiers (DIDs) and Verifiable Credentials (VC) is the most appropriate approach to addressing this requirement.

This document covers the blockchain infrastructure developed under Task 3.3: Collaboration Governance Ledger, consensus protocols and Smart Contracts of the URBANE project. This task is responsible for building the blockchain infrastructure composed of a scalable blockchain service (Hyperledger Fabric), blockchain based security for the URBANE platform (DIDs and VCs), and smart contract services to support Living Lab last mile delivery event monitoring and non-repudiation. The sections of the report discuss each of these developments. In addition, the deliverable describes the process requirements for each Living Lab associated with the utilization of the smart contracting services.

The integration of commercial, governmental, and societal players into a workable urban logistics model that addresses the triple bottom line focus of all players (people, profit, planet) is a difficult collective action problem to resolve. The key to its resolution is trust. For trust to occur the system employed must be trustworthy. This implies that such a system must address the issues of security, privacy, equity, transparency, and usability in an open and fair manner. The blockchain infrastructure of the URBANE platform described in this document provides one component for building a trustworthy system for last mile logistics operations in an urban environment. By providing security through the use of state-of-the-art digital identification processes, an immutable ledger, and service level monitoring through smart contracts, the URBANE blockchain infrastructure ensures that users can rely on a system built on leading edge access control. The URBANE platform, of which the blockchain infrastructure is a service component, provides additional security services creating a system that is trustworthy. This is a necessary, but not sufficient, condition for creating trust. Significant work on addressing the collective action problem of achieving the environmental and social outcomes desired by the EU's various sustainability initiatives is still required. However, it is hoped that this contribution helps in moving adopting cities closer to achieving the goals of these initiatives.

Contents

List of figures

List of tables

Glossary of Terms and Acronyms

Acronym / Term	Description
AGV	Automated Guided Vehicle
ADV	Autonomous Delivery Vehicles
API	Application Programming Interface
BFT	Byzantine Fault Tolerance
CA	Certificate Authority
CCAAS	Chaincode as a Service
CFT	Crash Fault Tolerant
Chaincode	Hyperledger Fabric term for Smart Contracts
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DID	Digital Identifier
EVM	Ethereum Virtual Machine
gRPC	Google Remote Procedure Call (now simply gRPC without Google)
HLF	Hyperledger Fabric (URBANE's blockchain foundation)
IAM	Identity Access Management
ID	Identifier
JVM	Java Virtual Machine
LL	Living Lab
LSP	Logistics Service Provider
MSP	Membership Service Provider
NFS	Network File System
OAuth	Open Authorization
PaaS	Platform as a Service
PBFT	Practical Byzantine Fault Tolerance
PI	Physical Internet
RAFT	Resource Aggregation for Fault Tolerance
REST	Representational State Transfer
RPC	Remote Procedure Call
SLA	Service Level Agreement
SSI	Self-Sovereign Identity
TLS	Transport Layer Security
TLSCA	Transport Layer Security Certificate Authority
TPS	Transactions Per Second

UI	User Interface
URL	Uniform Resource Locator
VC	Verifiable Credentials
VDR	Verifiable Data Registry

1 Introduction

The URBANE project proposes to support the European Union's objective of building safe and sustainable last mile transport operations through the development of repeatable and scalable innovative last mile delivery solutions using a novel service collaboration model based on an analogy to the Internet called the Physical Internet. Through the deployment, monitoring, and testing of innovative last mile delivery services in four Lighthouse Living Labs (Helsinki, Bologna, Valladolid, and Thessaloniki) baseline performance data, implementation models, and operational requirements will be documented and made available to two Twinning Living Labs. To transfer the knowledge gained from the four Lighthouse Living Labs, URBANE will develop an Innovation Transferability Platform. This platform will provide users with configurable operational optimization tools, simulation tools, real time monitoring services (digital twin simulation and delivery operations monitoring), and support for future application development based on city specific requirements. In a manner like the "Platform as a Service (PaaS)" layer in cloud-based infrastructures, the URBANE Innovation Transferability Platform is a first step in realizing a general purpose "Urban Cloud" structure allowing cities to dynamically manage their infrastructure while providing value added services to their citizens (Figure 1).

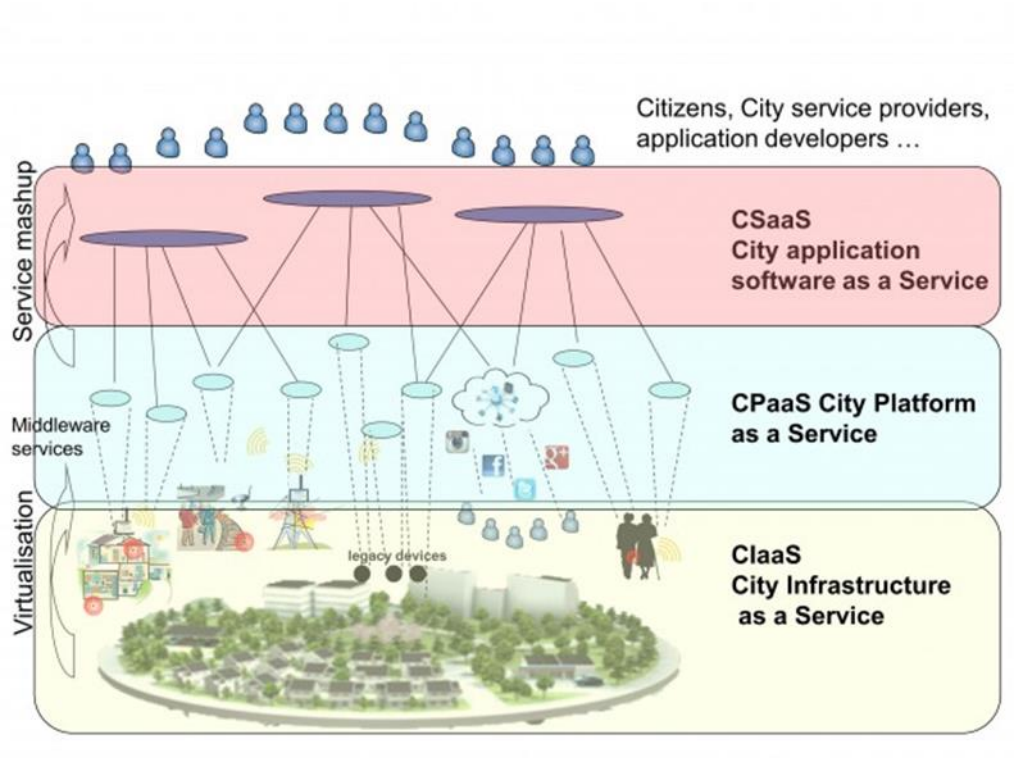


FIGURE 1: CITY AS AN "URBAN CLOUD" (SOURCE: CLOUT FP7 PROJECT)

Work Package 3 of the URBANE project is responsible for the design and development of the information technology infrastructure for the project. The scope of work to be undertaken in the work package covers the development of tools to measure and assess the effects of the innovative last mile delivery services deployed by the Living Labs. The tools being developed and enhanced to accomplish this objective include an enhanced digital twin platform (based on a digital twin platform originally developed in the Horizon 2020 LEAD project), operational optimization tools that can be connected to one another based on the operational problem being addressed and whose output can be fed into the digital twin, and a blockchain and smart contract service to document various delivery performance variables in a non-

reputable manner. Work Package 3 is also responsible for the development of performance indicators and an Impact Assessment Radar that will be used to determine the impact of the various innovative last mile approaches on a city's economic, social, and environmental objectives for urban logistics operations.

Work Package 3 consists of seven primary tasks. These tasks are:

1. Architecture Design
2. Impact Assessment Methodology and KPIs
3. Collaboration Governance Ledger, Consensus Protocols and Smart Contracts
4. Modelling Framework and Agent-Based Models
5. AI-Driven Models and Services
6. Digital Twins Infrastructure and Open Model Library
7. Data-Driven Impact Assessment Radar

This deliverable covers the third task of the work package, the development of a collaboration governance ledger, consensus protocols and smart contracts.

1.1 Task 3.3: Collaboration Governance Ledger, Consensus Protocols and Smart Contracts

The primary responsibility of Task 3.3 in the project is the development of a blockchain infrastructure that enables secure sharing, operation, and decision making for urban users of the URBANE Innovation Transferability Platform. To accomplish this objective, Task 3.3 is composed of three sub-tasks:

1. Setup of Blockchain Infrastructure
2. Identity Management Rooted on Blockchain
3. Smart Contracts Design and Deployment

The relationship of these sub-tasks to the overall task is shown in Figure 2:

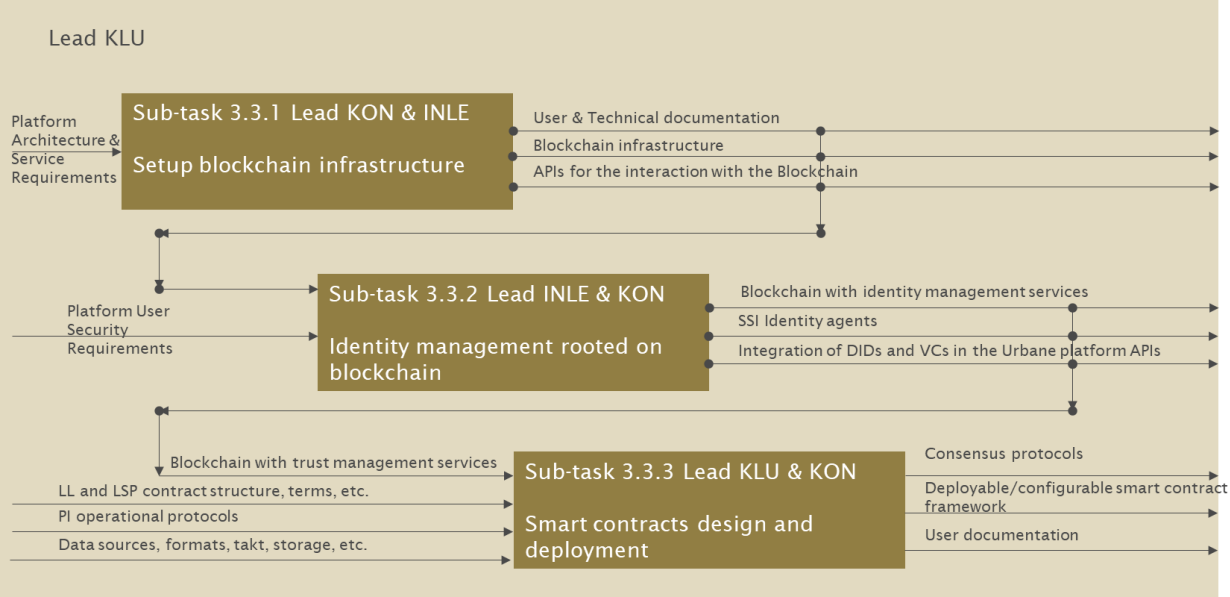


FIGURE 2 : TASK 3.3 ORGANIZATION

As has been noted, URBANE's approach to last mile delivery operations is based on the Physical Internet framework. This framework, drawing upon similarities in the movement of freight to the movement of data packets over the Internet, assumes that logistics service providers (LSPs) collaborate and share

assets to generate the greatest efficiencies and effectiveness in the delivery of freight for their customers. D1.1: URBANE Framework for Optimized Green Last Mile Operations identified the issue of trust as a key issue for LSPs in participating in collaboration efforts for last mile delivery of goods. The D1.1 deliverable identified several collaboration schemes that have worked for different cities. Each of these approaches employed a trusted entity to oversee the collaboration efforts and contractual commitments by the LSPs to ensure their participation. Unfortunately, the schemes identified in that deliverable were static in nature. In more dynamic collaboration efforts, which URBANE is focusing on, the concepts developed for these static models provide guidance for developing a dynamic collaboration framework, which is the focus of the URBANE Innovation Transferability Platform and its blockchain infrastructure that is the subject of this deliverable.

In addition to a trustworthy system, service providers require visibility to the services that their partners execute on their behalf to ensure that service levels are met and, should problems arise, pro-active corrective actions can be taken. To provide the strict security, privacy, and trust services demanded by these actors requires an infrastructure that is as secure as possible, transparent, configurable for contract and service level monitoring, and non-reputable. Given the technological landscape available today, this means that a blockchain infrastructure, deploying dynamic and configurable smart contracts, and accessible through Decentralized Identifiers (DIDs) and Verifiable Credentials (VC) is a key element in the URBANE platform for addressing user security requirements.

1.2 URBANE Outputs Mapping to GA Commitments

URBANE GA Item	URBANE GA Item Description	Document Chapter(s)	Justification
DELIVERABLE			
D3.1 **Collaboration Governance Ledger & Smart Contracts**	D3.1 will setup the Blockchain infrastructure and implement a distributed blockchain-based and privacy-preserving identity management scheme rooted on Blockchain. It will also define the consensus protocols, design and deploy the Smart Contracts for Wave 1 LLs.	*Sections 2, 3, and 4 describe the technical details of the blockchain, digital identity management, and smart contract infrastructure that has been developed for URBANE*	*The blockchain infrastructure that has been developed for the URBANE project, and which is integrated with the URBANE platform is described in this document. The blockchain infrastructure that has been developed addresses all functionality described in the GA for this component of the URBANE platform and the sections of this report cover how this infrastructure has been implemented. In addition, links to the GitHub repositories for the code that has been developed, as well as the process analyses performed for each LL that has driven the development of the smart contracts service is included.*
TASK			

***T3.3* Collaboration Governance Ledger, consensus protocols and Smart Contracts**	This subtask focuses on the design and development of a blockchain-based infrastructure that enables the secure sharing and storage of data between Urban Logistics communities. In addition, it supports the execution of smart contracts, aligned with the requirements from the Lighthouse and the Twinning LLs.	*Sections 2, 3, and 4 describe the technical details of the blockchain, digital identity management, and smart contract infrastructure that has been developed for URBANE*	*The blockchain infrastructure that has been developed for the URBANE project, and which is integrated with the URBANE platform is described in this document. The blockchain infrastructure that has been developed addresses all functionality described in the GA for this component of the URBANE platform and the sections of this report cover how this infrastructure has been implemented. In addition, links to the GitHub repositories for the code that has been developed, as well as the process analyses performed for each LL that has driven the development of the smart contracts service is included.*

TABLE 1: DELIVERABLE ADHERENCE TO GRANT AGREEMENT DELIVERABLE AND WORK DESCRIPTION

1.3 Deliverable Overview and Report Structure

This deliverable is organized around the sub-tasks that are part of WP 3 Task 3.3 and that have been realized in the URBANE project. The deliverable also provides an overview of how this blockchain-based infrastructure is being utilized in three of the Lighthouse Living Labs (Bologna, Thessaloniki, and Helsinki). The fourth Lighthouse Living Lab, Valladolid, will not be utilizing the smart contract component of the blockchain infrastructure due to the focus of its innovation project.

