Use Cases Development Guide (RFC-020)

Early Draft	The RFC is proposed (nice to have) but has no content.

Document Maintainers: Andi Gabriel Tan 2023. List of other contributors in Annex. 1.

Copyright

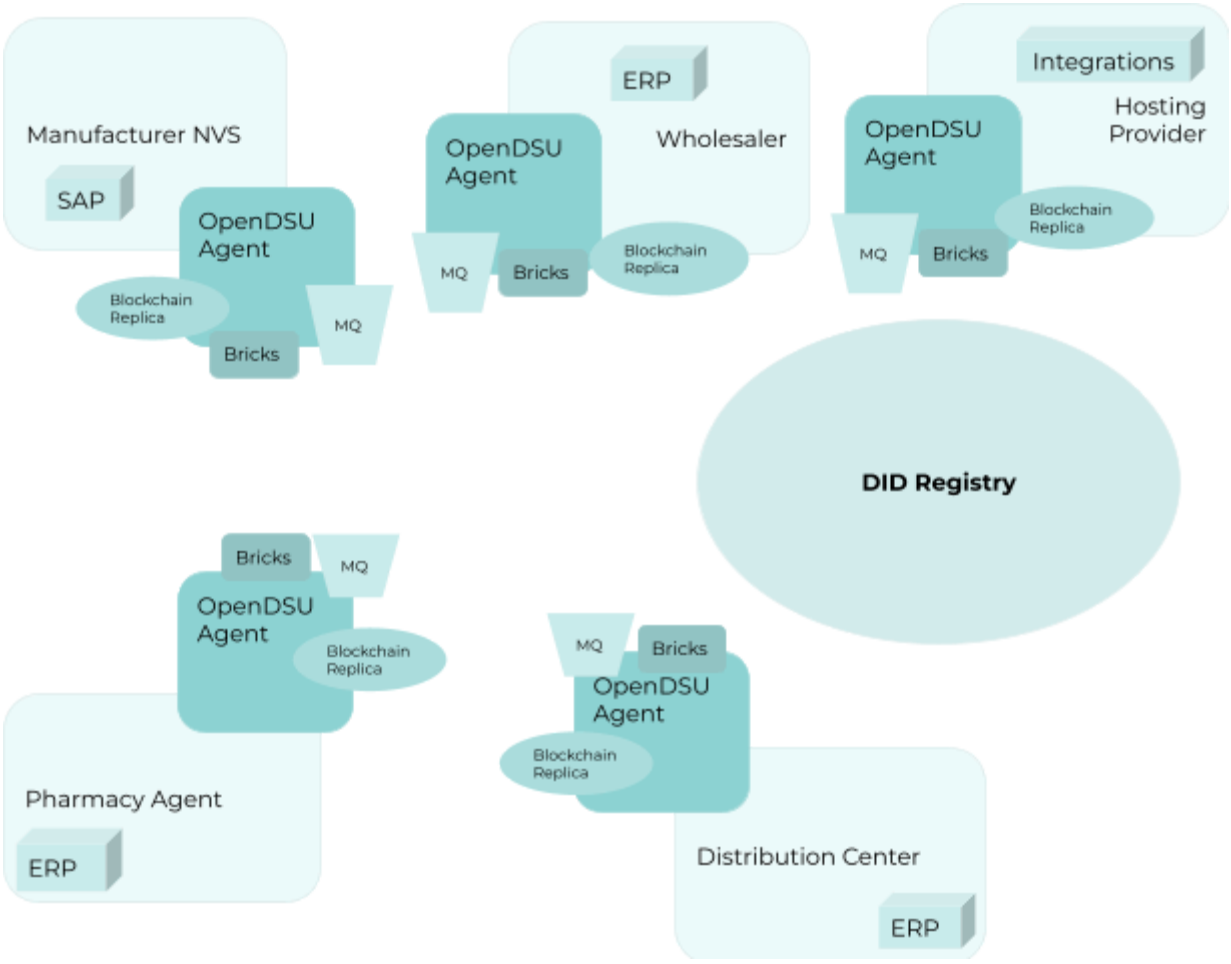
Contributors	Description
Axiologic Research www.axiologic.net	New content, cleaned the original texts under PharmaLedger Association and Novartis funding. MIT licensed content accordingly with the contracts. Publish and maintain www.opendsu.com site.
PharmaLedger Project Members www.pharmaledger.eu	Review, feedback, observations, new content and corrections MIT licensed accordingly with the consortium agreements.
PrivateSky Research Project	Initial content (www.privatesky.xyz). MIT licensed content accordingly with the contracts. https://profs.info.uaic.ro/~ads/PrivateSky/

Abstract

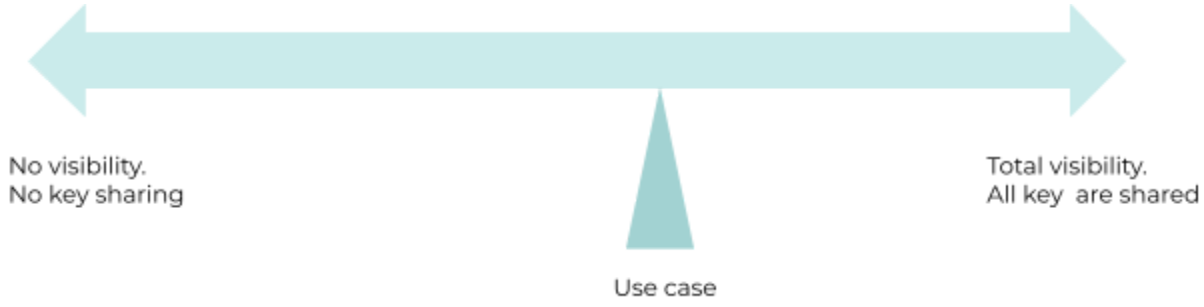
The purpose of this document is to summarize a general approach that use-case implementation can be used to develop projects using OpenDSU. Besides providing insights on the development phases, this document opens the black box represented by the blockchain and the DSU Storages by explaining how the integration with the legacy system should work.

