BLOCKCHAINS FOR CONDITION MONITORING

B4CM

Project title:	Blockchains for Condition Monitoring
Starting date:	1st December 2018
Duration in months:	48
Call identifier:	H2020-S2RJU-OC-2018
Topic:	S2R-OC-IPX-03-2018
Grant agreement number:	826156

Deliverable D1.1

B4CM software framework: An implemented software framework and supporting documentation for the monitoring of data exchanges and attribution of associated costs in industrial RCM systems

Due date of deliverable:	31st August 2021
Actual submission date:	31st August 2021 (Updated 13th January 2022)
Lead contractor for deliverable:	University of Birmingham (UoB)
Dissemination level:	Public
Revision:	Final

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826156.

Shift2Rail

Authors

Author(s)	**University of Birmingham (UoB)** Rahma Alzahrani
Contributors(s)	Callum Jones
	John Easton

Document History

Date	Description
31st August 2021	Draft for approval.
13th January 2022	Updated in line with reviewer comments: • Update to S2R logo • Addition of new section justifying Hyperledger platform choice with reference to other available options • Additional of link to case study description in D2.1 and diagram of stakeholder context for monitoring scenarios • References added as appropriate to detailed descriptions of case studies and need for the use of Escrow in published papers linked to the B4CM project and D2.1 • Note that reference ontology should be used for data descriptions, and that this will be investigated before the end of the project • Flowcharts updated to include specific labelling of Y/N pathways • Chaincode transaction model moved to separate section • Structural use of bold headings updated to numbered sections

Disclaimer

Executive Summary

The aim of this deliverable is to report on the B4CM project software framework, giving basic instructions for deployment and providing documentation enabling the usage, extension, and maintenance of a deployed system in the context for which it was developed. The project code has been made available via a public git repository (release versions only), enabling users to easily access and extend the work for their needs, and can be accessed via GitHub (https://github.com/B4CMProject/B4CMProjectSoftwareReleases).

The document begins by describing the software stack being used by the team, before looking in detail at each of the software constructs that form the blockchain network deployment of the framework. Finally, the framework is demonstrated in a toy industry context based on the UK rail industry.

Abbreviations and Acronyms

Abbreviation / Acronym	Definition
API	Application Programming Interface
B4CM	Blockchains for Condition Monitoring
CA	Certificate Authority
DB	Database
DfT	Department for Transport
EU	European Union
ECC	Elliptic Curve Cryptography
HLF	Hyperledger Fabric
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MSP	Membership Service Provider
SDK	Software Development Kit
UI	User Interface
UK	United Kingdom

Table of Contents

1. Background to the B4CM Project 1
2. Objective / Aim of Deliverable 2
3. Selection of Blockchain Platforms Based on Candidate Case Studies 3
3.1 Automation and Contracts 5
3.2 Consensus 5
3.3 Identity Management and Roles 6
3.4 Resource Management 6
3.5 Platform Selection 6
4. System Architecture 7
5. High-level Description of Software Components, Operational Context and Workflow . 12
5.1 Technology Stack 12
5.2 Intended Operational Context 12
5.3 Transactional workflow and Determination of Consensus 13
6. Chaincode Transaction Model 15
6.1 Transaction Registration 16
6.2 Use of Transactions by Providers 17
6.3 Use of Transactions by Consumers 21
6.4 Use of Transactions by Automation 23
7. Demonstration Application 26
8. Conclusions 37

1. Background to the B4CM Project

Over the past decade there has been a significant level of investment throughout Europe in the digitalisation of the rail network. This includes the installation of sensors on the infrastructure and vehicles, the deployment of next generation traffic management systems that allow real-time management of the system, and the provision of mobile applications for passengers and staff. Despite the wealth of new data provided by these systems, the railways are still struggling in their aspiration to be an information-led industry due to a lack of traceability of information usage, and the commercial barriers between stakeholders.

Blockchains are a disruptive technology that have the potential to accelerate the development of rail as the primary medium-distance carrier within the wider multi-modal transportation system. Directly funded by the rail industry via the EU Shift2Rail Joint Undertaking, the B4CM project will identify key use cases for the technology within the railways, deliver a blockchain-based testbed that enables the benefits of the technology to be formally evaluated, and demonstrate the value of blockchains in the attribution of data costs across organisational boundaries within the European rail sector.

The overall aim of the B4CM project is to develop and deliver a blockchain-based testbed for the attribution of data costs across organisational boundaries, and to demonstrate the operation of the framework and in the context of the European Rail Industry, enabling future developers to extend the tools produced based on a known working configuration.