Section 2 addresses the reasons for selecting the specific blockchain infrastructure used in URBANE (Hyperledger), how this infrastructure is implemented within the URBANE platform architecture, and how the blockchain implementation used in the project operates. It should be noted that the blockchain infrastructure is a component service of the URBANE Transferability Platform. This deliverable focuses on the blockchain infrastructure that has been developed for the project and does not address the Transferability Platform itself or its development.

Section 3 of this deliverable covers the security approach taken in the project. This section covers the implementation of the Decentralized Identifier and Verifiable Credentials infrastructure, how this infrastructure is expected to work, and how it fits within the overall architecture of the Urbane platform.

Section 4 of the document covers the smart contract approach taken for the project. This section addresses the smart contract generation process, user setup of their smart contracts, the monitoring/visibility processes enabled through the smart contracts, the API for feeding event data into the smart contracts, and the architecture used to implement the smart contracts within the URBANE platform.

In Section 5 we discuss the implementation of the blockchain infrastructure, DIDs, and smart contracts for three of the URBANE Living Lab demonstrations. The implementations to be discussed cover the Bologna, Thessaloniki and Helsinki living labs and demonstrate different use cases for employing the URBANE blockchain services in support of Physical Internet based last mile services.

Section 6 concludes this deliverable with a summary of what has been accomplished to date, enhancements being considered for the blockchain infrastructure, and a general overview of how additional functionality requirements for follower cities can be addressed.

An Appendix concludes this deliverable. In the Appendix directions for finding the code used to implement the blockchain, DID, and smart contract infrastructure is presented so interested readers can review the current state of the services deployed.

2 Blockchain Infrastructure for the URBANE Platform

2.1 Hyperledger – The Blockchain Foundation and consensus protocols

A comprehensive survey of the state-of-the-art blockchain frameworks was carried out to identify the most suitable tools for the implementation of the blockchain infrastructure of the URBANE platform. Various frameworks such as Hyperledger Fabric [1], Quorum [2] and R3 Corda [3] were investigated and evaluated based on factors such as the supported consensus protocol, performance, security, modularity, openness, and size of the community supporting them.

2.1.1 Reviewed Blockchain Frameworks

Hyperledger Fabric

Hyperledger Fabric (HLF) stands out as the most widely embraced among the Hyperledger projects and holds a prominent position in the realm of private permissioned blockchains. Initially open-sourced by IBM in July 2017, it now enjoys the support of a large community comprising of 300 developers, 45 companies and 100+ individuals. HLF delivers a uniquely adaptable and extensible architecture, setting it apart from alternative blockchain solutions. It facilitates the execution of smart contracts, integrates pluggable consensus, employs a private data mechanism, and provides an identity management service. The Fabric network consists of Peer Nodes, responsible for executing smart contracts and hosting the ledger, an Orderer Service ensuring the consistency of the blockchain, by ordering and distributing blocks back to Peer Nodes, and an Identity management Service (referred to as Membership Service Provider or MSP) handling the identities of network components and users through X.509 certificates issued by a Certificate Authority (CA).

A notable advantage of HLF lies in its use of standard, general-purpose programming languages (Python, Go, Java, Nodejs) and its independence from relying on cryptocurrency, although it can support one if needed. In terms of performance, HLF can achieve an end-to-end throughput exceeding 3500 transactions per second and scales effectively to over 100 Peers, depending on network parameters [4]. Recent studies even suggest achieving throughputs of up to 20000 transactions per second in a HLF network, following some architectural adjustments [5].

Quorum

Consensys Quorum, acquired by ConsenSys from JP Morgan, transforms an already battle-tested blockchain implementation i.e., the public Ethereum network - into a permissioned blockchain with enhanced capabilities. It maintains an Ethereum client implementation in Go, GoQuorum, which complements Go-Ethereum, incorporating performance optimizations, confidential transactional and contractual executions, alternative consensus mechanisms, and smart contract-based peer permissioning.

Quorum operates within the same execution environment as its Ethereum-based counterpart, adhering to the rules of the Ethereum Virtual Machine (EVM). The EVM comprehends its own low-level language, akin to Assembly. Consequently, languages like Solidity and Vyper can be compiled to EVM bytecode, making them executable on both Ethereum and Quorum. Despite differences in syntax from conventional programming languages, the existing Ethereum developer community can directly apply their skills to Quorum.

The transactional throughput of Quorum depends heavily on transaction types and the chosen consensus mechanism. Private transactions in Quorum are routed through the Tessera module, which supports a certain degree of parallelization. This allows it to benefit from multiple CPUs, particularly when used with asynchronous RPC calls. On the other hand, public transactions in Quorum are handled by the Go-Ethereum client, benefiting from faster clock speeds due to its sequential nature.

The choice of consensus mechanism in Quorum significantly influences the Transactions Per Second (TPS) metric. Quorum defaults to two types of consensuses: Raft-Based Consensus and Istanbul BFT. Raft-based consensus achieves a steady TPS of around 100, while properly optimized Istanbul BFT has demonstrated TPS metrics reaching upwards of 450. Both consensus mechanisms in Quorum support transaction finality upon submission, like HLF and Tendermint.

Corda

Led by R3, Corda primarily aims to facilitate direct and highly confidential transactions using smart contracts for businesses in Banking, Capital Markets, Trade Finance, Insurance, and beyond. The objective is to reduce transaction and record-keeping costs while optimizing overall business operations *[6]*. However, many of these use cases appear to be in their early stages, if not still in the conceptual phase. While Corda shares certain concepts with blockchain technology, it diverges from being a conventional blockchain protocol. Instead, it adopts a similar approach but with a key design principle: sharing only necessary information.

In Corda, there is no unique ledger accessible to all nodes. The concept of a "fact" prevails, with these facts shared among a subset of network members. Unlike traditional blockchain setups, there is no global view of all network facts; nodes are aware only of the facts relevant to them. Smart contracts in Corda can be written in any JVM (Java Virtual Machine) based language, including Java or any language capable of compiling and generating JVM bytecode. Ensuring determinism, where the same input consistently produces the same output, is crucial for smart contracts. Corda achieves this by executing contracts in a Sandbox environment—a JVM with a whitelist of specific libraries callable by the contract.

Initially designed to address banking challenges, Corda's use cases extend beyond banking. However, the overrepresentation of the banking industry in Corda, driving R3's evolution, poses potential risks for other industries. This dominance may hinder agility in addressing simpler use cases and present challenges for industries beyond banking.

Summary and Selection

To sum up, HLF is an open-source framework for building enterprise-grade blockchain applications and offers a modular architecture for deploying, running, and managing smart contracts in a broad range of use cases. GoQuorum is an open-source, permissioned implementation of the Ethereum blockchain, designed to provide a secure, fast, and scalable platform, closely tied to ConsenSys, which may influence the direction of its evolution. Corda, on the other hand, is designed specifically for financial institutions and focuses on privacy, interoperability, and security.

After the thorough investigation of the state-of-the-art technologies and after discussions with the involved partners, HLF was chosen as the blockchain technological framework of the blockchain component for the following reasons:

- The privacy of data and security of transactions that can be achieved using channels and private data collections in a permissioned blockchain network.
- Scalability and performance of the network to accommodate the complexity and intricacies that exist across the last mile delivery sector.
- The modular architecture and pluggable consensus offered by HLF enables a great experimentation environment to tweak key parameters and install a custom consensus algorithm specifically designed for the IoT settings. HLF supports by default a Crash Fault Tolerance (CFT) consensus mechanism based on RAFT[1] which ensures fault tolerance, but also enables the configuration with other off-the-self BFT algorithms such as SmartBFT[2] for tolerance on adversary nodes and even custom implementations that can be more focused on the context e.g. a lightweight consensus for IoT [7].

[1] RAFT Consensus Algorithm, https://raft.github.io/
[2] SmartBFT library, https://github.com/SmartBFT-Go/consensus

- Its open-source licence and the large and diverse community that is developing, maintaining, and supporting the framework with contributions from various industries.

2.2 Blockchain Infrastructure Operation

The backbone of the URBANE Blockchain infrastructure is based on two cutting-edge technologies, namely HLF and Kubernetes that form an integrated framework to enable the administrator to orchestrate the deployment of a HLF network within a Kubernetes environment, as a robust and flexible foundation for building a blockchain application. Peer Nodes of HLF allow the deployment of smart contracts for transparency and integrity, while Kubernetes oversees and dynamically scales the performance. An extensive list of the framework's internal components is presented below:

Kubernetes Orchestration: At its core, Kubernetes functions as the orchestrator, seamlessly managing the deployment, scaling, and maintenance of containerized applications. In this context, HLF's modular components are containerized, offering the advantage of easy scalability and dynamic management of individual elements. The integration of Kubernetes empowers the network with automated deployment, ensuring optimal resource utilization and rapid adaptability to changing workloads.

Consensus Approach: Critical to the system's integrity is the consensus mechanism. HLF adopts a modular and pluggable consensus approach, tailoring its algorithmic underpinnings to specific use cases. Notably, Practical Byzantine Fault Tolerance (PBFT) and Raft emerge as reliable choices. PBFT guarantees consensus even in the face of malicious nodes, demanding a two-thirds majority for transaction validation. Meanwhile, Raft simplifies the consensus process by designating nodes as leaders and followers, streamlining decision-making across the network.

Orderer Services: The backbone of transaction sequencing lies within the jurisdiction of the Ordering Service. Tasked with managing the chronological order of transactions, this component receives endorsed transactions from Peer Nodes, arranges them into blocks, and disseminates these blocks uniformly across the network. The meticulous orchestration by the Orderer Service ensures a harmonized and unambiguous ledger state across all participating nodes.

Peer Node: Peer nodes are the active participants in the network, shoulder the responsibility of maintaining the ledger and executing smart contracts. These nodes engage in the consensus process, corroborating the validity of transactions and ensuring a unanimous view of the blockchain's state. Furthermore, they act as conduits for communication between external applications and other peer nodes, fostering a cohesive and interconnected blockchain ecosystem.

Smart Contracts (Chaincode): The heart of the decentralized applications running on HLF lies in the smart contracts, implemented as chaincode. These contracts encapsulate the business logic governing transactions, residing on Peer Nodes. Upon receiving endorsements from the majority of Peers, the chaincode executes, bringing automation, transparency, and integrity to the contractual agreements within the blockchain network.

Identity Management: Ensuring secure access to the blockchain network is a non-negotiable fact of the infrastructure. Robust identity management is achieved through cryptographic key pairs, certificates, and Membership Service Providers (MSPs). These elements collectively authenticate and authorize participants, safeguarding the network against unauthorized access and ensuring that only legitimate transactions find a place in the immutable ledger.

Logging and Monitoring: Transparency and accountability are maintained through robust logging and monitoring mechanisms. These tools provide a comprehensive view of smart contract executions,

consensus decisions, and overall network activities. By tracking performance metrics, detecting anomalies, and facilitating proactive interventions, these features enhance the overall reliability and health of the blockchain infrastructure.

The integration between HLF and Kubernetes enhances the infrastructure with inherent scalability and high availability. Kubernetes dynamic scaling capabilities, coupled with the distributed nature of HLF, enable the network to seamlessly adapt to fluctuating demands, ensuring uninterrupted service and resilience in the face of unforeseen challenges. The next subsections will present the URBANE Blockchain Infrastructure as part of the URBANE Platform and will also describe in detail the individual components.

2.3 Infrastructure Architecture

As part of the URBANE project, a custom implementation of a highly scalable, secure, and resilient blockchain infrastructure was designed and developed, available in the URBANE's GitLab repository (access can be provided upon request) https://gitlab.com/urbane-horizon-europe/blockchain/hlf-platform-k8s. As shown in Figure 3, the core components of the blockchain infrastructure are incorporated into a Kubernetes cluster and support the dynamic generation of SLA contracts between last mile delivery stakeholders through user-friendly interfaces, namely the Blockchain Dashboard, developed as a component of the URBANE Innovation Transferability Platform. Moreover, a list of real-time logistic events is shown in the Blockchain Dashboard, sourced by information sent from diverse Last Mile Communities backend systems. All the external APIs of the Platform are safeguarded using cutting-edge authentication and authorization approaches based on well-established standards (DIDs, VCs, OAuth). More details on the Identity management aspects of the URBANE Platform can be found in Section 3.