1. Big Picture

Use cases will always involve **different actors**. In order to exchange data in a secure way with OpenDSU they will need an **OpenDSU agent** or **wallet** that will take care of all OpenDSU-related operations, such as the storage of the DSU's key, the bricks storage in a local or remote server, or the anchoring of keys' Anchor IDs in the blockchain. These agents can either be used locally (which is optimal to attain data self-sovereignty) or remotely in the cloud when it's necessary (for example if you need the agent to be available 24 hours a day). We will often have a mix of local agents (or edge) where the most sensitive data will be stored and cloud agents to get the benefits of cloud storage, such as availability.

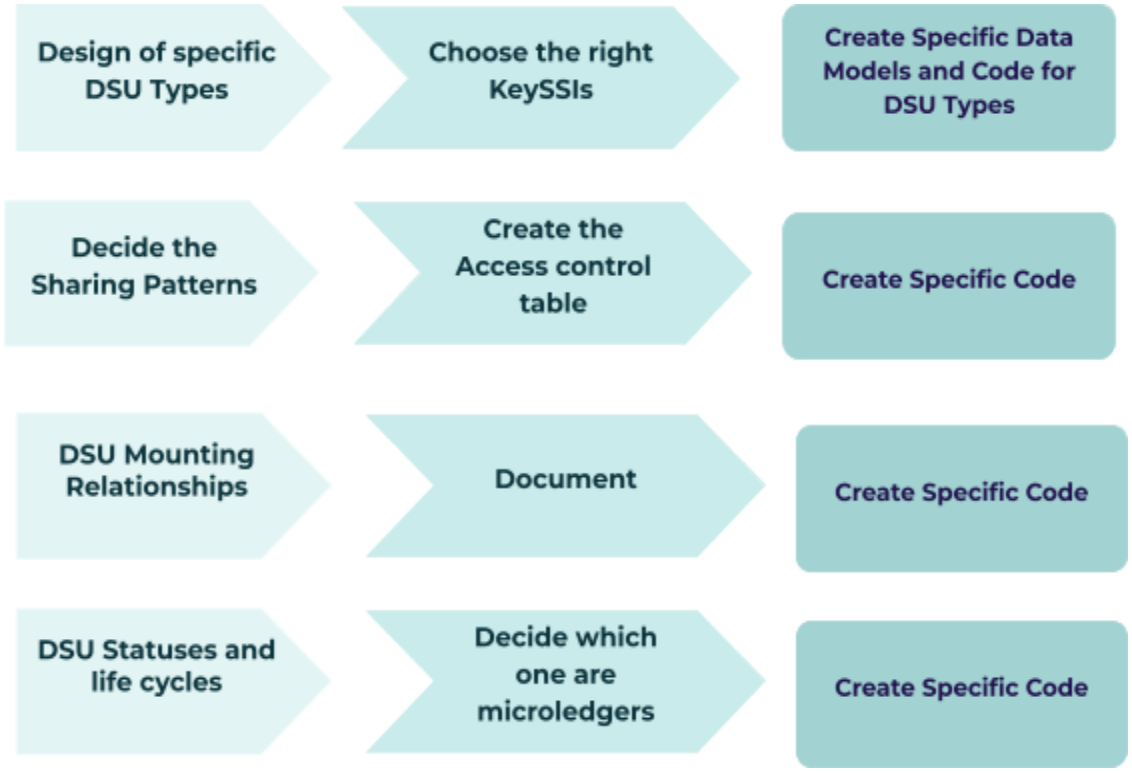


The data stored in the DSU is only accessible using the associated keys. These keys will be resolved to obtain necessary information about the brick storages or anchoring services that will allow the agents to reconstruct the DSU in its execution environment. These mechanisms allow the keys' owners to keep data hidden from the other actors. The highest key of a family (SeedSSI) is providing total control over the DSU it is associated with. In order for these key owners to share control of their DSU, they can send the same key to their collaborators. They can also decide to give restricted access to their DSU without sharing the original key, by using key derivation processes to create new keys that they will be able to share safely (e.g. SReadSSI, for read-only access). Use cases should use a mix of keys to create access control policies for their DSUs.



2.1. Development Process from the OpenDSU Perspective

We can see a DSU as a record in a table in a relational database for simplicity, but the implementation is different from this. As stated above, access to a DSU is only done with a KeySSI that is unique for each DSU. When designing a use case, the architect creating an OpenDSU-based solution should go through the steps explained in this section.

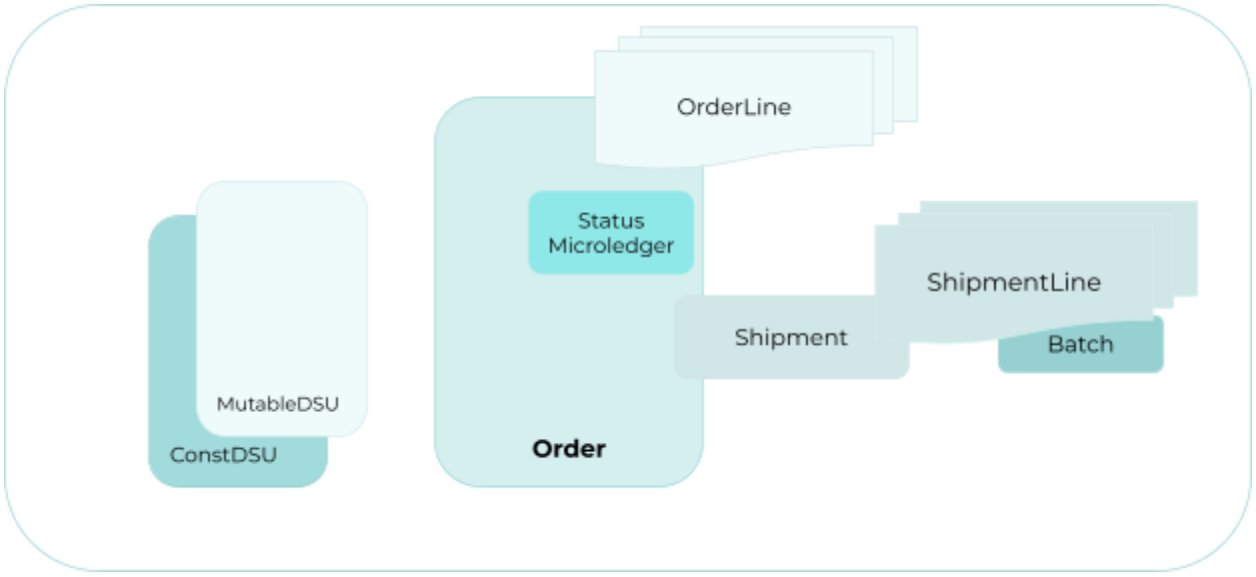


DSU Types

A DSU is one unique technology enabling safe storage and safe transfer of data through key ownership. Still, DSUs can be created for different purposes, different processes and/or for different people. It is possible to configure these different DSUs under specific types to facilitate their replication.