B4CM has the following research and training objectives:

Objective 1: To identify and develop use cases that support the application of blockchain in the railway sector;

Objective 2: To develop an implementable blockchain framework for the attribution of data costs in systems crossing organisational boundaries;

Objective 3: To evaluate mechanisms for the incorporation of the developed blockchain framework into the financial processes of the European rail sector;

Objective 4: To develop a testbed, demonstrating the operation of the framework in the context of rail sector, enabling future developers to extend the tools produced based on a known working configuration;

Objective 5: To disseminate the findings of the project and the lessons learned to influence best practice in innovation and technology uptake in a key and evolving field within the European rail sector;

Objective 6: To support the development of a researcher in gaining a PhD and thus generating a skilled specialist valuable to the European rail sector.

This document, reporting the B4CM project software framework, is written primarily in response to Objective 2 of the B4CM project.

2. Objective / Aim of Deliverable

As outlined in the description of work this deliverable will report on the B4CM project software framework, giving basic instructions for deployment and providing documentation enabling the usage, extension, and maintenance of a deployed system in the context for which it was developed. The project code has been made available via a public git repository, enabling users to easily access and extend the work for their needs.

3. Selection of Blockchain Platforms Based on Candidate Case Studies

Within the B4CM project, the ultimate performance of the developed framework to improve trust, automate a fair cost attribution process and payment, and enforce agreements between parties, will be evaluated through the use of two industrial case studies. In order to provide continuity with previous work in this area, the B4CM team have based these on published work by RSSB in project T857 [6] [7] [8].The first case study will be the Unattended Overhead Line Equipment Monitoring System (UOMS), a train-based system monitoring infrastructure. The second will be axle journal bearings monitoring system (RailBAM), an infrastructure-based system monitoring trains. In UOMS, equipment is mounted on a Class 390 train and used to monitor and measure the health of the pantograph line which belongs to the infrastructure. In the second case study, RailBAM, the acoustic devices are mounted on the main infrastructure track and used to monitor the axle journal bearing upon which the wheel of rolling stock is rotating. All case studies involve cooperation between different stakeholders across the rail industry. In other words, there are several parties in the rail industry interested in the data generated in both case studies, such as Network Rail, Train Operating Companies (TOCs), Freight Operating Companies (FOCs), and other train manufacturers and maintainers. In Figures Figure 1 and Figure 2, the main stakeholders' roles for each case study are depicted to show their responsibilities which increase their need to enquire the generated monitoring data. These include the Train Operating Companies (TOCs and FOCs for passengers and freight respectively in the UK), the Rolling Stock leasing companies (ROSCOs), and the manufacturers and suppliers of equipment to the industry. The key relationships between these entities in the overall governance structure of the industry in Great Britain is illustrated in Figure 6. A more detailed introduction to the B4CM case studies is provided in B4CM Deliverable 2.1. These will then be worked up as the basis for the demonstrator to be developed under Work Package 3.

In the remainder of this section of the document we will provide an overview of the features of a range of blockchain platforms, and their relevance to needs of the B4CM project.

Figure 1: Main stakeholders in monitoring axle journal bearings.

Figure 2: Main stakeholders in the monitoring of OLE.

Based on the permission levels they support, blockchain frameworks are primarily categorized as public/permissionless blockchain networks or private/permissioned blockchain networks. A public blockchain network, e.g. Bitcoin, is open to the public to join, i.e. anonymous participants have uniform access privileges to the network ledger. In these cases powerful consensus mechanisms must be imposed to preserve security, particularly in terms of the integrity of data being entered into the ledger. The most popular consensus algorithm used in public blockchains is Proof-of-Work [1]. In contrast, in private blockchain networks, participants are known / identified by specific credentials to enable tailored restrictions to be placed on their access to the ledger and the actions they can perform.

In the B4CM framework, there is a clear requirement that the identities of all participants in the network must be known, thus protecting the integrity of what is essentially an industrial system. To deliver this, the developed proof of concept will be implemented using a blockchain network supporting the auditing and tracking or actors and processes. There are several blockchain platforms that could be employed in implementing the proposed model. Determining the most appropriate one has a considerable influence on the design as there are no one-size-fits-all-platform blockchain initiatives. Therefore, a tradeoff analysis was conducted on the most used blockchain platforms: Ethereum [1], [2], Fabric [3], Sawtooth [4], and Iroha [5], based on the criteria listed below in Table 1.

Table 1: Comparison of functionality between Ethereum, Fabric, Sawtooth, and Iroha.