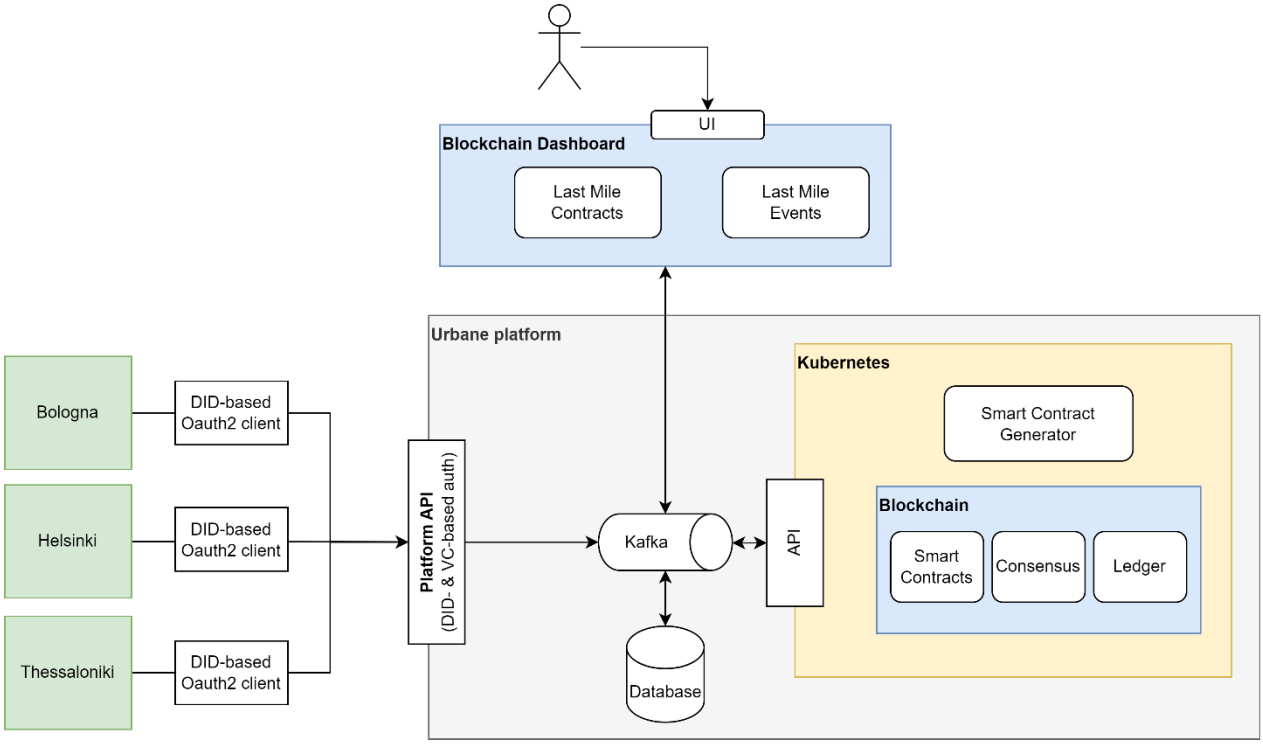


FIGURE 3: HLF-PLATFORM-K8S AS PART OF THE URBANE PLATFORM

The following sub-sections can be used as a reference by last mile communities to deploy a fully decentralized and scalable blockchain infrastructure.

2.3.1 Prerequisites and Installation

Deploying a multi-component system like HLF in a near-production environment poses significant challenges. This section delves into our approach to deploying HLF on Kubernetes employing the hlf-platform-k8s framework. The communication with the control plane is achieved through a library of scripts based on Kubernetes' command line tool (kubectl[3]) to communicate instructions and custom Manifest files for specific components deployment. Rather than relying on a monolithic automation script for the entire deployment, hlf-platform-k8s embraces a modular strategy, by providing separate manifests for each HLF component, including the Certificate Authority (Fabric-CA), Peer, CouchDB, and Orderer. The upcoming sections outline the steps required to create and operate a fully functional blockchain system, offering insights into the hlf-platform-k8s (more details can be found in Annex I) tailored deployment methodology.

Table 2 lists the version required for each of the components of the hlf-platform-k8s.

	Component	Version
1	Fabric Peer	2.4.6
2	Fabric Orderer	2.4.6
3	Fabric CA	1.5.5
4	CouchDB	3.2.2
5	NFS	4
6	Chaincode Go	1.18.2

TABLE 2: HLF-PLATFORM-K8S COMPONENTS VERSIONS

Further insights into the implementation of the hlf-platform-k8s framework are provided below:

Environment Variables: The project relies on an `.env` file containing essential environment variables crucial for its seamless operation. These variables are integral to configuring and customizing various aspects of the deployed HLF network.

NFSv4 Servers for Organizations: Each organization benefits from a dedicated NFSv4 server deployed in a distinct namespace (e.g., `fabric-network-${ORG}-nfsv4`). This strategic deployment ensures a shared filesystem across different components and pods within the organization, fostering cohesion and efficient communication among diverse elements.

Mutual TLS Encryption: Security is paramount, and the project ensures robust network communications through mutual TLS encryption. Every component, including Peer-to-Peer, Peer-to-Orderer, Peer-to-Chaincode, Orderer-to-Orderer, and Fabric-CA Client to Fabric-CA Server, operates with mutual TLS enabled, fortifying the security and confidentiality of the network.

[3] Kubernetes Command Line Tool, https://kubernetes.io/docs/reference/kubectl/

Fabric-CA Server Setup: Each organization hosts a Fabric-CA server within its dedicated namespace (e.g., `fabric-network-${ORG}`). This server manages two distinct CAs: an IdentityCA for creating identities (called **ca**) and a TLSCA for generating TLS certificates (called **tlsca**). Both CAs employ a hierarchy of Root and Intermediate CAs, forming secure chains of trust, meticulously loaded into Cert-Manager for further certificate management.

Orderer Organization Configuration: The Orderer organization, a critical component of the network, is configured with a minimum of three Orderer nodes, ensuring adherence to the RAFT consensus algorithm's minimum requirements. These nodes, with Admin Plugin enabled, facilitate seamless communication and channel management through the osnadmin[4] binary.

Peer Organization Configuration: Organizations, excluding the Orderer Organization, can deploy multiple Peer nodes, each with Fabric Gateway enabled for streamlined transaction submission. These Peers, registered as Anchor Peers, play a pivotal role in block dissemination, retrieving blocks directly from the ordering service.

Gossip Protocol Deactivation: The Gossip Protocol is intentionally deactivated, redirecting block dissemination responsibilities to Anchor Peers. This ensures efficient and direct block retrieval from the ordering service, optimizing network performance.

User Registration and MSP Setup: Upon deploying a Fabric CA Server, a custom Membership Service Provider (MSP) filesystem is created. While identity registration is performed by the Fabric CA Admin on the host, the enrolment process takes place directly within the Fabric CA Server Pod. Organizations can register one or multiple users, and the MSP setup is adaptable based on configuration.

Multi-Organization Deployment: The project supports the deployment of multiple organizations within the Kubernetes cluster, allowing organizations to join a shared channel.

Agnostic Manifests and Configurations: All Kubernetes Manifests YAML files are designed in an agnostic manner, irrelevant to organization-specific details. Instead, configurations from the `.env` file dynamically populate these manifests, ensuring a flexible and customizable deployment process.

Chaincode Deployment and Gateway Client API: Chaincodes are deployed using the Chaincode As A Service (**CCAAS**)[5] method, enhancing the modularity and maintainability of the deployed network. Applications interacting with the HLF network must utilize the Fabric Gateway Client API, with the hlf-platform-k8s framework leveraging the GO implementation for seamless integration with the network deployed on Kubernetes.

Figure 4 presents the internal technical architecture of the Kubernetes cluster deployed for URBANE, showing one of the Fabric Organisations namespaces, namely GEL Proximity. Similar namespaces exist in the infrastructure for the rest of the URBANE Organisations, simulating the different actors of the URBANE Last Mile Communities:

- Bologna: GEL, Bologna municipality, TYP, Due Torri, ITL
- Helsinki: Forum Virium Helsinki, LMAD, DB Schenker
- Thessaloniki: ACS, Certh

[4] onsadmin command in HLF, https://hyperledger-fabric.readthedocs.io/en/latest/commands/osnadminchannel.html

[5] Chaincode As A Service in HLF, https://hyperledger-fabric.readthedocs.io/en/latest/cc_service.html

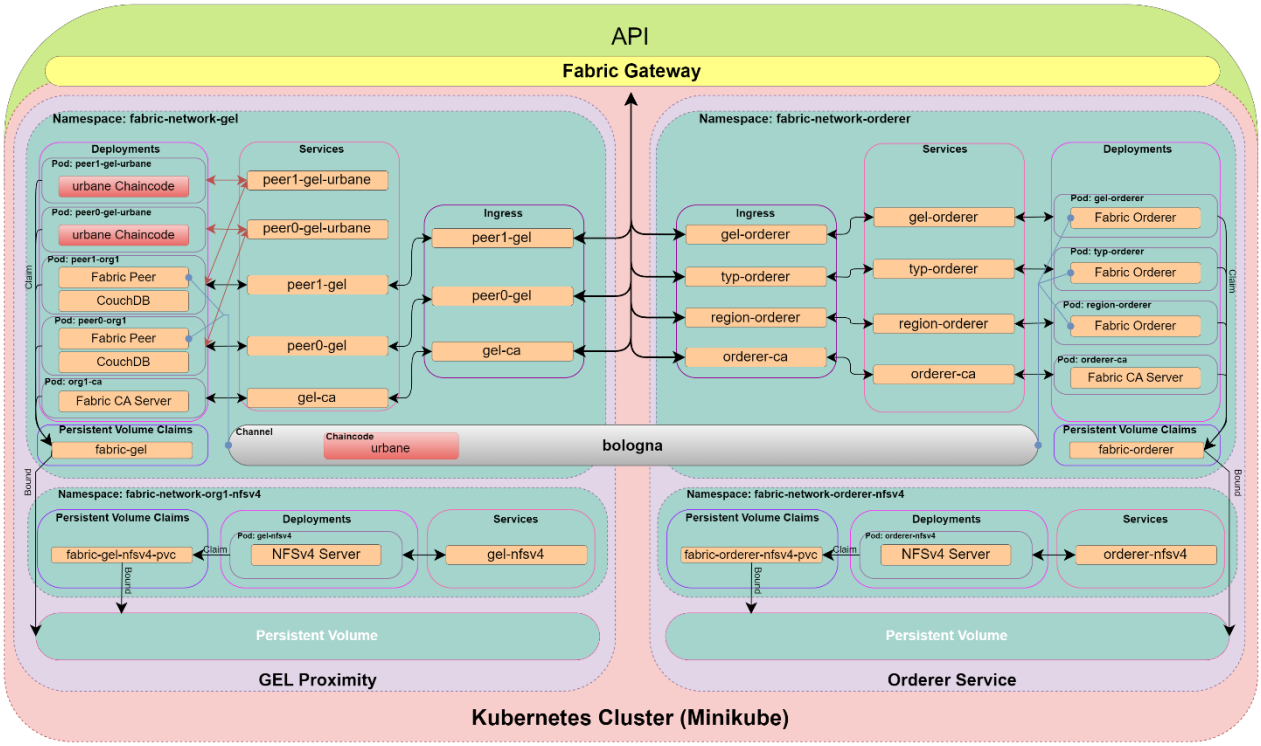


FIGURE 4: URBANE BLOCKCHAIN COMPONENT ARCHITECTURE

2.3.2 Operation

The repository of the framework consists of multiple files, while the following four scripts are considered as the main tools to operate and control the blockchain network:

- **minikube.sh**: This script is responsible for spawning or destroying a Minikube instance as well as checking the status of the Minikube instance.
- **nfsv4.sh**: This script is responsible for deploying a new NFSv4 Server as well as its required components to Kubernetes.
- **networkStart.sh**: This script is responsible for numerous operations such as:
  - Deploying organizations to Kubernetes
  - Creating channel artifacts
  - Creating a channel
  - Joining organization Peers to the channel
  - Deploying chaincode to Kubernetes
  - Installing chaincode to Peers
  - Approving chaincode for organization
  - Committing chaincode to the channel
- **networkDown.sh**: This script is responsible for tearing the HLF blockchain network down.

3 Security - Digital Identification for Platform Services

One of the most important tasks within the URBANE project is the development of a distributed, blockchain-based and privacy-preserving identity and access management scheme to secure the exchange of information between the URBANE platform and external actors. Identity and Access

Management (IAM) Frameworks are designed to provide the necessary technology for ensuring that only authorized individuals/users can access the resources of a specific application. This is achieved by implementing appropriate policies that contain information to authenticate a user, identify the resources they are authorized to access, and specify the actions they can perform with those resources.

3.1 DIDs - Building security into the platform

In this section, the URBANE decentralized IAM framework is presented which is used to provide secure access to the resources of the URBANE platform. The framework is based on the latest emerging standards on decentralized Identities, namely Decentralized Identifiers (DIDs)[6] and Verifiable Credentials (VCs)[7] and follows a Self-Sovereign Identity (SSI) approach to provide full control of the identity to the users and organizations of the URBANE communities. At the same time, it supports the current best practices in access management protocols i.e., OAuth 2.0 protocol which is the go-to solution for API security and authorisation, to enable easy integration with existing services in the LLs. To this end, two different worlds are brought together to guarantee security, privacy, and flexibility. The decentralised IAM framework employs the same blockchain infrastructure deployed in T3.3 for the sharing of logistic assets and uses smart contracts to store and verify identities in a decentralised manner, as described in section 2.

3.2 DID assignment, use, and management

The IAM scheme in URBANE follows the OAuth 2.0 flow[8] and includes three main actors, the Resource server, namely the URBANE Platform, the Authorisation server, and the client namely the external backend services of the LLs. The difference with the standard OAuth 2.0 flow is that all the communications and verification processes are timestamped and rooted on the blockchain for increased transparency and decentralisation.