Each use case will have to develop specific DSU Types to achieve their goals. DSU Types are similar in their design to Object Oriented Programming classes or to the design of tables in relational database models. They are neither classes nor tables, but they play a similar role.

In the following example, we can see that different DSUs were created for different steps of a product supply chain. We can imagine that different people would need different types of access to these DSUs. The transporter would need access to create and modify the shipment DSU, while the customer would only own a key to read the same DSU to see their order and know at what time they will receive the shipment.



DSU	Main KeySSI type	KeySSI for mutable Real DSU
Order		
Orderline		
Order Status		
Shipment		
Shipment Line		
Batch		

Access control table (defines communication lines between agents)

DSU types	Created by	Write mode (Anchoring)	Read Only Sharing with	Existence Shared with
Order	PHA	PHA	WHS, MAH	HA
Orderline	PHA	PHA	MAH	HA
Order Status	PHA	PHA, DST		HA
Shipment	MAH	MAH	DST, COU	HA
Shipment Line	DST	DST		HA
Batch	MAH	MAH		HA

DSU Mounting Relationships

DSU	Mounted DSU	Mounted Points	Mounting Way (write, read, existence)
Order	Orderline types	/orderlines/1 /orderline/2 /orderline/3	existence
Orderline			
Order Status			
Shipment			
Shipment Line			
Batch	Batch type	/batch	read

Shared DSUs with Statuses (micro-ledgers)

Order Statuses for Order	Description
Created	
At WHS	
At MAHs	
Shipped	

Choose a Blockchain Domain for the Use Case

In OpenDSU, blockchains are used for anchoring. In other words, all anchor IDs of the DSUs will be written in a decentralized and immutable ledger. In order for all actors present in the use case to communicate, we have to choose a common Blockchain Domain from which we will be able to get the correct ledger system (e.g. ePI).

Agents

Trading (Agent)	Partner	Role & Description	Blockchain Domain for Bricks
MAH		Manufacturer	e.g. msd.epi, nvs.epi. pl.epi
PHA		Pharmacy	

WHS	Wholesaler
DST	Distributor
HA	Health Authorities/External Auditors
PAT	Patient
COU	Courier
HOS	Hospital

3. Generic components for all use cases

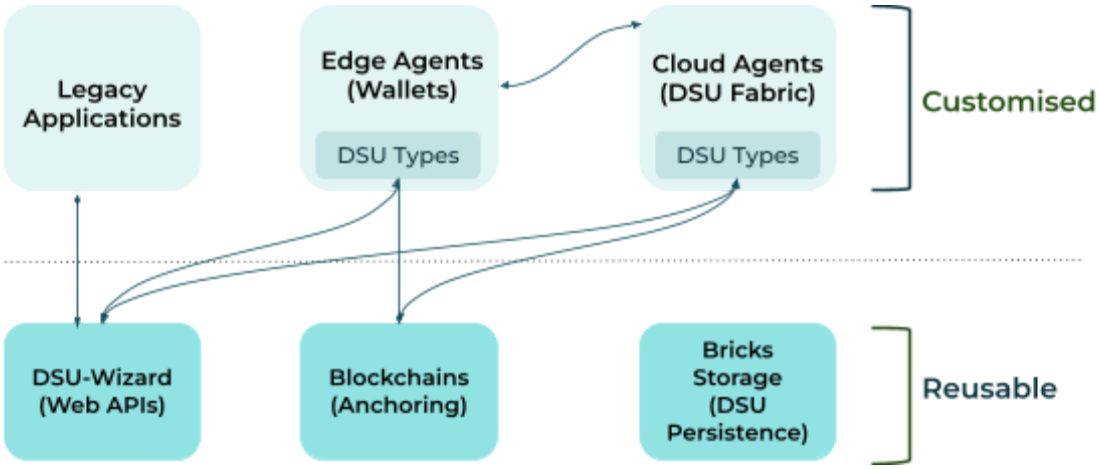


Diagram 1: Relevant architecture boxes for all Use Cases

In this section, we assume that in all use cases we were able to identify agents, legacy applications, the blockchain used for anchoring DSUs, and the bricks storage used to store encrypted DSUs off-chain (or near-chain: some data directly anchored and other off-chain). The cloud and edge agents are special applications created independently for each use case but reusing the same general techniques.

The legacy applications include existing software such as SAP, ServiceNow etc., that will be integrated with Pharmaledger using a new component that we describe below as “DSU-Wizard”. In the following subchapters, we will describe the concept of agents and the DSU-Wizard. All information about the other components can be found in the OpenDSU documentation.

Agents

An agent can have access to one or more wallets, and each wallet has access to one or more DSUs. In general, the user who controls an agent is a person and not a company, but depending

on the role in the company, it is possible to imagine a superuser (e.g. a manager) supervising the wallets of other employees.

Diagram 2: Agent Overview

Edge Agent

An Edge Agent is a software entity that securely holds data and keys on behalf of a user, locally. The edge agent could run on a computer, a mobile device or a browser. Running user interaction through edge Agents is the best way to obtain data self-sovereignty and provide control of data to its owner/patient, without intermediaries. Sometimes, in order to satisfy certain business needs, it is necessary to have Cloud Agents that are under the control of companies or external intermediaries.