	Criteria	Ethereum	Hyperledger Fabric	Hyperledger Sawtooth	Hyperledger Iroha
1-	Supports smart contracts.	✓	✓	✓	✓
2-	Consensus algorithm modularity.	×	✓	✓	×
3-	Built-in components for managing identities.	×	✓	×	✓
4-	Supports payment in fiat currency.	×	✓	✓	✓
5-	Proficient in maintaining different privacy levels between users.	×	✓	✓	✓

3.1 Automation and Contracts

Process automation in the B4CM framework will be delivered by SCs implemented in Turing-complete languages. Ethereum uses Solidity, a new programing language that provides reasonable expressivity but is computationally expensive and to some extent limited in implementing complex contract terms. The remaining blockchain platforms support SC development in more advanced programming languages such as Java, Go, Rust, and C++. The Iroha platform is focused heavily on supporting the development of mobile applications and embedded systems alongside web applications. This platform provides a set of libraries and prebuilt components, including predefined SCs and queries, designed to support the interfacing of IoT-style infrastructure to the distributed ledger platform; this makes Iroha a useful complement to the Fabric and Sawtooth platforms.

3.2 Consensus

Ethereum, as a public blockchain platform, handles the abuse of trust by imposing a proof-of-work (PoW) consensus algorithm which is known to be rigorous but power and time consuming due to the mining process and the need for propagation across the network. The Fabric and Sawtooth platforms by comparision, support several consensus algorithms that

can be changed on the fly while the network is running, making them more easily adaptable to different environments. The use of consensus modularity (as provided by Fabric or Sawtooth) will give the B4CM team the ability to implement a range of concensus mechanisms within the framework to examine and measure the throughputs according to each one. Iroha embraces its own consensus algorithm, a crash fault-tolerant consensus called Yet Another Concensus (aka YAC). In fact, consensus protocols in private blockchain avoid all unnecessary hurdles and complexities since reaching a total agreement on the common truth between predefined identities will be easier and faster.

3.3 Identity Management and Roles

A feature of the Ethereum platform is that it can maintain the anonymity of nodes in the network, enabling them to join or leave without restrictions. This does not serve our purposes in this research however, because it is essential to identify each participant in the network. Hyperledger Fabric provides Membership Service Provider (MSP) and Certificate Authority (CA) services to identify the participant in an easy and manageable way. Sawtooth does not have a CA service similar to the one in Fabric, thus the developer might need to integrate external identity software. Iroha has an intrinsic support for identity management.

In the B4CM framework there is a requirement for the provision of varying privacy levels between users, i.e. not all agreements and payment processes should be available for all network users. Some users may choose to have a private agreement and keep the cost attribution hidden from others who are not involved in that agreement. Ethereum relies on the use of an identical role for all network participants; all transactions are available and visible to all participants in the network. The Hyperledger platforms by comparison have a range of mechanisms by which user roles can be assigned. In Fabric, this issue is managed by creating a separate channel to isolate participants that need private agreements and cost attributions, while in Sawtooth, changing the identity namespace in the transaction family will restrict access to certain identities. In Iroha it is also possible to define access control rules against resources on the chain.

3.4 Resource Management

Ethereum incurs fees (gas) in exchange for every SC execution and has its own native payment currency (Ether) while the Hyperledger platforms Fabric, Sawtooth, and Iroha are cryptocurrency-independent and payment in fiat currencies is available.

3.5 Platform Selection

All in all, a permissioned blockchain network seems to be the best choice to fulfill the design decisions we mentioned in our framework when considering faster settlement, scalable performance, and a more controlled environment. Based on the trade off in Table 1, our proposal will be tested using Hyperledger Fabric as the underlying blockchain platform.

4. System Architecture

Figure 3 depicts high-level abstract of the proposed system architecture. It consists of three kind of main nodes which are as follows:

Organization: A set of organization nodes will build the underlying blockchain network and the maintained network will be running at this component to provide a distributed system. Users' identity management, communication, and consensus process will be built on this component within Peer-to-Peer network. Then, the constructed blockchain network will manage the ledger among all participants, consensus algorithm, and smart contract services to ensure consistency and traceability is handled as well.

Users: Users represent the participants who are willing to communicate with specific organization to consume or provide data. Users will be registered, and their identities should be authenticated and authorized before any access to the blockchain.

Client application: The communication process between users and the underling blockchain network will be through this intermediate node.