As shown in Figure 5, at the beginning of the process, an external LL service requests its registration from the Authorisation server (or Issuer in the SSI world), which enrols the service by creating a DID in the Blockchain and by then issuing an Access Token to the service, in the form of a VC. The communication of the external service (or device) is handled by a specific identity client (wallet), developed in the context of URBANE that interacts with the platform using the Oauth2.0 protocol, with the only difference being that it uses DIDs and VCs instead of Basic Authentication (username, password). Finally, the URBANE platform API verifies the validity of the Access Token through the data stored in the blockchain (DIDs), without the need for interacting with the Authorisation server as SSI suggests.

[6] Decentralized Identifiers (DIDs) v1.0. https://www.w3.org/TR/did-core/
[7] Verifiable Credentials v2.0, https://www.w3.org/TR/vc-data-model-2.0/
[8] OAuth 2.0 flow, https://autho.com/docs/authenticate/protocols/oauth

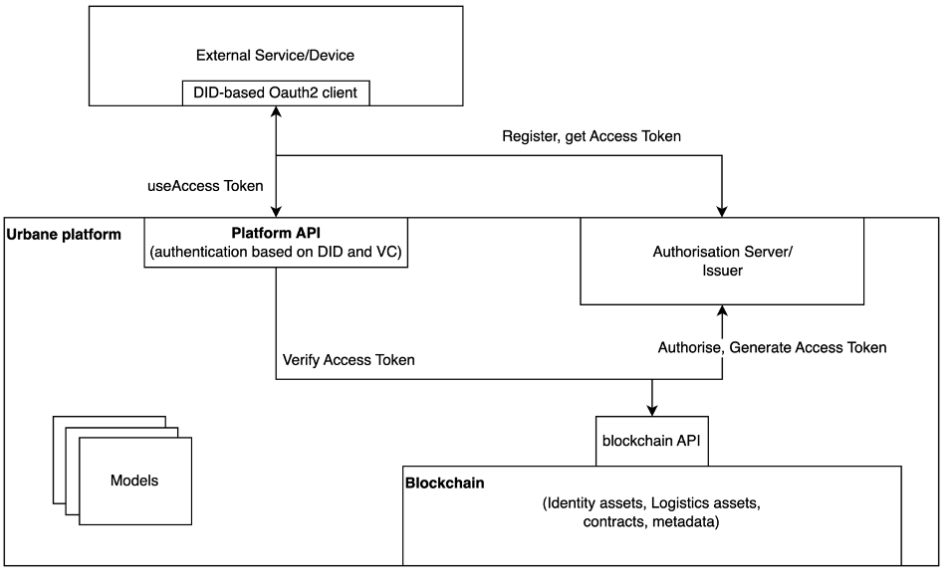


Figure 5: DID-based authentication in URBANE

Figure 6 presents the sequence diagram of the decentralised IAM component of the URBANE platform, which includes the registration phase and the authentication and authorisation phases of a client requesting (or sending) resources from (or to) the URBANE platform. At the registration phase, all actors create a DID in the blockchain infrastructure, using the dedicated smart contract, as described in section 4. Then, the client submits a request to the /oauth2/authentication endpoint of the authorisation server to initiate a challenge response process for its authentication. Consequently, the client submits a request to the /oauth2/token endpoint of the authorisation server to get an Access Token which will be used in each of its interaction with the URBANE Platform API. Finally, The URBANE Platform API, as the verifier roles suggest in SSI, employs the data stored in the blockchain (DIDs) to validate the provided Access Token.

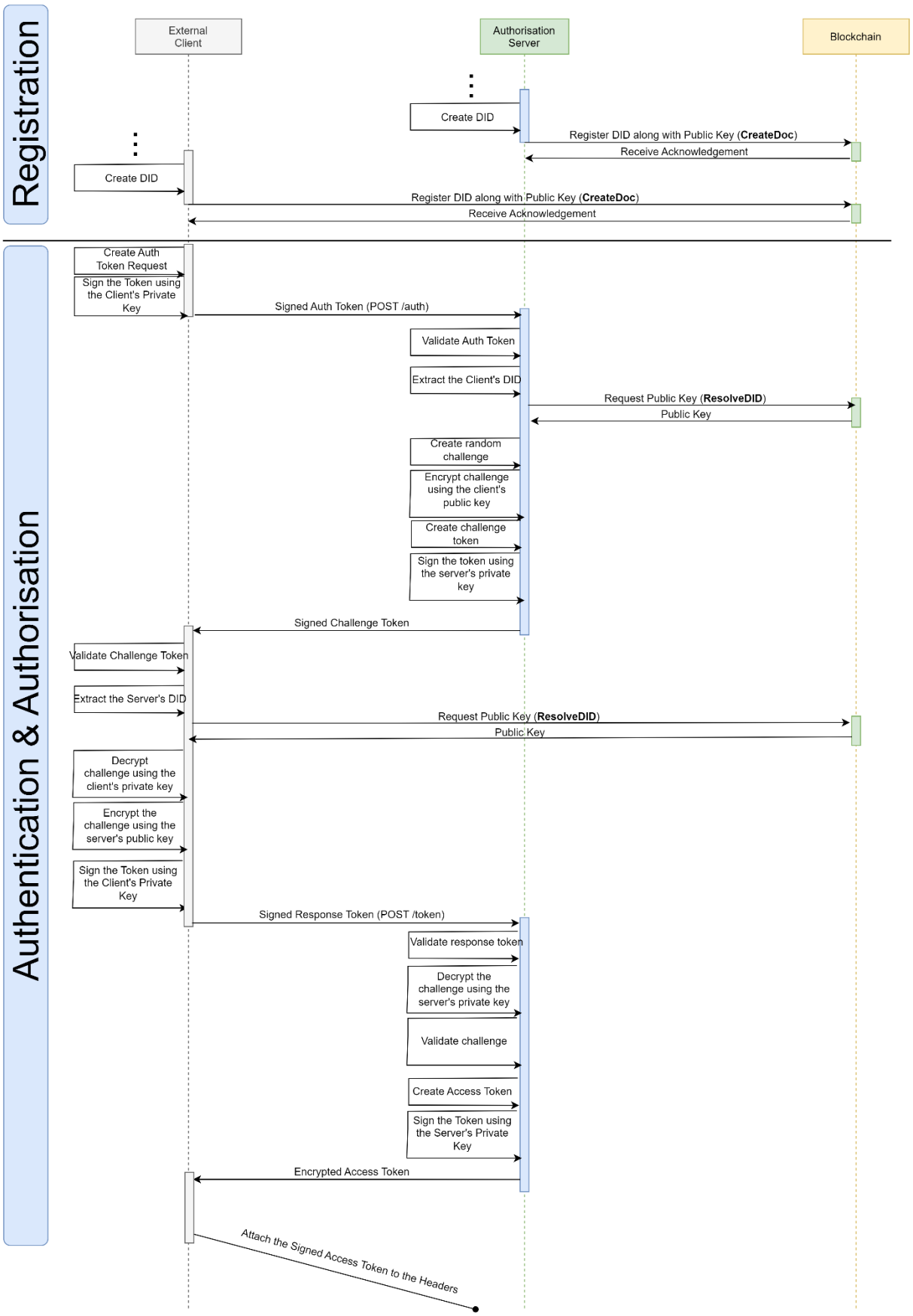


FIGURE 6: SEQUENCE DIAGRAM OF THE DECENTRALIZED IAM FRAMEWORK IN URBANE

Table 4 lists the endpoints of the URBANE IAM component, together with useful details for the end-users and the administrators of the URBANE Platform. The /vdr/ group of endpoints is exposed by the Verifiable Data Registry (VDR) Smart Contract which handles CRUD (Create, Read, Update, Delete) operations for DIDs. It uses the gRPC protocol for enhanced performance and scalability in the registration of new clients. On the other hand, the OAuth 2.0 relevant endpoints, namely the /oauth2/ group, are exposed by the Authorisation Server to handle requests for Authentication and Access Tokens.

TABLE 3: DID ENDPOINTS IN URBANE PLATFORM

Endpoint	Description	Input	Output	Protocol	Component
/did/create	Creates a DID Document and stores it in the blockchain	DID Document	DID Document	gRPC	Blockchain API
/did/resolve	Retrieves a DID Document from the blockchain	DID	DID Document	gRPC	Blockchain API
/oauth2/authenticate	Validate Authentication token	Authentication Token	Signed Challenge	gRPC or REST	Authorisation Server
/oauth2/token	Validate Response token	Response Token	Access Token	gRPC or REST	Authorisation Server

4 Trust – Smart Contracts

In the URBANE project, the logistics transportation processes engage multiple carriers, necessitating the establishment of a mechanism to validate whether actors adhere to their service commitments as stipulated in agreements. This mechanism also empowers actors to substantiate that errors did not originate from their actions, affirming their compliance with regulations. However, it is essential to avoid the establishment of a centralized system wherein data sovereignty is concentrated in the hands of a single actor. Such a setup would require the involvement of a neutral entity managing the information flow and necessitate all participating actors to relinquish data sovereignty to this entity. Implementing such a practice is challenging and introduces an imbalance among the involved actors.

The integration of Smart Contracts within the HLF framework emerges as a crucial enabler for monitoring these complex transportation and delivery monitoring processes. Smart Contracts offer a sophisticated solution to elevate transparency, efficiency, and accountability across the entire freight transportation lifecycle. Their capacity to autonomously establish and enforce agreements creates a tamper-resistant and transparent transaction record.

Of particular significance is the role of Smart Contracts in ensuring non-repudiation, where parties cannot deny their involvement or dispute the authenticity of recorded transactions. This feature fosters a high level of trust among stakeholders. Without Smart Contracts, the blockchain's capability to monitor and

securely store vital shipment details and events would be severely constrained. Smart Contracts, within the HLF ecosystem, play a pivotal role in automating and fortifying these processes. They facilitate a comprehensive and dependable tracking mechanism for shipments, events, and agreements. In essence, the utilization of Smart Contracts in HLF becomes indispensable for establishing a robust and trustworthy system adept at addressing the challenges inherent in monitoring transportation processes with diverse stakeholders.

4.1 Smart Contract Architecture and Generator

In URBANE and the last mile delivery ecosystem, there are numerous use cases and changing variables that make it impossible to create a rigid Smart Contract for all use cases. Therefore, one of the main requirements of the Smart Contracts is to be able to dynamically adapt to variables (e.g., delivery windows, volumes, carriers, etc.). For this reason, a Smart Contract Generator is integrated into the URBANE platform that automatically generates Smart Contracts based on incoming events, installs, and instantiates them on the blockchain. First, in starting the Smart Contract generator, user applications must feed it with the configuration parameters (events to monitor, what is a starting trigger, what is an ending trigger, any outbound addresses to send messages to, what happens when a delivery failure arises, how to handle reschedules, etc.). These parameters may be different in the different city contexts and associated use cases and must therefore be provided once at the beginning by the user applications to set up the automatic creation of Smart Contracts. The following diagram (Figure 7) illustrates in concise form how the automatic creation of Smart Contracts is organised:

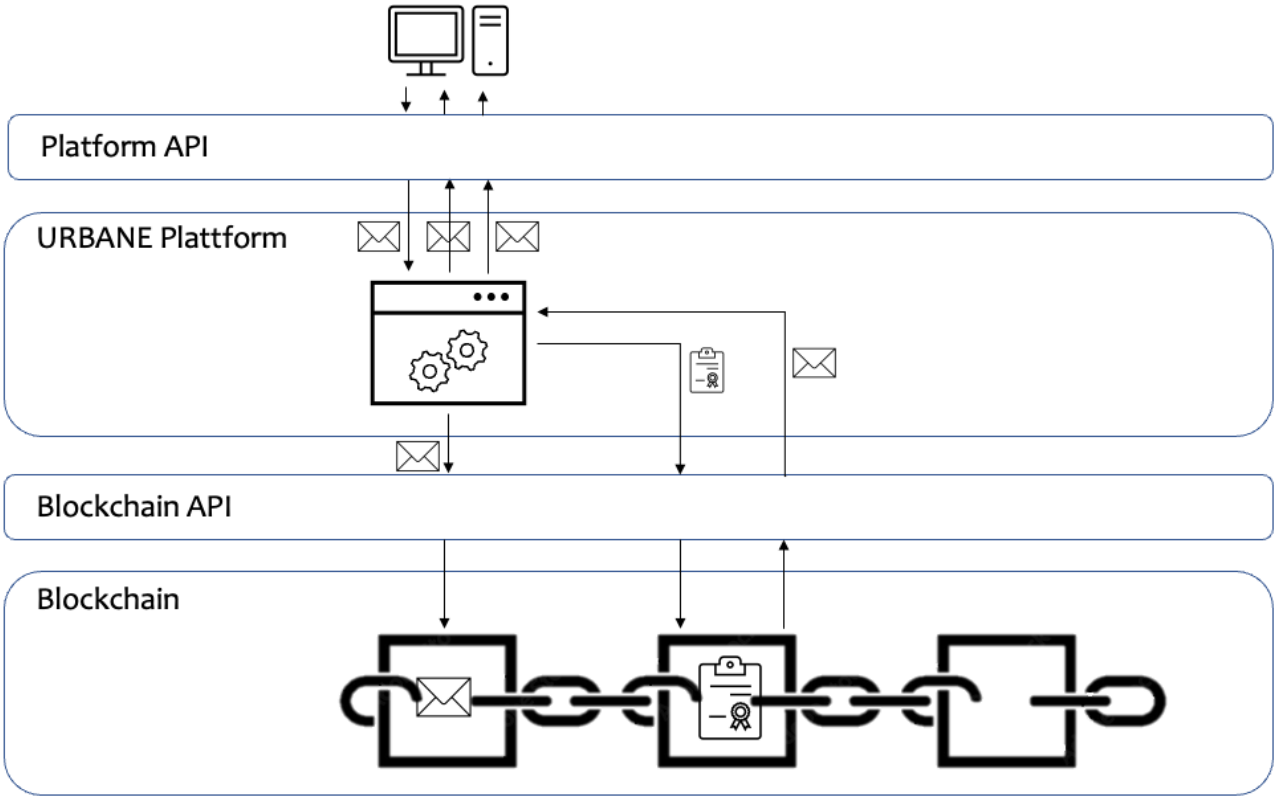


FIGURE 7: SMART CONTRACT GENERATION

4.1.1 Setting up the Smart Contract - UI, monitored items, contract generation

The Smart Contract Generator facilitates urban logistics communities in setting up agreements, configuring the events sources and monitoring the adherence to the terms of the contract, hiding the infrastructure complexities from the application end users. By tracking deliveries in real-time through smart contracts, last mile communities streamline their operations and reduce manual interventions in key aspects of the delivery process, such as order confirmation, route planning, and proof of delivery. This leads to cost reduction, reduced administrative overhead and minimal paperwork, but also real-time tracking and increased visibility throughout the entire last mile chain. It also increases the accountability urban logistics providers by reducing risks of errors and delays.