Cloud Agents

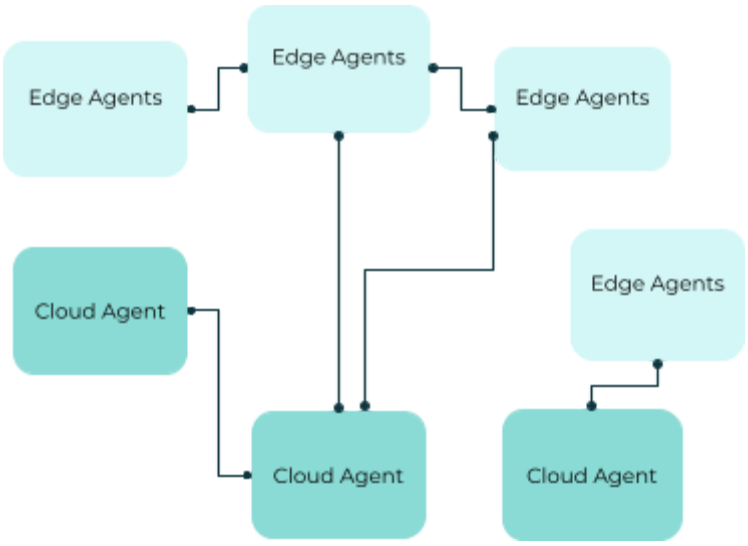


Diagram 3: Communication between agents

A Cloud Agent is a software entity that securely holds data and keys on behalf of a company remotely. As for edge agents, cloud agents have access to one or multiple wallets and those wallets contain DSUs. The advantage of cloud agents is that they can be available 24 hours a day from anywhere. Cloud agents are generally controlled and maintained by a company and not by a person. However, concrete people controlling edge agents can control the same wallets as the cloud agents if they own the same keys.

DSU Wizard

The integration between legacy systems and the world of the OpenDSU Agents capable of manipulating DSUs is managed by a component that we call “DSU Wizard”. The DSU Wizard is a set of Web APIs that are called from the external systems (the legacy systems). The Purpose of the DSU Wizard is for the OpenDSU SDK to be used by the legacy system code by offering a set of simple Web APIs to create or update the DSUs as required by businesses.

Diagram 4: Wizard phases for each new DSU

As the name implies, the DSU Wizard is a set of APIs that should be called in a specific order, starting from the initialization of a creation process until the actual creation of bricks and anchors, as expressed in the above diagram.

Diagram 5: Wizard phases for DSU updating

The DSU Wizard is also capable of updating an already existing DSU, as depicted above.

4. The standard model of operations

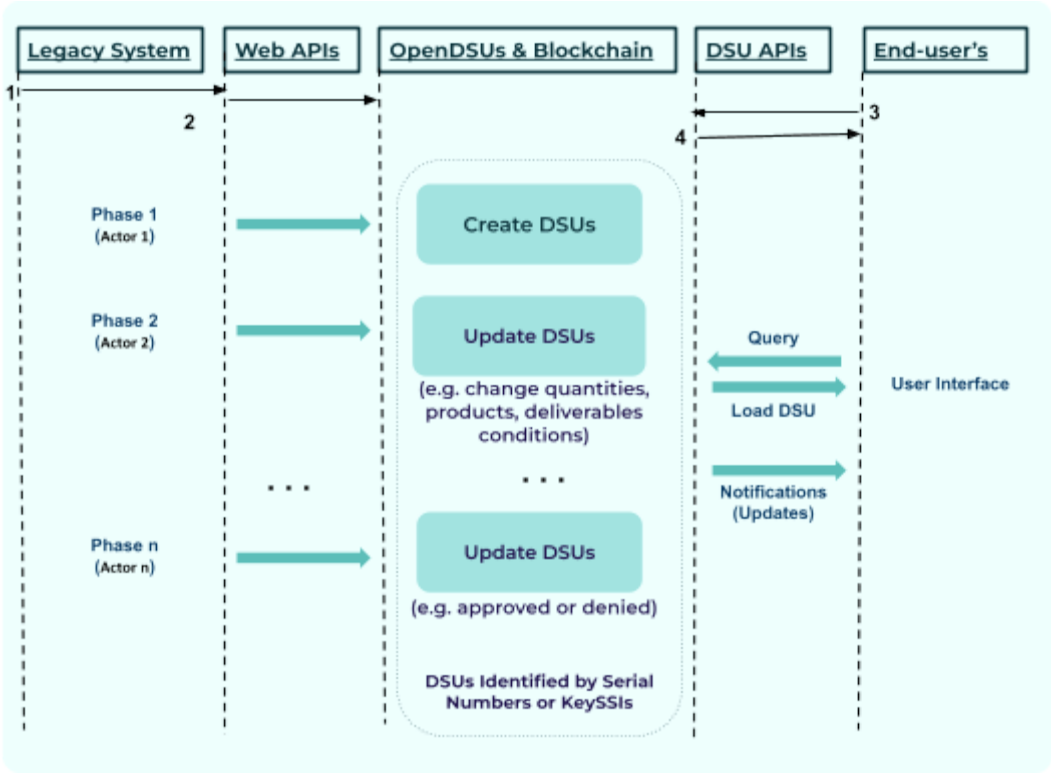
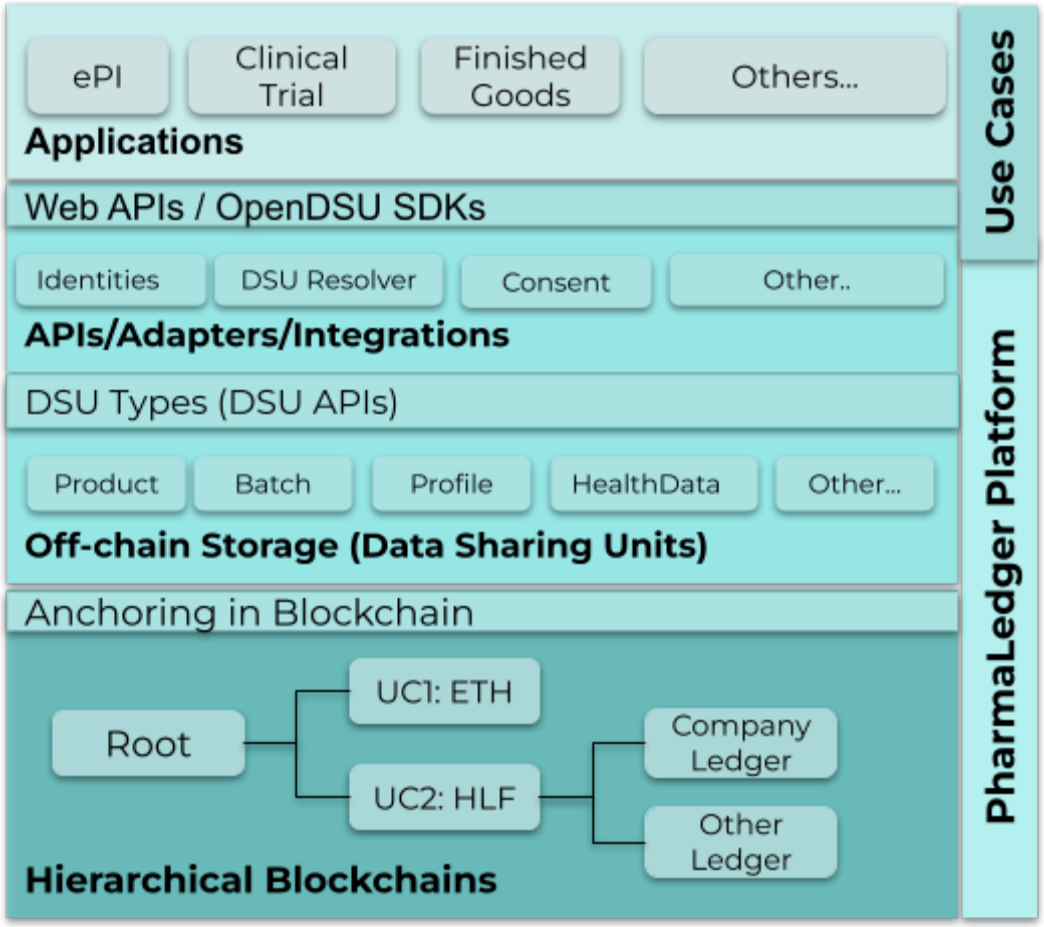