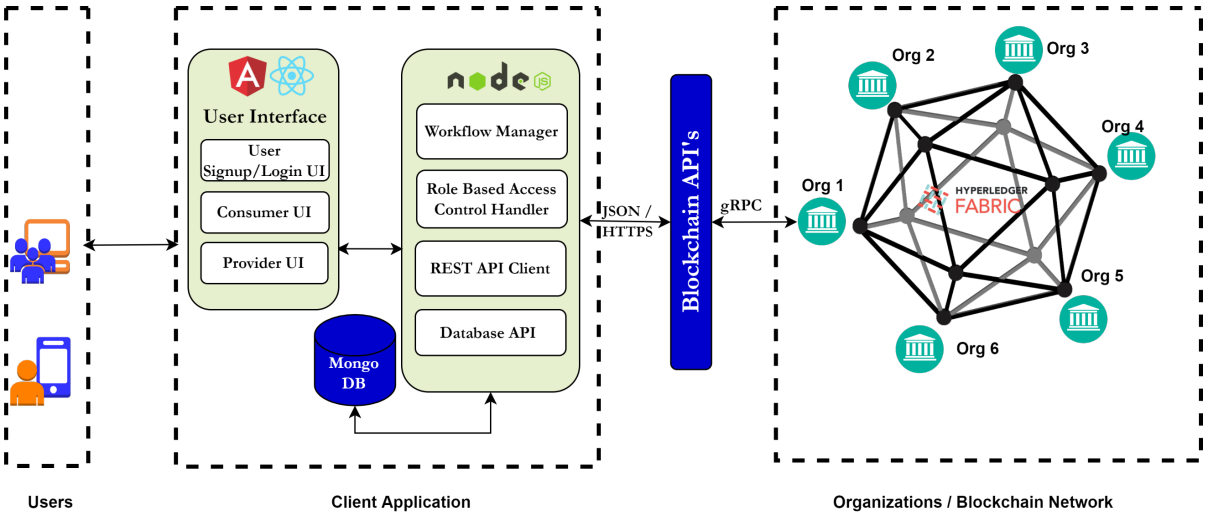

Figure 3: High-level system architecture.

Organization Node

Each organization node in the network has several essential parts which is maintained by the organization or interact with it as follows:

Ledger: Ledger represents the shared database among all participant nodes in the network and it consists of two distinct, though related, databases: a world state and a blockchain. The world state holds set of key-value pairs reflecting the current values of the ledger states according to the validated and committed transactions in the blockchain. While the blockchain is a transaction log which records all the changes that have leaded to the current world state database. Therefore, the data structure in both databases are different and the immutability is guaranteed for the blockchain database but not for the world state database. The world state database helps in making advanced query operations more efficient due to the small-time response needed and the ability to use CouchDB to apply more complex queries. On the other hand, the blockchain uses LevelDB for transaction log which only has create and read access.

Peer: Peer is an entity maintained and monitored by the organization and plays vital role in HLF network as response coordinator to all other components. The Peer node keeps the ledger coordinated across the blockchain network. Connect with the channels, receive all the transactions that are getting broadcasted on that channel. Each Peer could be one or more of the following types:

- Committing Peer: A committing Peer is the one who upon receiving from the ordering service, commits the block into their copy of the blockchain. This block will contain a list of transactions to validate each transaction in the list and confirm such transactions as either valid or invalid and then would commit them to the block. All such transactions, irrespective of whether they are valid or invalid, are committed to the blockchain, and this may be used for audit purposes going forward.
- Endorsing Peer: These are special type of committing Peers who apart from their regular role, will have an additional responsibility to endorse a transaction in the network. Any request coming from the client's node is endorsed by such a Peer. Each of these Peers will generally have a copy of the ledger and the installed Chaincode. The endorsers are entrusted with the responsibility to simulate the transaction and would generate Read / Write sets which are then sent to the requesting client. The transaction is not committed to the ledger during such a simulation.
- Anchor Peer: Generally, a Fabric Network can spread across multiple different organizations and hence there is a need to have Peers to communicate with these multiple organizations. Such a privilege is not available with all the Peers in the network but there are special Peers who only have the authority to do so and these Peers are called Anchor Peers and they are usually defined in channels.
- Leading Peer: Like Committing Peers and Anchor Peers, there are a set of Peers called Leading Peers and are the ones who communicate the messages from ordering service to other Peers in the same organization. The protocol that these Leading Peers use is Gossip and thus all the other Peers in the organization will receives the message seamlessly. One important thing to be noted is that Leading Peers are confined to communicate only within the organization and cannot communicate outside an organization.