The Smart Contract Generator comprises multiple modules for the input of data, the processing of information and the actual deployment of the smart contract in the appropriate blockchain namespace. Regarding the input of data from a user of the Urbane Platform, the Blockchain Dashboard exposes a simple UI that enables the end-user to fill in all the contract details, as shown in Figure 8. The UI acts as the gateway for users to interact with the Smart Contract Generator. Its intuitive design empowers users to effortlessly specify contract details, choose templates, and initiate the creation process. The UI serves as the bridge between the intricacies of blockchain architecture and the simplicity desired by end-users.

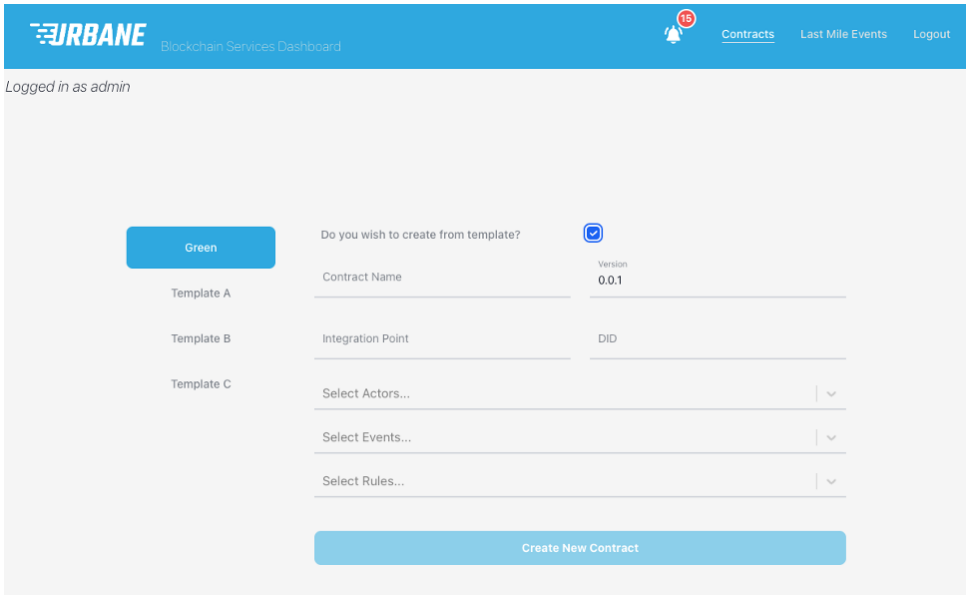


FIGURE 8: SMART CONTRACT GENERATOR UI IN THE BLOCKCHAIN DASHBOARD

The detailed description of each one of the fields of the Smart Contract Generator UI can be found in Table 4.

TABLE 4: DESCRIPTION OF THE SMART CONTRACT GENERATOR UI FIELDS

Field	Description	Value
Contract Name	The name of the contract. This is also used as its unique ID	Filled in by the user

Integration Point	The URL of the backend system, from which the Platform API will receive data	Filled in by the user
DID	The DID that the backend system will use to send data to the Urbane Platform API	Filled in by the user
Actors	The actors involved in the contract. The Smart Contract Generator will spin up a separate Organisation in the blockchain network for each one of the actors	Filled in by the user
Events	The events to be monitored by the smart contract	One or more of the URBANE events as listed in this deliverable
Rules	The rules to be monitored by the smart contract. The Smart Contract Generator will enable the dedicated function for each of the rules	One or more of the rules: Missing events Damaged Delayed
Green	This will enable the monitoring of environmental metrics by the smart contract throughout a shipment's lifecycle	N/A

The submission of a new contract request in the Blockchain Dashboard will trigger the Smart Contract Generator backend (Figure 9) to process the request and issue a createContract transaction in the blockchain ensuring transparency and traceability. The Blockchain records the vital details of every contract born from the Smart Contract Generator. This archival system preserves information such as contract addresses, template references, and timestamps — a testament to the commitment to security and auditability. This will enable the monitoring of shipments by receiving logistic events from LLs through the Platform's API.

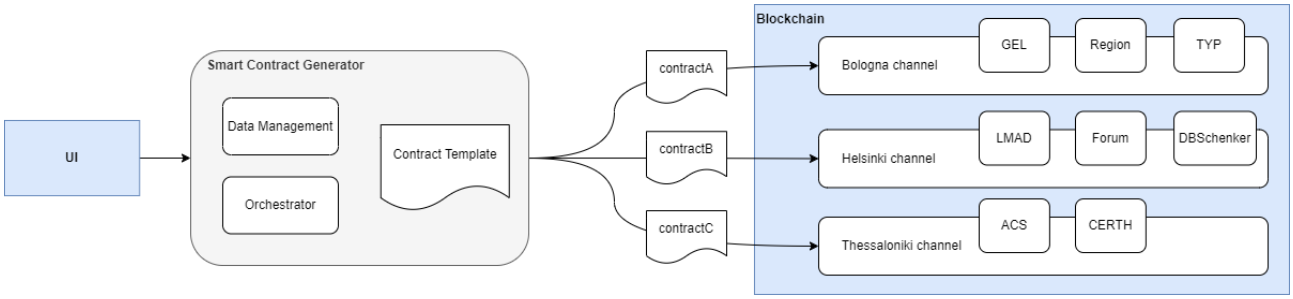


FIGURE 9: HIGH LEVEL VIEW OF SMART CONTRACT GENERATOR OPERATION

The Smart Contract generator checks whether all parameters required for the Smart Contract have been sent via the platform API. In case parameters are missing, an error message is sent to request the required information. If a compliant set of information was passed an ack is sent, the generator creates a Smart Contract, passes the monitoring parameters and addresses/events to listen for to the Smart Contract, and posts the Smart Contract on the blockchain. The Smart Contract now operates as programmed.

4.1.2 Monitoring the Smart Contract – UI, event notifications

Monitoring the delivery process using the Smart Contracts setup via the URBANE platform is performed through the Blockchain Dashboard. A simple user interface (UI) is available to authorized users that displays the various events being tracked through the Smart Contract in the delivery of an order (Figure 10). The event monitoring dashboard provides the user with views on each order, whether the events to monitor have been completed, if events were skipped, and whether the order was delivered per service agreement. Additional information, such as coordinates of delivery, authorized signatures, etc. can also be transmitted based on the Smart Contract setup.

While the user can access all Smart Contract information via the Blockchain Dashboard, all Smart Contract information can be electronically sent to an authorized user's internal monitoring system via the bidirectional nature of the URBANE blockchain API. This capability allows users to see the progress of their order deliveries in their systems of record without having to use the Blockchain Dashboard.

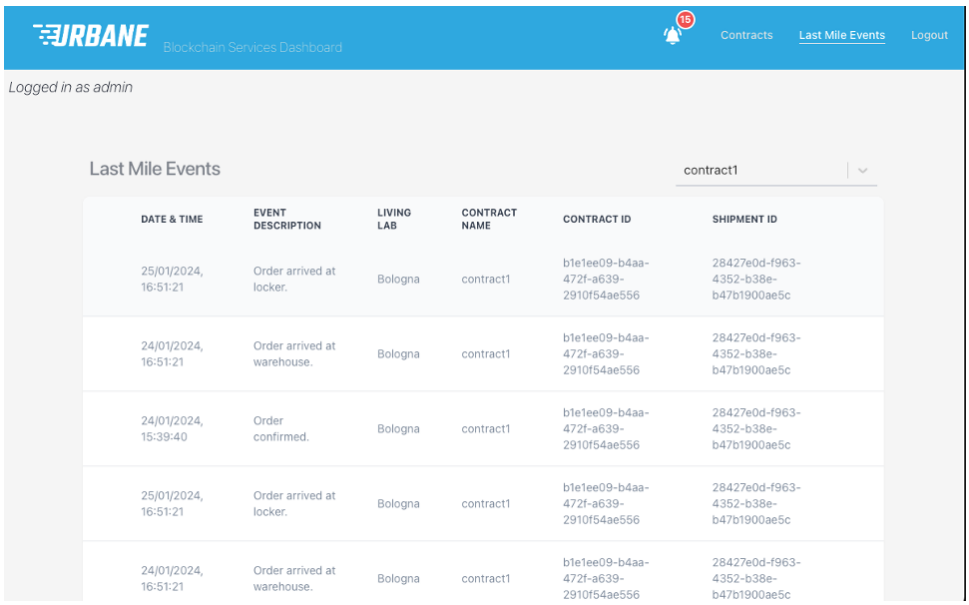


FIGURE 10: LAST MILE EVENTS PAGE OF THE BLOCKCHAIN DASHBOARD

4.2 Smart Contract Structure

In close collaboration with the LLs, we conducted a comprehensive analysis of the entire transportation process. Through this analysis, we pinpointed critical junctures where the transfer of parcels occurs, either among the involved service providers or towards the end consumer. Subsequently, we developed a unified terminology across the Living Labs and ultimately derived the associated events, as outlined in Figure 11.

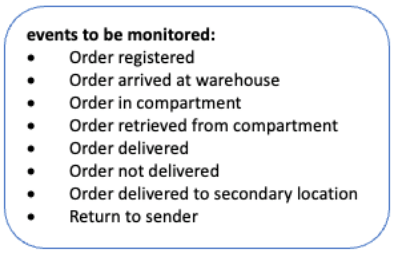


FIGURE 11: EVENTS OF THE TRANSPORTATION PROCESS

In a second step, we collectively identified the parameters essential for storage concerning each shipment and event on the blockchain. A shipment refers to a parcel subject to monitoring through the platform and smart contract. This encompasses fundamental details about the parcel, including dimensions, weight, and the designated delivery time. As denoted above, Events, on the other hand, denote occurrences throughout the transportation process, commencing from order registration, parcel insertion into a compartment, to the ultimate delivery, whether successful or unsuccessful. Information recorded at each stage of transportation encompasses IDs, descriptions, timestamps, and, for orders delivered, additional details such as coordinates, signature, or recipient name. To enhance clarity, the various structs and events slated for monitoring are presented in the figure below.

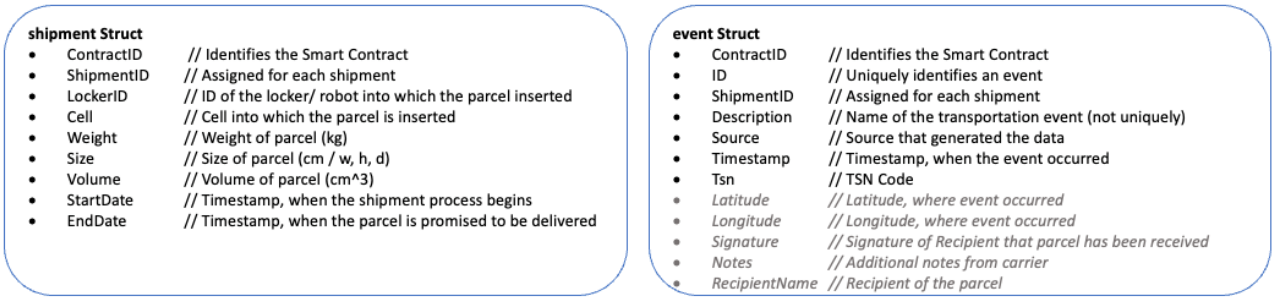


Figure 12 : Shipment & Event struct/events to be Monitored

4.2.1 Functions

Each function within the contract has been developed as a standalone entity within the Smart Contract. This design allows LLs to select only those aspects of a shipment that they wish to monitor to be enabled when they set up a Smart Contract. This ensures that the LLs can utilize the Smart Contract Generator to tailor the smart contract's capabilities, incorporating only the functionalities they specifically require.

The initial function of the smart contract enables the secure storage of shipments on the ledger, safeguarding them against unauthorized alterations. This includes recording essential details such as the start and end times of the transportation process, along with metrics like volume and weight. The sequential steps, illustrated in Figure 13, are as follows:

- Retrieve all parameters corresponding to the Shipment struct.
- Create a unique shipmentKey using the provided shipmentId and checks the ledger for its existence.
- If the shipmentKey already exists, the function returns an error message indicating that the shipment with the specified key already exists on the ledger.
- If the shipmentKey does not exist, the function proceeds to create a new shipment with the provided parameters.
- The finalized shipment information, including the event details, is securely stored on the ledger under the previously generated shipmentKey.