Diagram 6: Wizard phases for DSU updating

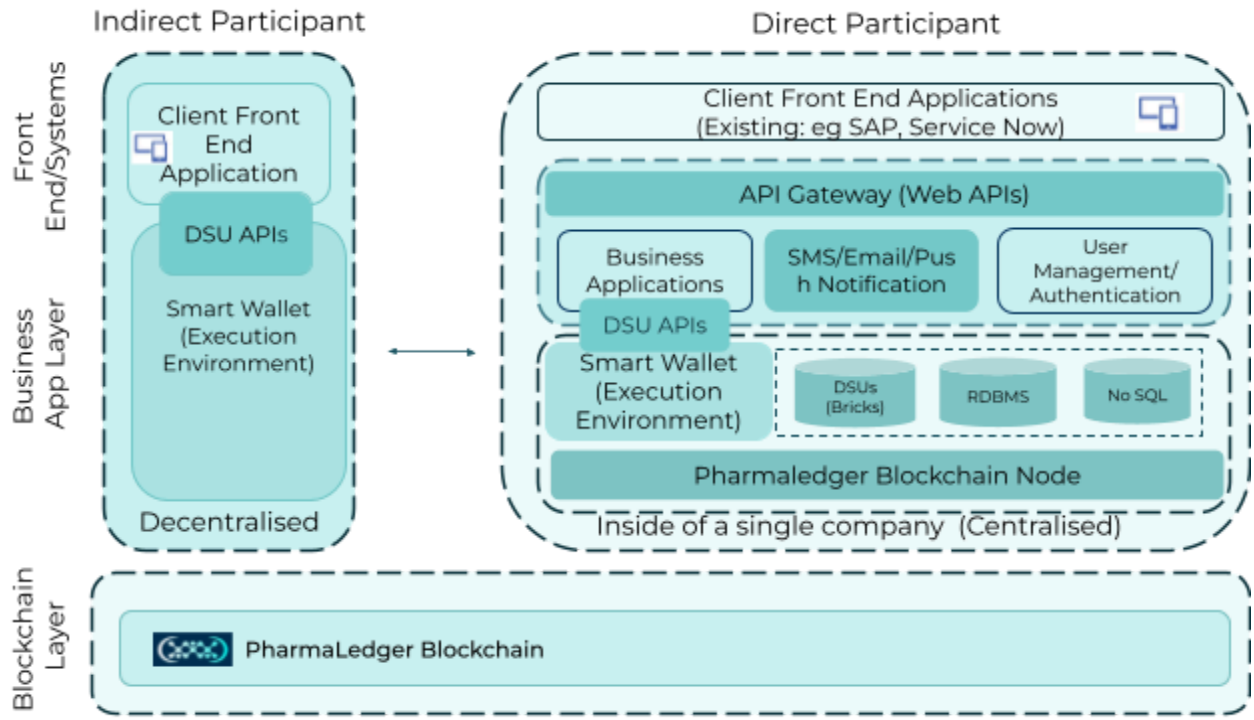
DSUs can be seen as an extension of the blockchain. Each DSU is a micro-ledger (a set of operations or transactions validated by anchoring in a parent blockchain) and any imaginable use case only updates the status of DSUs using DSU APIs. Being a micro-ledger, it offers an alternative method to implement "off-chain" smart contracts in a way that does not suffer the same problems as on-chain smart contracts regarding confidentiality and scalability.