Orderer: The Orderer as the name implies will be responsible for ordering transactions into a block. Usually, a separate ordering node does this job which along with other ordering nodes form ordering service cluster. Based on the application design, the organization may have its own ordering service to increase the transaction throughput that will then interact with the ordering service.

Channel: The channel is the connection which ensure the isolation and the confidentiality of data among the consortium participants. Each organization will be registered to one or more channel and the users of that organization will gain the access to the channel their organization belongs to.

Certificate authority: The CA is behind the process of issuing and administrating the certificates which identify the identity and role of each member in the organization. By this entity, unauthorized access is prevented, and consortium remains private.

Smart Contract: In Hyperledger Fabric the smart contract is referred to as Chaincode which holds the business logic and coordinates the interactions between the application and the

ledger. The Chaincode can be written in different programming languages and in our system, we used Golang and Node.js to write and interact with the Chaincode. The Chaincode comprises a set of functions that are designed and coordinated to achieve a specific business logic. It is possible to have more than one Chaincode and accessing the Chaincode will be maintained through the endorsing Peers on which the Chaincode is installed and initiated.

API and Command Line Interface: The APIs are the gateway through which the frontend applications will communicate with the organization in the blockchain network. The REST API server in most cases uses two Node.js modules: the first module is used to define a loopback connection for the blockchain, and the second module is used to set an exposure of the available capabilities over REST. The command line interface is an administrative alternative to the API that can be used to send transactions, invoke Chaincode, and query Chaincode.

All the mentioned parts above are depicted in Figure 4, which shows the architecture of the proposed Hyperledger Fabric network on which the system is built.

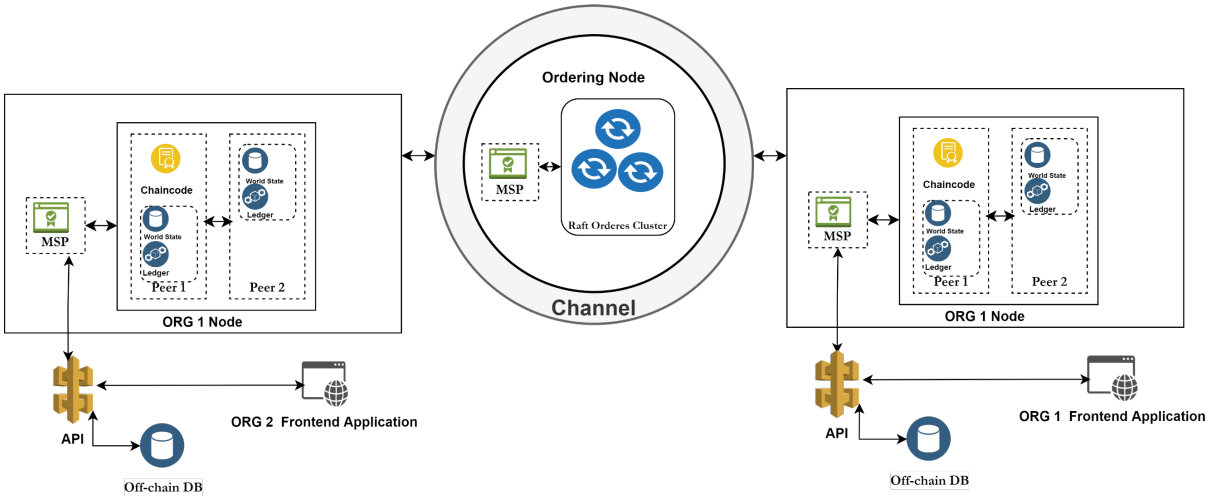

Figure 4: Hyperledger Fabric network architecture.

Client Application Node

This node uses the API REST server to gain access to communicate with the blockchain network by submitting transactions. The API server will in turn processes all requests and communicate directly with the blockchain network to invoke the appropriate Chaincode. Therefore, authentication, authorization, and access policy need to be managed before any transaction submission. Local authentication and authorization for each organization can carried on, consequently, registered participants can sign transaction using their issued private keys.

The access of any client node across the network will be granted by connecting to a Peer for each transaction submission. Then, client node will be able to invoke the functions provided through Chaincode, perform related transactions, and execute the proposal request and responses related to the Orderer's transactions. In addition, when there is usage of external storage, client node needs to calculate the checksums stored in the ledger as well as storing data in external storage.