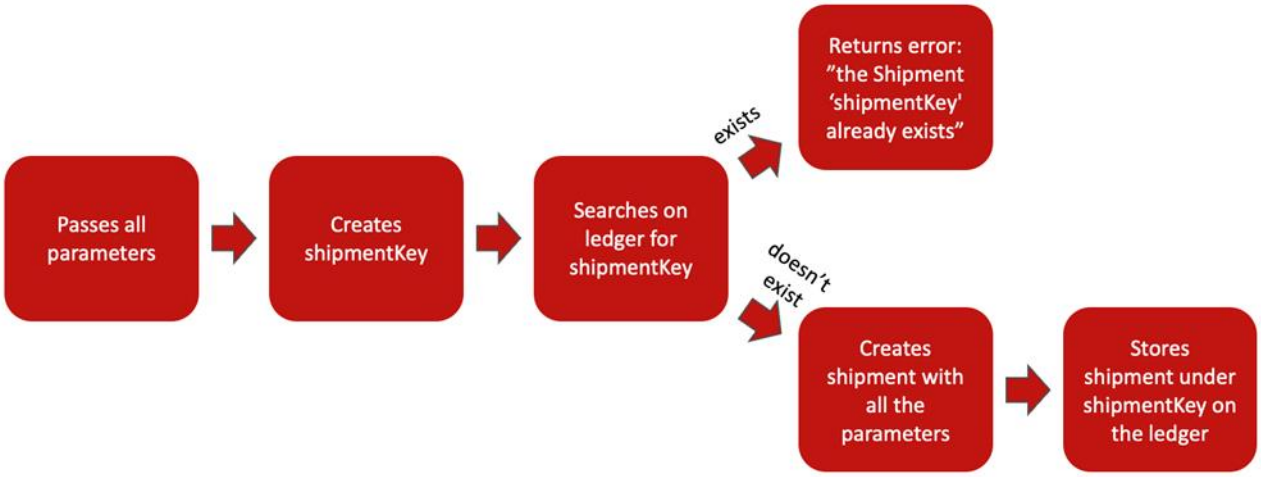


FIGURE 13: CREATE SHIPMENT FUNCTION

The second function of the smart contract ensures the secure storage of events on the ledger, establishing an immutable record that serves as a reliable reference in case of disputes among collaborating entities. The sequential steps, illustrated in Figure 14, are as follows:

- Retrieve all parameters corresponding to the Event struct.
- Create a unique eventKey using the provided shipmentId and checks the ledger for its existence.
- If the eventKey is already present on the ledger, the function retrieves the data and compares the 'ID' of the events.
  - If the 'ID' is the same, it returns an error message indicating that the event with the specified description and eventKey already exists.
- If the 'ID' is different it checks if the new event was prior to the latest event stored.
  - If the event was prior to the latest event, the new event is not stored on the ledger and the function terminates.
- If the 'ID' is not the same or the eventKey was not found on the ledger, a new event is created.
  - If the 'description' is "Order delivered" the additional required parameters are added.
- The newly created event data, including the description and event details, is securely stored on the ledger under the previously generated eventKey.

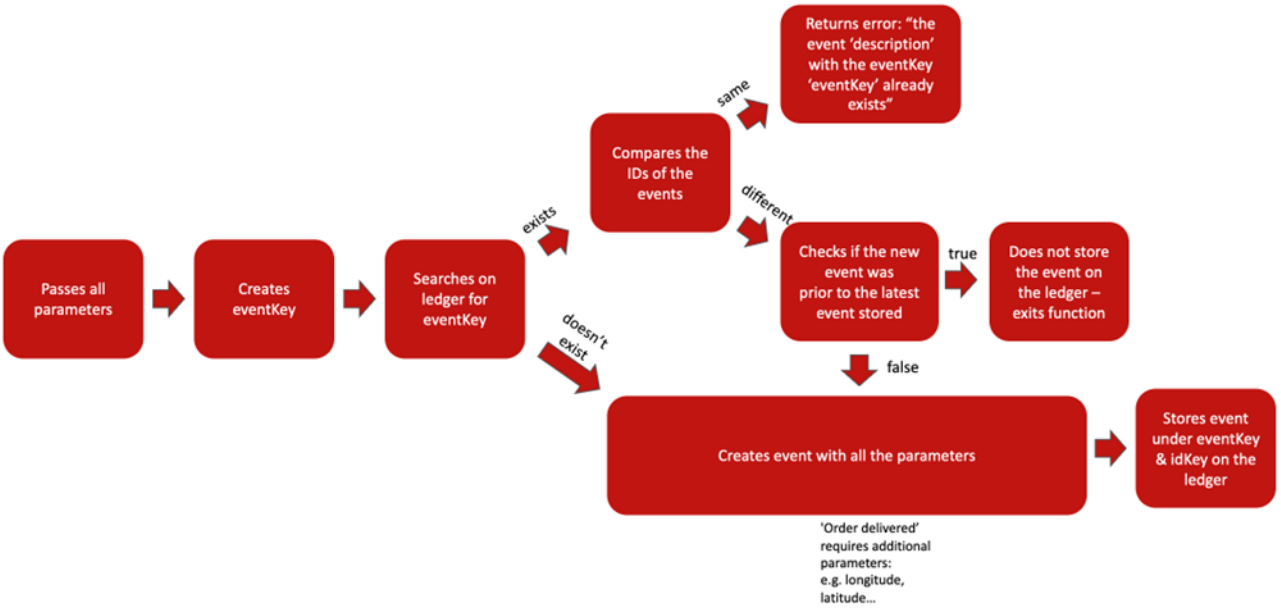


FIGURE 14: CREATE EVENT FUNCTION

In the CompareShipmentEndDate function, the primary objective is to assess whether the delivery is punctual, thereby facilitating quality monitoring. The sequential steps, illustrated in Figure 15, are as follows:

- Retrieve all parameters corresponding to the Event struct.
- Firstly, the function examines whether the description is "Order delivered."
  - If the condition is false, the function is terminated.
- If the condition is true, the function proceeds to verify the existence of the corresponding shipment by creating a shipmentKey.
- The ledger is then searched for the presence of the shipmentKey.
- If the shipmentKey is absent from the ledger, an error message is generated, indicating the inability to execute the subsequent steps of the function.
- If the shipmentKey is present, a deliveryKey is generated.
- A check is made to determine if the deliveryKey already exists on the ledger.
- If the deliveryKey is found on the ledger, a message is sent: "Data for 'ShipmentID' has already been compared with 'ID'. Stored Result: 'Result'".
- If the deliveryKey is not stored on the ledger, the function retrieves information from the previously located shipmentKey to obtain the EndDate.
- Subsequently, the function compares whether the EndDate is earlier or later than the timestamp of the event intended for storage.
- The resulting evaluation is then stored on the ledger.
- Finally, the outcome is conveyed to the platform for further processing.

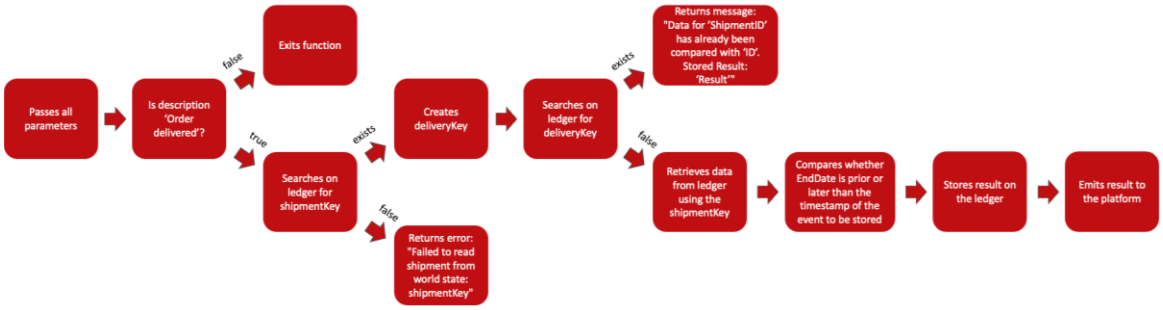


FIGURE 15: COMPARE SHIPMENT ENDDATE FUNCTION

The CompareDescription function is designed to validate whether the transportation events unfold in the correct order. The sequential steps, illustrated in Figure 16, are as follows:

- Retrieve all parameters corresponding to the Event struct.
- Initially, the function checks if the description is "Order registered."
  - If this condition is met, the result is directly posted under the orderKey on the ledger, including all relevant parameters.
  - An event is emitted to the platform.
  - Subsequently, the function concludes.
- If the description is not "Order registered," the function proceeds to examine the ledger data.
- Search for the eventKey on the ledger and retrieves the stored information.
- Search for the orderKey on the ledger and retrieves the stored information.
- Check if the existing entry has the same "ID."
  - If a match is found, the function returns a message: "Data for 'ShipmentID' has already been compared for 'ID'. Stored Result: 'Result'".
- In cases where there is no identical ID, the function assigns a numeric value to the description using description mapping.
- Compares whether the new event occurred before the latest event stored, determining whether an event in the transportation process is missing or if the events are in the correct order.
- The result, along with ShipmentID, ID, and Timestamp, is stored under the orderKey on the ledger.
- Lastly, an event is emitted to the platform for further processing.

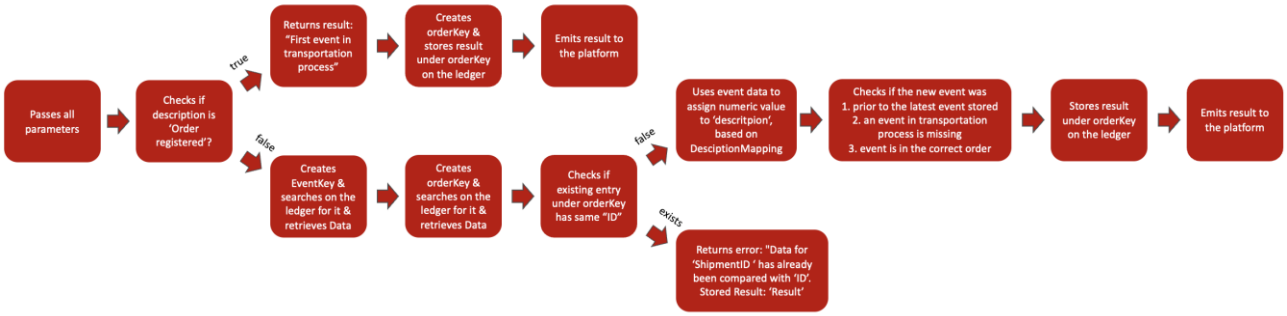


FIGURE 16: CHECK IF SEQUENCE OF TRANSPORTATION STEPS IS CORRECT FUNCTION

The GetEventByID function is designed to retrieve the information from the ledger based on the unique ID. The sequential steps, illustrated in Figure 17, are as follows:

- Retrieve the unique ID.
- Initially, the function creates an idKey and searches on the ledger for it
- If it does not exist on the ledger the error message "event with ID, 'ID' is not found.

- If the idKey is found, the data stored under the idKey is retrieved.
- In case the description is not 'Order delivered' default values for longitude, latitude, signature, notes, and recipient name are assigned.
- Lastly, an event with all the information is sent to the platform for further processing.

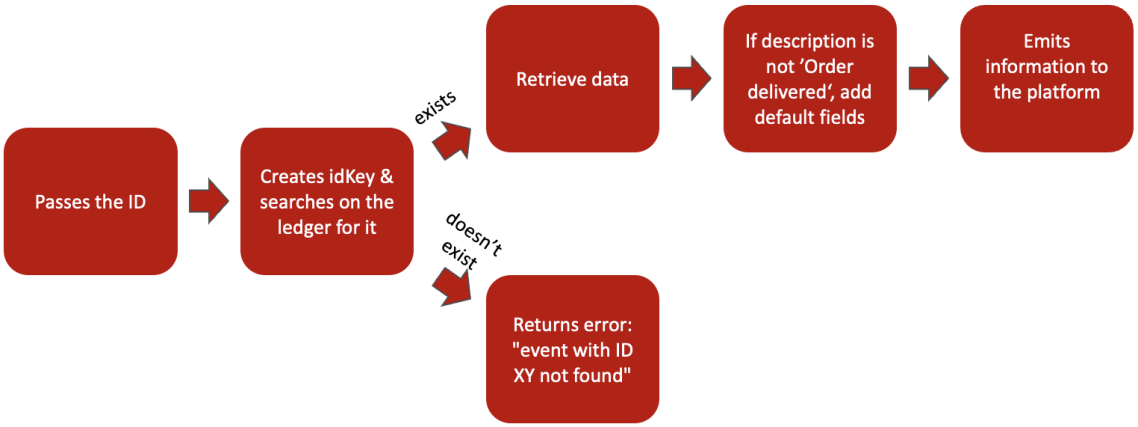


FIGURE 17: RETRIEVE INFORMATION FROM LEDGER

5 Representative Use Cases

This section of the report covers the three Lighthouse Living Labs (Bologna, Helsinki, Thessaloniki) that will be employing the entire blockchain infrastructure (access authorization and smart contracts). Valladolid, the fourth Lighthouse Living Lab, due to the innovation pilot that is being tested in their project, has decided that using the smart contract component of the blockchain infrastructure would not add value to their project at this time. Depending on how their project evolves, they have indicated that the full blockchain infrastructure may be deployed.

It should be noted that while the blockchain infrastructure has been fully tested using historical data files from the three Living Labs described below, none of the Living Labs have "gone live" with their innovation use cases. This means that the blockchain infrastructure will need to be tested with the operational deployments of each Living Lab's innovation project once they commence operations. The blockchain infrastructure will be operated in parallel with the partner operational systems at this time to ensure that, should any issues arise with the infrastructure, it does not compromise the commercial aspects of these projects.

Future enhancements of the blockchain infrastructure are planned as Wave 2 cities are added to the portfolio of URBANE platform users. The requirements for these cities will be developed as cities establish the formal outlines and implementation plans for their pilot operations.