5. Annexes (examples/diagrams)

5.1. Major components

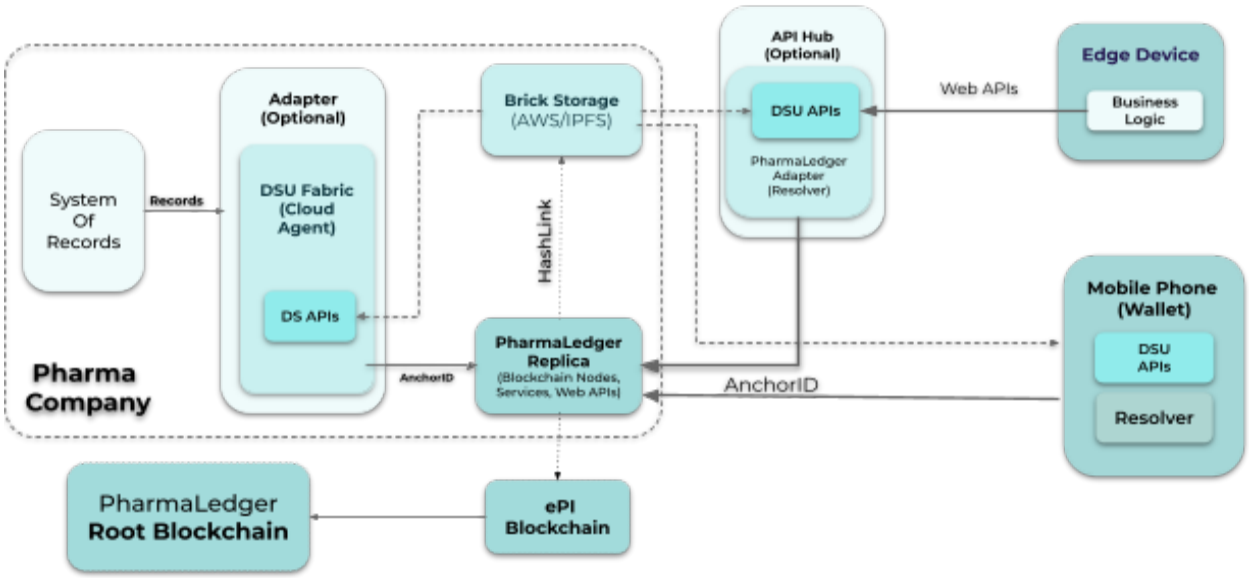


5.2. Indirect and direct participants

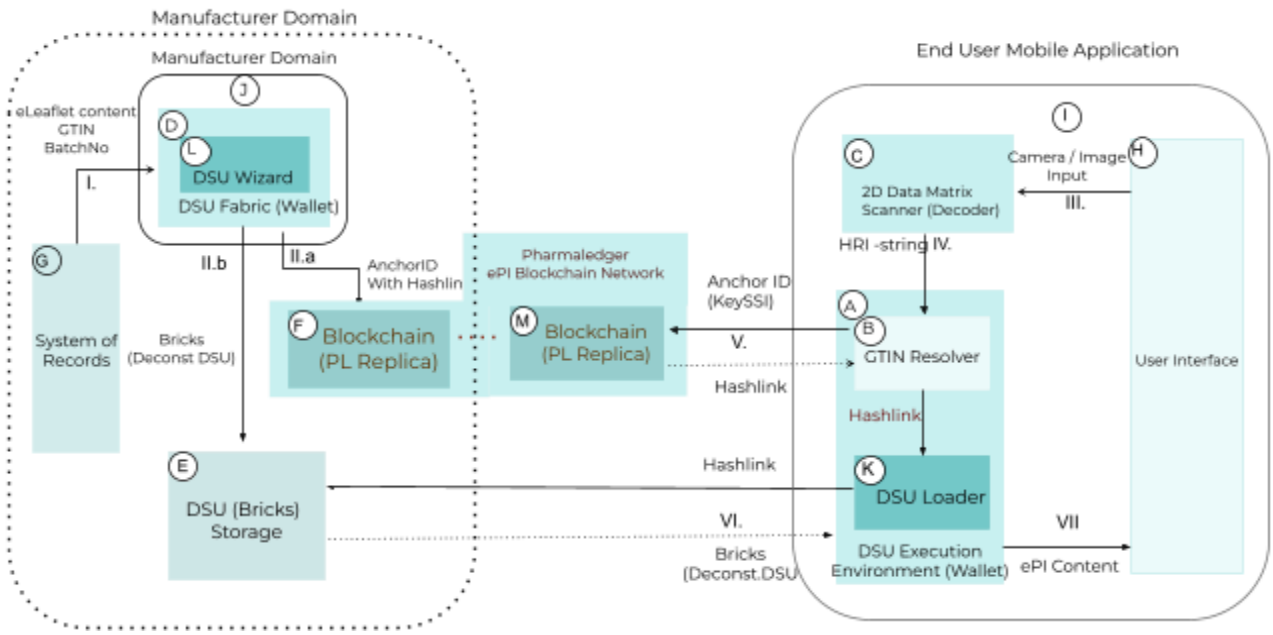


5.3. Generic Architecture

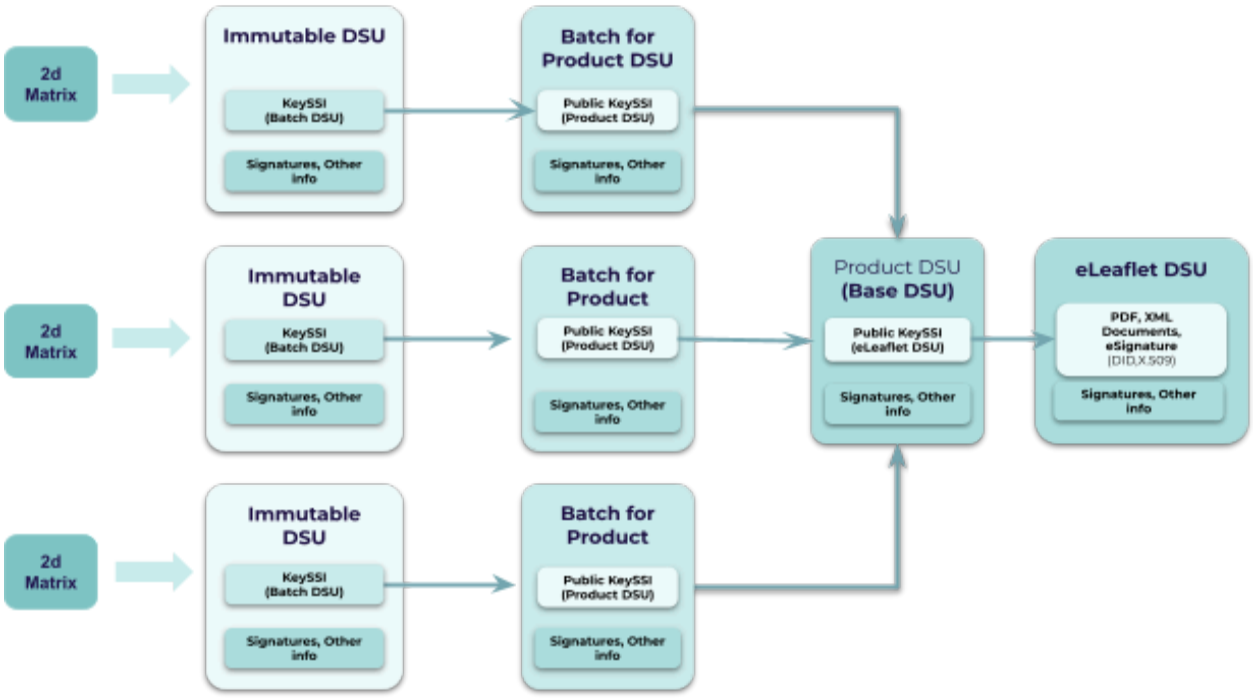
AnchorIDs are stored on the ePI Blockchain (Consortium Blockchain). The ePI Blockchain is readable by everyone (Public), but only writable (AnchorID) for authorized parties, such as Pharma Companies.