From this perspective, the system can be split as shown in Figure 5 into the following layers:

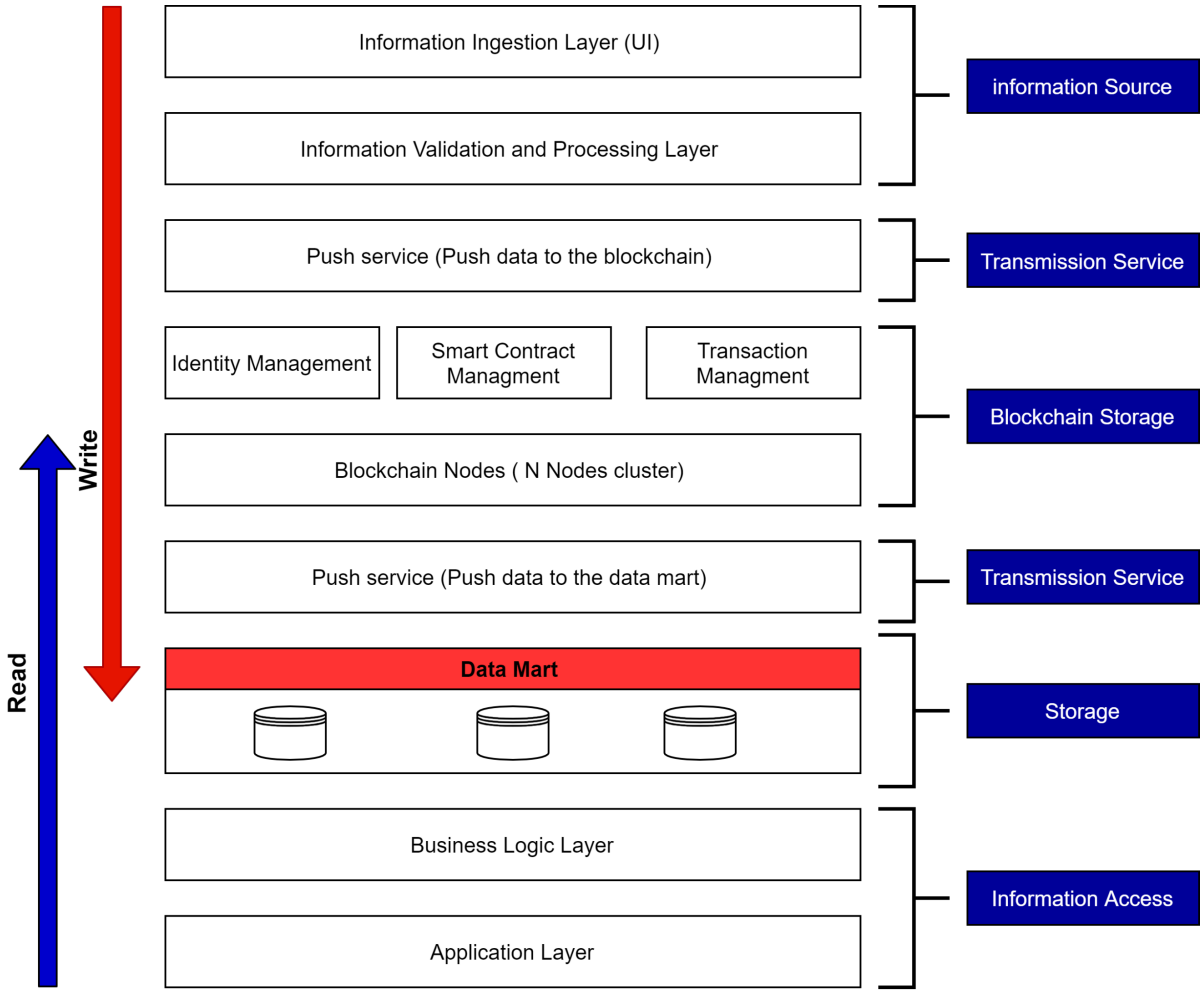

Figure 5: Logical structure of network layers.

Information Ingestion Layer (IIL): Information Ingestion Layer is the UI layer that is responsible for ingesting the data on the other service. This layer is the entry point of the data into the application.

Information Validation and Processing Layer (IVPL): The Information Validation layer validates the data which it receives from the IIL. Its main task is to ensure that input received is clean and is in the correct format. The processing layer processes the incoming data, sanitizing the data and route it to the push services.

Push Service: Push services deal with sending the data to the blockchain nodes. It essentially routes the data received from IVPL and then sends data to the blockchain. Push service is using Fabric SDKs for smart contract management and transaction management to send data to the blockchain. It is also leveraging the Fabric CA SDK for identity management.

Blockchain Nodes: It is a Hyperledger Fabric Blockchain network, which is private and permissioned in nature. This blockchain network uses RAFT consensus which is Crash Fault-Tolerant and is capable of handling crashes.