5.1 Bologna

The Bologna use case focuses on tracking shipments into and out of the locker system that forms the foundation of the Bologna Living Lab demonstration. The Bologna demonstration involves using several lockers located in the Bologna city center as micro-hub distribution centers for parcels that are to be picked up and delivered to businesses or residents in the city center. LSPs deliver parcels to these micro-

hubs and arrange for pickup and delivery of the parcels by authorized sustainable last mile delivery actors (Figure 18).

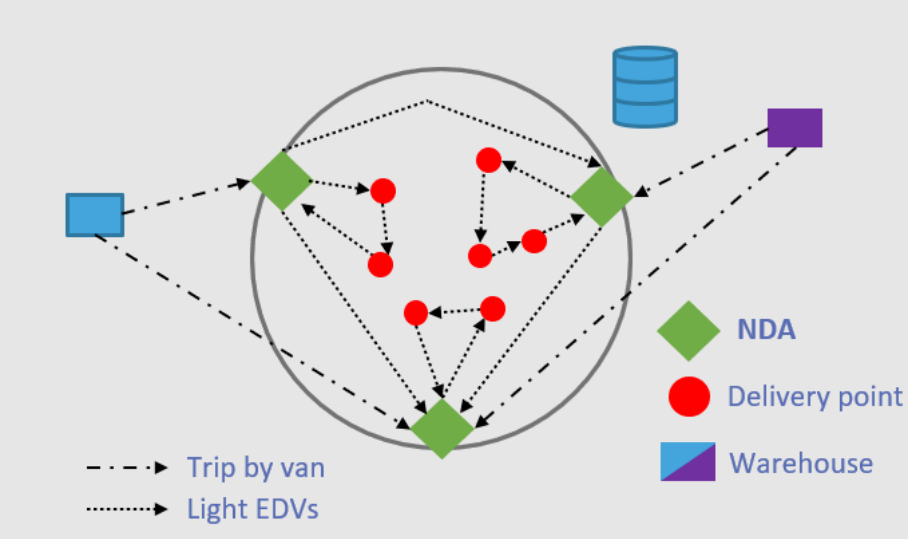


FIGURE 18: SCHEMATIC OF BOLOGNA LL DEMONSTRATION
NOTE: NDA REFERS TO A PARCEL LOCKER

To manage the information flows between the various actors involved in the delivery process, Bologna has contracted a third-party logistics platform, GEL Proximity. GEL Proximity operates as an information exchange and scheduling hub for the flow of information between the LSPs, micro-hubs, and last mile delivery entities. As multiple LSPs deliver items to the micro-hubs and multiple last mile delivery operators pick up parcels from the micro-hubs and deliver them all orchestrated through the communications and scheduling platform of GEL Proximity, it is critical that the entire process be monitored in a secure manner to ensure that all parties adhere to the terms of their contracts and that orders are delivered in a timely manner.

To build a trustworthy system for the sustainable delivery of items in its city center and create trust amongst the many service providers involved in the logistics process, Bologna will employ a smart contract process that includes the monitoring of the following events:

- Order registered (with the LSP)
- Order arrived at LSP's warehouse
- Order placed in compartment
- Order retrieved from compartment (by last mile service provider)
- Order delivered (along with delivery information)
- Order not delivered
- Order delivered to secondary location (if order cannot be delivered)

The smart contract for this process has been tested using test data from the GEL Proximity system and shown to properly log the noted events on the blockchain. It has also been demonstrated to properly populate the URBANE platform's UI when events have been logged or contractual requirements have not been met. The operational testing of the smart contract will be conducted in the spring of 2024 when the city of Bologna and its Living Lab partners begin operational testing of the demonstration project.

5.2 Helsinki

The Helsinki pilot project focuses on testing the concept of using a micro-hub in conjunction with direct delivery of orders using an automated delivery vehicle (ADV). Several phases of this project will be tested using different last mile delivery approaches and more involved micro-hub operations. The first phase is the focus of the initial URBANE smart contract deployment. In this initial pilot customers will select how they wish their orders to be delivered when they place their order. The LSP will then either arrange for a direct delivery from the micro-hub or, if a more environmentally friendly delivery is desired, assign the delivery to the ADV and provide the customer with an access code to pick up their order from the ADV. In either case, the need for monitoring the delivery process will be critical as the order will be out of the LSP's direct control once it has been delivered to the micro-hub. The process envisioned for this pilot operation is shown in Figure 19 below.

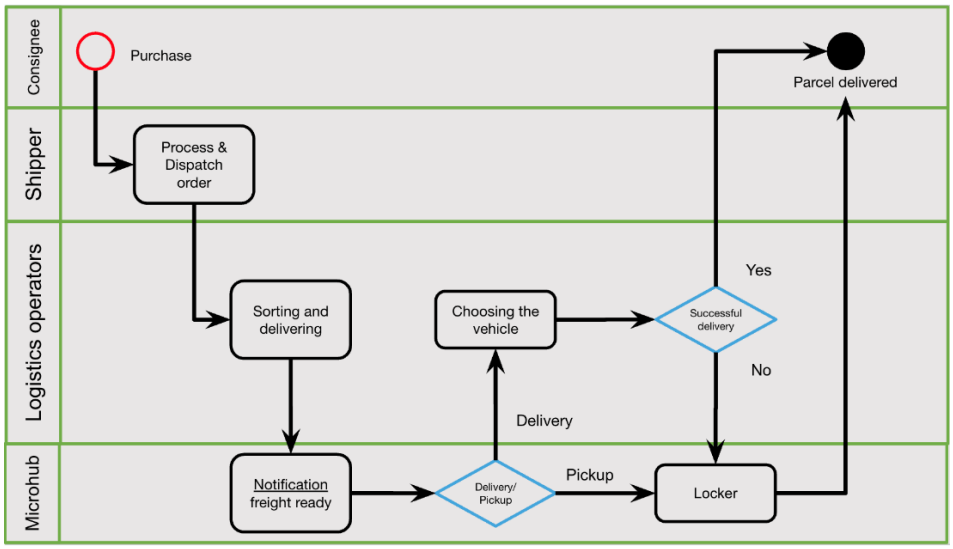


FIGURE 19: HELSINKI PILOT DELIVERY PROCESS FLOW

To allow the LSP to monitor the delivery of orders that they have entrusted to the pilot, the URBANE Hyperledger and smart contracting infrastructure will be employed. The events that will be monitored are:

- Order registered (customer books delivery with AGV operator for AGV delivery)
- Order in compartment (inserted into one of the compartments on the AGV)
- Order delivered (along with delivery confirmation information along with appropriate location and customer information)
- Order not delivered (sent once the undelivered item has been returned to the micro-hub)
- Order delivered to secondary location (the order has been picked up at a second location on the AdV's route along with appropriate location and customer information or order is returned to micro-hub for rescheduled delivery – to be determined)

The smart contract setup and tracking process has been tested using test data provided by the Helsinki Logistics service company. Operational testing using "production" data will be conducted in spring 2024 when the city of Helsinki and the Living Lab partners begin operational testing of pilot project.

5.3 Thessaloniki

The Thessaloniki use case focuses on tracking shipments into and out of the locker system that forms the foundation of the Thessaloniki Living Lab demonstration. The process being followed is like that being explored in the Bologna Living Lab. However, the Thessaloniki demonstration involves the examination by a major logistic services provider of the benefits of using a hub and spoke delivery model integrated with a city-based micro-hub network for last mile deliveries. Specifically, micro-fulfilment centers are installed at the perimeter of the historical center of the city and are to be tested in an operational environment to achieve higher load factors and lower vehicles usage, enhancing the effectiveness of the operational planning process and the customer experience. The process that is to be followed appears in the flow chart below (Figure 20).

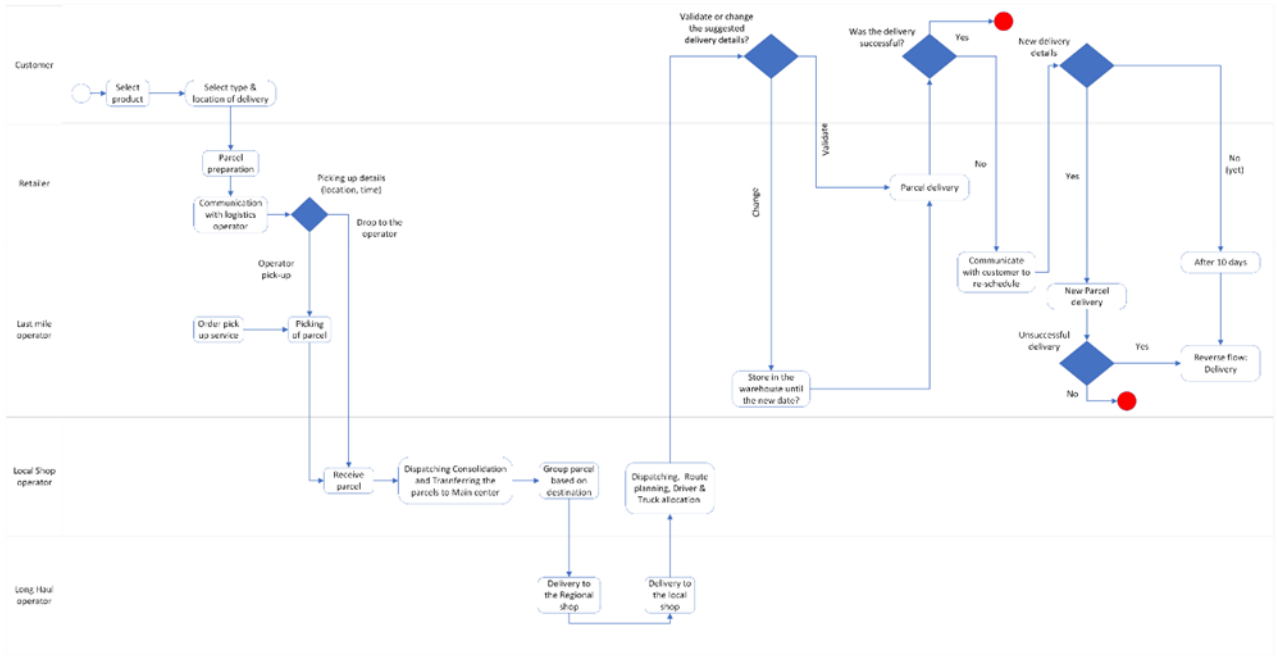


FIGURE 20: THESSALONIKI LL DELIVERY PROCESS

Thessaloniki will employ a smart contract process that includes the monitoring of the following events:

- Order registered (with the LSP)
- Order arrived at LSP's warehouse
- Order placed in compartment
- Order retrieved from compartment (by last mile service provider)
- Order delivered (along with delivery information)
- Order not delivered
- Order damaged
- Order delivered to secondary location (if order cannot be delivered)

The smart contract for this process has been tested using test data from the ACS system and shown to properly log the noted events on the blockchain (ACS is the LSP managing and testing the hub and spoke and locker system). It has also been demonstrated to properly populate the URBANE platform's UI when events have been logged or contractual requirements have not been met. The operational testing of the smart contract will be conducted in the spring of 2024 when the city of Thessaloniki and its Living Lab partners begin operational testing of the demonstration project.

6 Conclusions and next steps

The integration of commercial, governmental, and societal players into a workable urban logistics model that addresses the triple bottom line focus of all players (people, profit, planet) is a difficult collective action problem to resolve. The key to its resolution is trust. For trust to occur the system employed must be trustworthy. This implies that such a system must address the issues of security, privacy, equity, transparency, and usability in an open and fair manner. The blockchain infrastructure of the URBANE platform described in this document provides one such approach to building a trustworthy system for last mile logistics operations in an urban environment. By providing security using state-of-the-art digital identification processes, an immutable ledger, and service level monitoring through smart contracts, the URBANE blockchain infrastructure ensures users that they are using a trustworthy system. This is a necessary, but not sufficient, condition for creating trust.

To create an urban logistics environment in which all players trust one another requires going beyond building a trustworthy system. It involves active discussions amongst all players as to how their interaction affects the urban environment. It also requires governments to develop rules and regulations that harmonize with the intentions of citizens and that are commercially acceptable to the private companies that must perform the logistics services. The problem is a difficult collective action problem and one that the URBANE project hopes to contribute to solving through the development of the blockchain and smart contract infrastructure of the Innovation Transferability Platform. This platform implements the learning surfaced in D1.1 of the project building on the framework requirements for the Physical Internet. It provides a foundation for the LLs participating in the project, as well as other interested cities, to test the PI vision and determine the benefits of collaborative last mile logistics. As noted in this report, work will continue in developing and enhancing the blockchain infrastructure and training Living Lab stakeholders in using the developed tools. User interface enhancements, additional user functionality (e.g., shipment search functionality), additional Smart Contract monitoring events (based on Wave 2 city requirements), etc. are planned as the project progresses. In addition, it is anticipated that there will be a need to maintain the infrastructure, fix bugs that will inevitable arise (even though unit testing has been conducted), and generally continue to build the services to address future needs. As with all software development efforts, there are no ends to the development, only milestones reached.

7 Bibliography

[1] "Hyperledger Fabric," [Online]. Available: https://www.hyperledger.org/projects/fabric. [Accessed January 2023].

[2] "Consensys GoQuorum," January 2023. [Online]. Available: https://docs.goquorum.consensys.net/.

[3] "Corda - Open Source Blockchain Platform for Business," [Online]. Available: https://www.corda.net/. [Accessed January 2023].

[4] C. Ferris, "Does Hyperledger Fabric perform at scale," [Online]. Available: https://www.ibm.com/blogs/blockchain/2019/04/does-hyperledger-fabric-perform-at-scale. [Accessed January 2023].