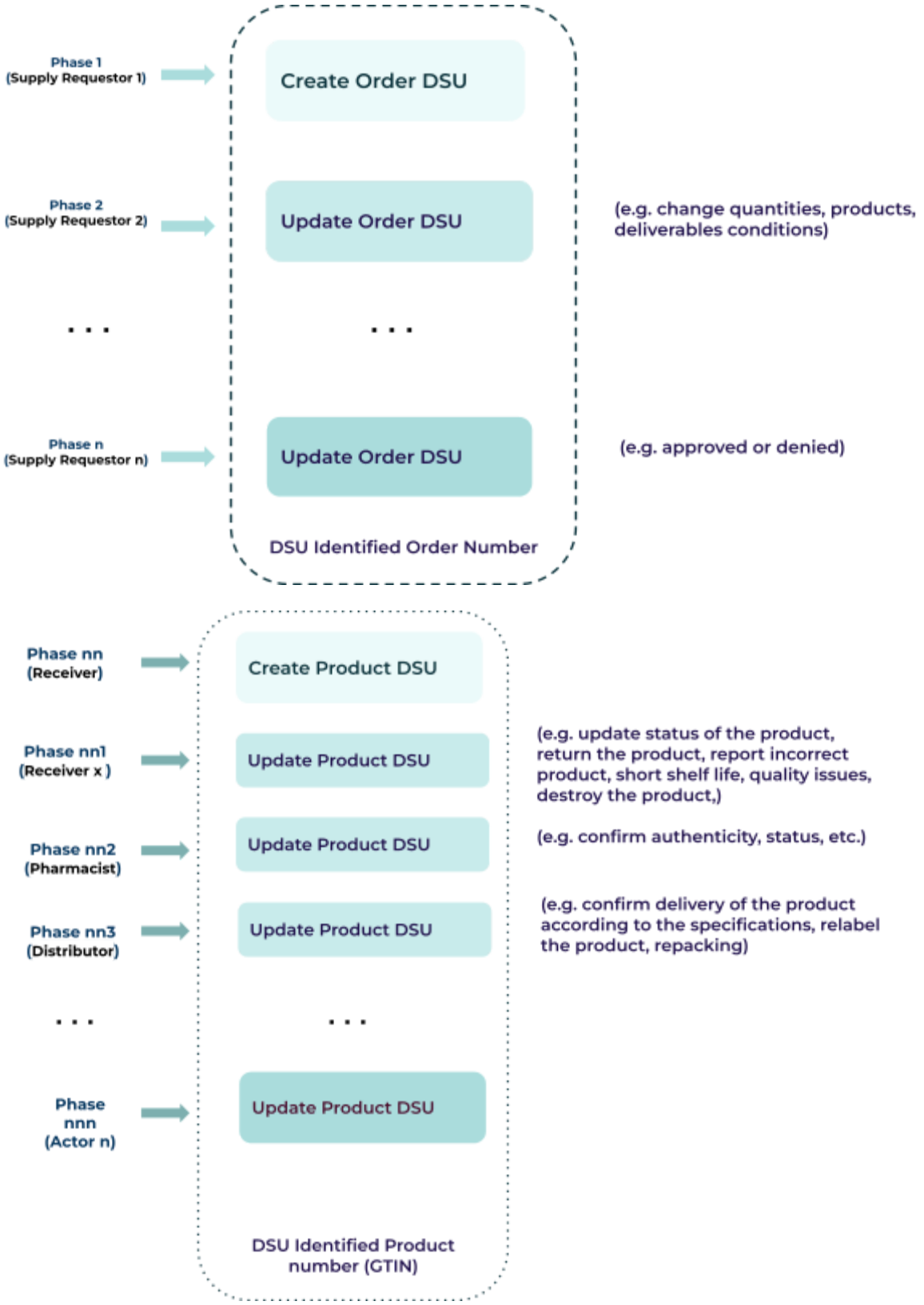
5.4. ePI Architecture

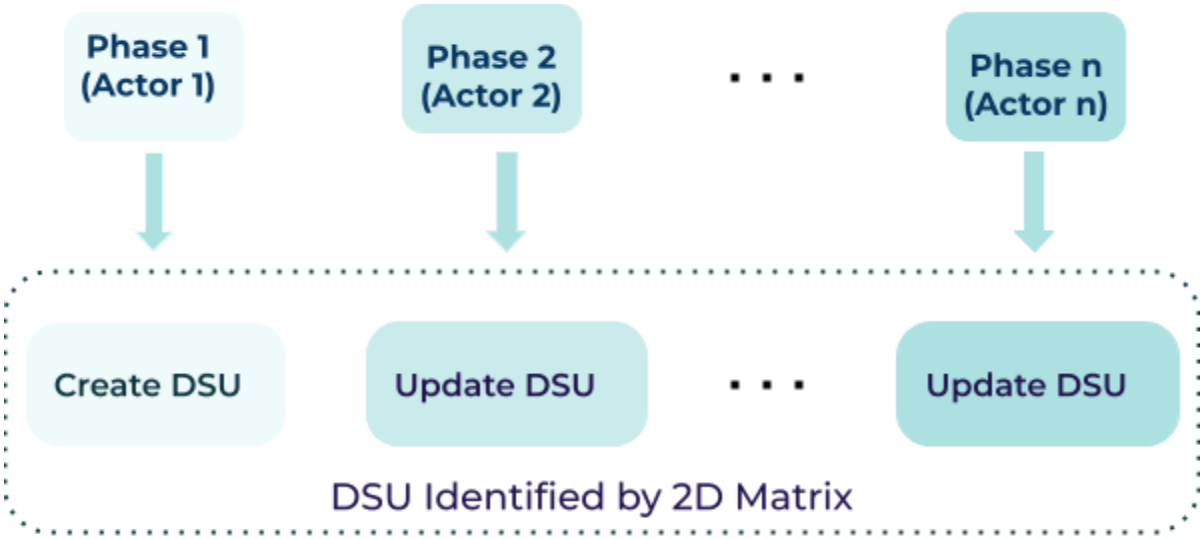


5.5. DSU Model design example

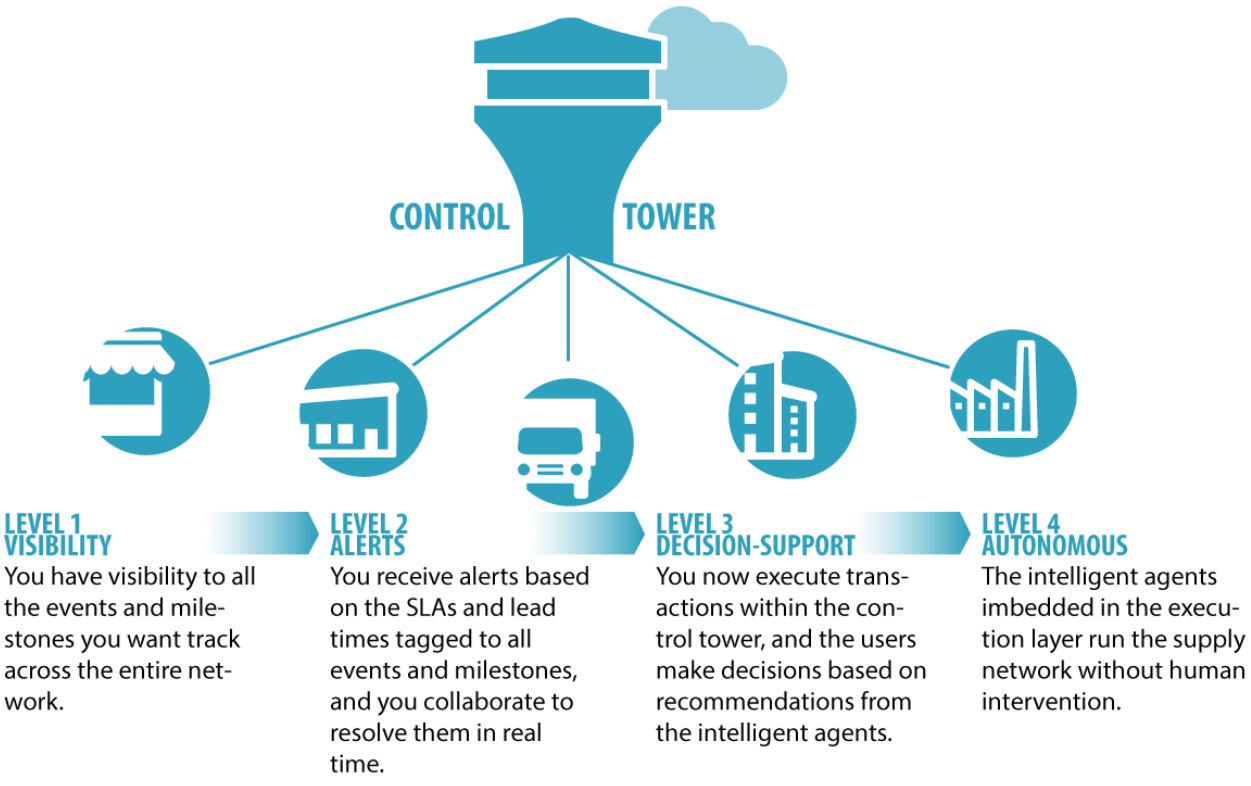


5.6. Other use cases





5.7. Control tower



Annex 1. Contributors

Current Editor	
Andi-Gabriel Țan	andi@axiologic.net

Contributors Axiologic Research	
Adrian Ganga	adrian@axiologic.net
Andi-Gabriel Țan	andi@axiologic.net
Cosmin Ursache	cosmin@axiologic.net
Daniel Sava	daniel@axiologic.net
Nicoleta Mihalache	nicoleta@axiologic.net
Valentin Gérard	valentin@axiologic.net

PrivateSky Contributors	
Alex Sofronie	alsofronie@gmail.com(DPO)
Cosmin Ursache	cos.ursache@gmail.com(UAIC)
Daniel Sava	sava.dumitru.daniel@gmail.com(HVS, AQS)
Daniel Visoiu	visoiu.daniel.g@gmail.com(SGiant)
Lenuța Alboaie	lalboaie@gmail.com(UAIC)
Rafael Mastaleru	rafael@rms.ro(RMS)
Sînică Alboaie	salboaie@gmail.com(UAIC)
Vlad Balmos	vlad.balmos@gmail.com(Code932)

PharmaLedger	
Ana Balan	bam@rms.ro (RMS)
Bogdan Mastahac	mab@rms.ro (RMS)

Cosmin Ursache	cos@rms.ro (RMS)
Rafael Mastaleru	raf@rms.ro (RMS)