Data Mart: Data Mart is the cluster of databases that are acting as off-chain DB in the whole system. It is a cluster of MongoDB which is a document-based NoSQL database. The database cluster is used to achieve the High Availability (HA) and to remove the single point of failure risk.

Business Logic Layer: The business logic layer is responsible for handling the application logic, it deals with the retrieval, processing, transforming, and managing the application data.

Application Layer: Application Layer directly interacts with the application over the HTTP Protocols. It uses REST APIs to interact with the web services.

5. High-level Description of Software Components, Operational Context and Workflow

In this section, the network deployment, workflow, Chaincode model, and transaction models will be illustrated and discussed.

5.1 Technology Stack

The application is built on the following technology stack:

Hyperledger Fabric v2.2: The latest stable version of Hyperledger Fabric platform to the time of writing this report.

Node.js: Cross-platform JavaScript run-time environment that executes Server-Side JavaScript code.

Docker: Orchestration engine that performs operating-system-level virtualization. The docker container engine will be used to run multiple containers for data storage and service operations

Angular: A declarative, efficient, and flexible JavaScript library for building user interfaces.

MongoDB: Off-Chain Database for storing the user credentials and raw data.

The deployed stack provided in the GitHub repository runs on virtual desktop nodes each based on Ubuntu 20.04. The server architecture in our system uses docker containers to easily build and run the application on local machine or remote machine. At the end of deployment each server's docker engine will accommodate several images for Peers, Orderers, Certificate Authorities that provide membership service, and installed Chaincode containers. As depicted in Figure 4, the developed blockchain network's topology consists of:

- Two Organizations
- Four Committing Peers, two for each Organization
- Two Endorsing Peers, one for each Organization
- Three Orderers, each in separate container and all will be using Raft consensus
- One dedicated Channel
- LevelDB is used for the ledger's logs and CouchDB for the world's state on each Peer

The complete code for the demonstration is available from GitHub along with deployment instructions for a clean installation of Ubuntu 20.04. You can obtain the current release of the code at any time using the command line:

git clone https://github.com/B4CMProject/B4CMProjectSoftwareReleases

The demonstration of the deployed model is discussed in detail in section 7.

5.2 Intended Operational Context

Operationally, it is of course intended that the framework should be deployed within an appropriate operational context for the European railways. For the purposes of this study, the team have chosen to use the current operational structure of the railways in the UK to provide that context, although these are of course currently in flux as the industry prepares for the introduction of Great British Railways over the next 5 years. Assuming the current operational model, then practically speaking it is safe to assume that the setup and ongoing

administration of the network would fall under the responsibility of Department for Transport (DfT), that has the highest-level administrative authority in the GB railway industry, as shown in Figure 6. DfT establishes the UK rail network's strategic directions and collaborates with a variety of partners to conduct and fund all main projects.