[5] C. Gorenflo, S. Lee, L. Golab and K. S., "FastFabric: Scaling Hyperledger Fabric to 20,000," in *ICBC*, 2019.

[6] "Corda Finastra Case Study," [Online]. Available: https://www.r3.com/wp-content/uploads/2018/07/US_11_Finastra_CS_JUN26_final.pdf. [Accessed January 2023].

[7] K. L. Harris Niavis, "ConSenseIoT: A Consensus Algorithm for Secure and Scalable Blockchain in the IoT context," in *ARES*, Vienna, Austria, 2022.

Annex I : Documentation for Operation of the hlf-platform-k8s Framework

The hlf-platform-k8s framework requires the existence of a .env file containing essential environmental variables for smooth operation. The values of these variables are subsequently incorporated into Kubernetes Manifests YAML files and utilized by Kubernetes for seamless execution. Additionally, the framework relies on these environmental variables to proceed with operations according to the type of the node that is going to be deployed. Certain operations are exclusive to the Orderer Organization, while others are specific to Peer Organizations. Two different profiles, namely "orderer" and "peer", are used by the framework as templates for the appropriate configuration of the network. Even though there are numerous environmental variables available within the .env file, the only modifications that are required are the $ORG, $DOMAIN, $NODE_TYPE, $PEER, $USER, $CHANNEL_NAME AND $PATH, as documented in the Readme file of the Gitlab repository.

Table 5 presents the two different profiles that are used by the hlf-platform-k8s as well as their default - but totally configurable- environmental variables.

TABLE 5: HLF-PLAFTORM-K8S PROFILES

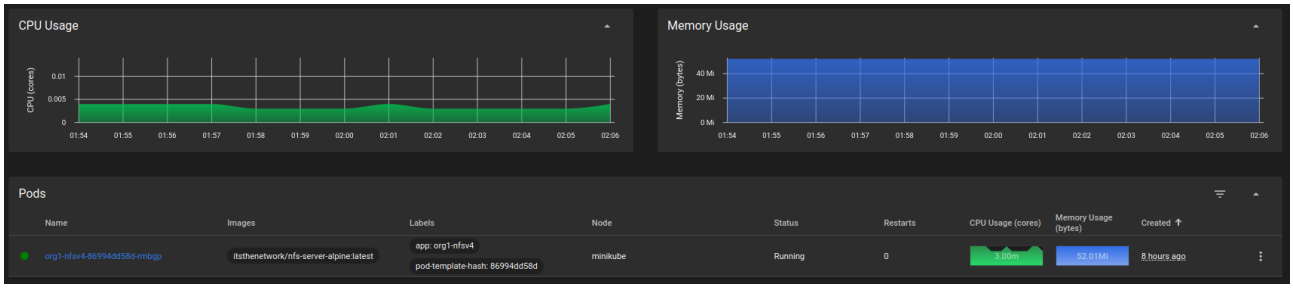
Environmental Variable	Orderer	Organization
ORG	<Orderer_Name>	<Organization_Name>
DOMAIN	<Orderer_Domain_Name>	<Orderer_Domain_Name>
NODE_TYPE	orderer	peer
PEER	3	2
USER	1	1

Switching profiles must be done with extreme CAUTION since decisions are being made according to the value of the environmental variable and according to the operation that needs to be executed. Even if protection mechanisms are being deployed in the scripts to detect such errors, it may be feasible that some errors may not be detected from the protection mechanisms if misuse or typos took place when entering the environmental variables.

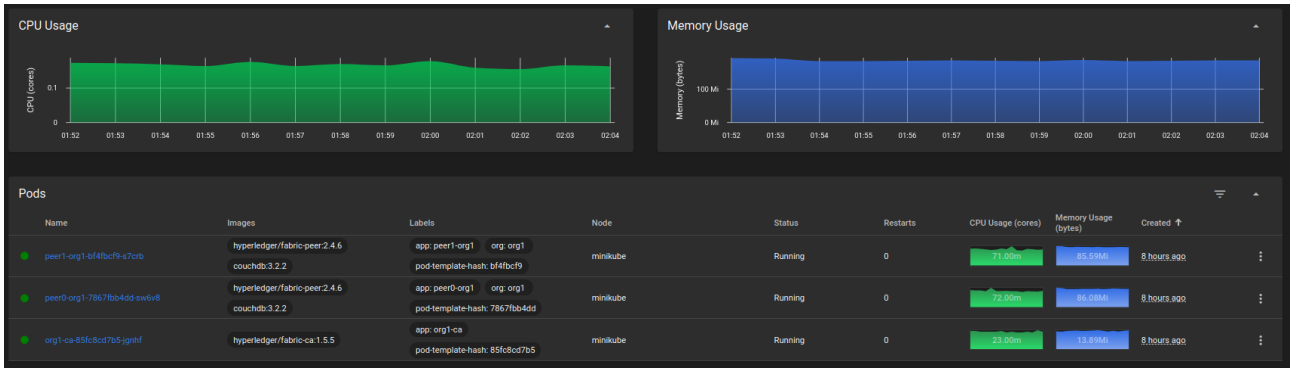
Embarking on the journey of unlocking the powerful features of HLF demands the foundational deployment of a Kubernetes cluster (in our case Minikube). In the act of bringing this digital landscape to life, a simple command, **"./minikube.sh up"** sets the stage. As the Kubernetes Dashboard opens in the browser, several possibilities arise. Verifying the proper setup with **"./minikube.sh status"** assures us that the **Minikube Kubernetes** instance is ready. This master node with 4 CPU cores and 8GB RAM is now ready to host the deployment of Hyperledger Fabric components.

Embarking on the next step of our journey involves the deployment of an Organization, a crucial step demanding a shift to the specific organization profile in the **.env** file. Let's examine **Org1**, where the stage is set with environmental variables such as **ORG, DOMAIN, NODE_TYPE, PEER,** and **USER**. As the process unfolds, executing **"./nfsv4.sh"** and **"./networkStart.sh deploy"** commands brings in the components of the organization.

The first command is quite crucial since it is responsible to deploy the NFSv4 Server. Later this NFSv4 server will be used as a common filesystem for all the necessary components that require access to the same filesystem. A quick visit to the Kubernetes Dashboard, selecting the **fabric-network-org1-nfsv4** namespace and navigating to Pods section, reveals the deployed NFSv4 Server – a primary component in the blockchain service.



Executing the second command allows us to deploy **(by default – but it can be managed through .env)** two (2) Peer Nodes (peer0-org1 and peer1-org1 Pods) and a Fabric CA (org1-ca Pod) within the **fabric-network-org1** namespace in Kubernetes. A visit to the Kubernetes Dashboard, Pods section, allows the user to see their organization's components instantiation within the system.



As a reminder, for multiple organizations **(including Orderer – only some environmental variables are changed in the .env)**, this deployment procedure remains the same, setting the scene for a multi-organization Hyperledger Fabric blockchain orchestration.

With all essential components from all Organizations and the Orderer Organization deployed within Kubernetes, we transition to a crucial phase. The objective: **constructing the Hyperledger Fabric network**. This operation, exclusive to the Orderer Organization, necessitates the selection of the orderer profile.

Executing the command **"./networkStart.sh artifacts create"** initiates the process. This command, a practical orchestration, collects MSP information from deployed organizations and builds vital channel artifacts, particularly the genesis block, leveraging the **configtxgen** binary.

```
ubuntu@misiakoulis:~/fabric-samples/multihost_k8s$ ./networkStart.sh artifacts create
✓ Creating Channel Artifacts
✓ Creating Channel mychannel genesis block
✓ Channel Artifacts are Ready
```

The result of this command's execution is succinct yet significant — the genesis block is now in place, paving the way of a functional HLF network.

With the genesis block successfully created, the next task involves channel creation, an operation exclusive to the Orderer Organization, necessitating the selection of the orderer profile.

Given that this tutorial focuses on a single organization (Org1 Organization) deployment, the genesis block encapsulates MSP information specific to this organization. The channel's name, declared in the **.env** file, restricts transaction submissions solely to members of the Org1 Organization, emphasizing the privacy aspect of **Hyperledger Fabric channels**.

Executing the command **"./networkStart.sh channel create"** is pivotal in this phase. This command utilizes the official binary **osnadmin**, relying on the condition that the **Admin API** is enabled on the Orderers **(available on latest versions)**. The ensuing result showcases the successful integration of all orderer nodes into the channel.

```
ubuntu@misiakoulis:~/fabric-samples/multihost_k8s$ ./networkStart.sh channel create
✓ Creating Channel mychannel
✓ Joining orderer.example.com Orderers to channel mychannel
✓ Retrieving orderer0.orderer.example.com Admin Certs
✓ Adding FQDNs entry to /etc/hosts
[sudo] password for ubuntu:
✓ Joining orderer orderer0.orderer.example.com to channel mychannel
✓ Checking if orderer orderer0.orderer.example.com has joined the channel mychannel
✓ Orderer orderer0.orderer.example.com has joined the channel mychannel
{
  "name": "mychannel",
  "url": "/participation/v1/channels/mychannel",
  "consensusRelation": "consenter",
  "status": "active",
  "height": 1
}
✓ Retrieving orderer1.orderer.example.com Admin Certs
✓ Adding FQDNs entry to /etc/hosts
✓ Joining orderer orderer1.orderer.example.com to channel mychannel
✓ Checking if orderer orderer1.orderer.example.com has joined the channel mychannel
✓ Orderer orderer1.orderer.example.com has joined the channel mychannel
{
  "name": "mychannel",
  "url": "/participation/v1/channels/mychannel",
  "consensusRelation": "consenter",
  "status": "active",
  "height": 1
}
✓ Retrieving orderer2.orderer.example.com Admin Certs
✓ Adding FQDNs entry to /etc/hosts
✓ Joining orderer orderer2.orderer.example.com to channel mychannel
✓ Checking if orderer orderer2.orderer.example.com has joined the channel mychannel
✓ Orderer orderer2.orderer.example.com has joined the channel mychannel
{
  "name": "mychannel",
  "url": "/participation/v1/channels/mychannel",
  "consensusRelation": "consenter",
  "status": "active",
  "height": 1
}
✓ Channel mychannel is Ready
```

It is worth noting that in Hyperledger Fabric, a channel acts as a private "**subnet**" facilitating confidential transactions between designated network members. The channel's definition encompasses members (organizations), anchor peers, shared ledger, chaincode application(s), and ordering service node(s). Each transaction is executed within a channel, with every participating party authenticated and authorized. Each peer joining a channel possesses a distinct identity provided by a membership services provider (MSP), ensuring authentication among channel peers and services.

Additionally, if a new organization wishes to join an existing channel, specific operations are conducted, falling under the Updating a Channel [https://hyperledger-fabric.readthedocs.io/en/latest/config_update.html] Configuration section. The assumption in this project is that all deployed organizations with a specific **PREFIX** in their namespace are considered channel members.

Currently, the channel stands as **ACTIVE**, yet the organization remains a non-member. To rectify this, the organization's peers must undergo the process of becoming channel members. This operation is exclusive to the organization, mandating the selection of the organization profile.

Executing the command **"./networkStart.sh channel join"** initiates this crucial step. The ensuing result, as depicted below, illustrates the successful integration of all the organization's peer nodes into the channel.

```
ubuntu@misiakoulis:~/fabric-samples/multihost_k8s$ ./networkStart.sh channel join
✓ Joinining org1.example.com Peers to channel mychannel
X Orderer Certificates are Missing.
✓ Checking if orderer0.orderer.example.com is UP
✓ Orderer orderer0.orderer.example.com is UP
✓ Choosing orderer0.orderer.example.com to connect
✓ Joining org1.example.com Peers
✓ Retrieving peer0.org1.example.com Client Certs
✓ Retrieving peer0.org1.example.com Client Operations Certs
✓ Adding FQDNs entry to /etc/hosts
✓ Joining peer peer0.org1.example.com to channel mychannel
2023-03-01 11:23:04.555 EET 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-03-01 11:23:04.944 EET 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
✓ Checking if peer peer0.org1.example.com has joined the channel mychannel
2023-03-01 11:23:05.014 EET 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-03-01 11:23:05.078 EET 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
✓ Peer peer0.org1.example.com has joined the channel mychannel
{
  "height": 1,
  "currentBlockHash": "ZSCe/TB0xDIm7Isn4M8MH+IIKUmCQVCterSMuzTON64="
}
✓ Retrieving peer1.org1.example.com Client Certs
✓ Retrieving peer1.org1.example.com Client Operations Certs
✓ Adding FQDNs entry to /etc/hosts
✓ Joining peer peer1.org1.example.com to channel mychannel
2023-03-01 11:23:05.569 EET 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-03-01 11:23:06.000 EET 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
✓ Checking if peer peer1.org1.example.com has joined the channel mychannel
2023-03-01 11:23:06.074 EET 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-03-01 11:23:06.166 EET 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
✓ Peer peer1.org1.example.com has joined the channel mychannel
{
  "height": 1,
  "currentBlockHash": "ZSCe/TB0xDIm7Isn4M8MH+IIKUmCQVCterSMuzTON64="
}
✓ Channel mychannel is Updated
```

This action marks the organization's official membership within the active channel, paving the way for seamless interaction and transaction execution within the Hyperledger Fabric network.

In conclusion, with the establishment of the Hyperledger Fabric network among the deployed organizations, a solid foundation has been laid for collaborative blockchain operations. Each organization now holds the responsibility of deploying their respective chaincode onto their peer nodes and, subsequently, committing these essential components into the shared channel. This marks a juncture where the decentralized nature of the network comes to life, empowering each organization to contribute and interact within the Hyperledger Fabric ecosystem. As the network continues to evolve, the collaborative efforts of each organization will play a crucial role in realizing the full potential of transparent, secure, and decentralized transactions facilitated by Hyperledger Fabric.