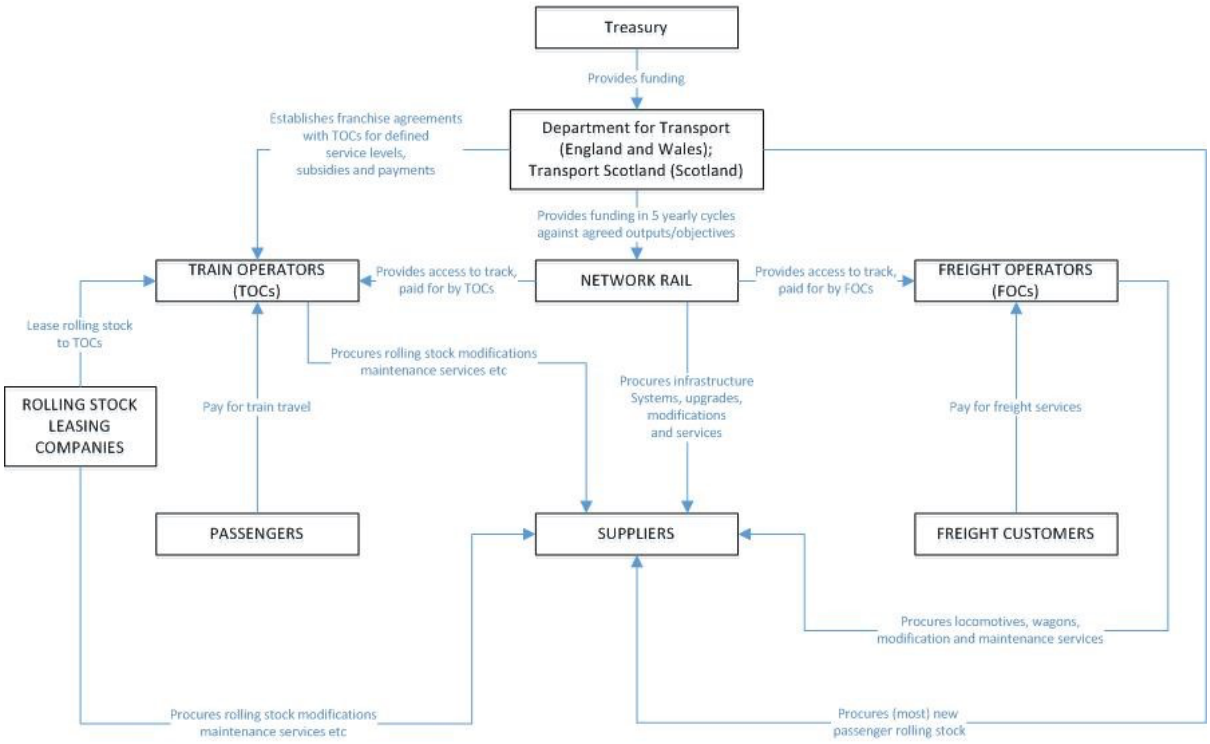

Figure 6: High-level structure of the GB rail industry.

5.3 Transactional workflow and Determination of Consensus

The client nodes are considered to create and submit transactions to invoke the related operations which the system behaviours depend on them. The transaction flow process in Hyperledger Fabric is illustrated in the following steps that adapted from similar illustrations in [3] and [9].

Step 1: Through the client node, a user in the member organization signs and sends transaction request as proposal. The proposal contains the procedure of a Chaincode that the client intends to invoke.

Step 2: The proposal will be broadcasted via the client application to all endorsing Peers which are already defined earlier in the endorsement policy for each Chaincode.

Step 3: Each endorsing Peer will authenticate the user certificate using the MSP to verify the sender signature before validating the transaction. Then, the transaction will be processed and via running the Chaincode installed on each endorsing Peer.

Step 4: The result of processing the transaction will be signed by the endorsing Peer along with the proposal response which contains a transaction approval or transaction rejection and will be sent back to the client application.

Step 5: On receiving sufficient number of approved proposals as defined in the endorsement policy, the client sends a transaction containing endorsed transaction

proposal responses to the Ordering service to order and place the transaction into a block, along with other transactions received from any clients.

Step 6: The Orderer will send the new generated blocks to the Anchor Peers of each member organizations within the same channel.

Step 7: Once the block is acquired, the Anchor Peer will propagate it to other Peers via gossip protocol to validate each transaction in the block following the same order to ensure that it has been consistently endorsed by all relevant organizations. Then, the transaction is committed and the ledger is updated accordingly. If there is detected inconsistency or transaction failure, the transaction is retained for audit but not applied to the ledger and their effects are discarded.

Step 8: Finally, when the transaction is committed and the ledger is updated, an event notification is emitted.

As revealed above, the Hyperledger Fabric adopts execute-order-validate paradigm in handling transactions. This allows parallel transactions execution and transactions recording which enhances the efficiency when maintaining the consensus to increase the overall throughput. When each transaction is signed by the private key of the user who invokes that transaction, part of the endorsement process of the endorsing peers is to verify this signature using the public key of the issuer before considering the transaction as valid one. This process will enforce an end-to-end security between the user and the blockchain to insure data attribution to the correct stakeholder.

6. Chaincode Transaction Model

The proposed Chaincode that holds the business logic contains assets, attributes and set of functions to manage the different transactions. For the assets, models are created with all the needed attributes to describe each asset as depicted in Figure 7. The project team note that several of these fields could be populated via controlled vocabularies, and this will be investigate later in the B4CM project. Specialised ontologies for rail have been presented in the literature and would prove highly useful in this context, examples include the RaCoOn ontology [12] developed previously by members of the project team, and work by the Shift2Rail funded LINX4RAIL project.

The relations between the assets are depicted in Figure 8 which shows also how the link to the external storage is created by including the filename inside the *DataHash* record along with the digested hash value of the raw data. The proposed accounting model and the need for the use of escrow to overcome issues around fraudulent transactions not otherwise managed in traditional payment models are discussed in detail in [10] (authored by the project team) and are also presented in B4CM deliverable 2.1. Examples of fraudulent transactions include the introduction of deliberate delays into the delivery of data to consumers by producers post-agreement, the delivery of data not matching the original advertisement,

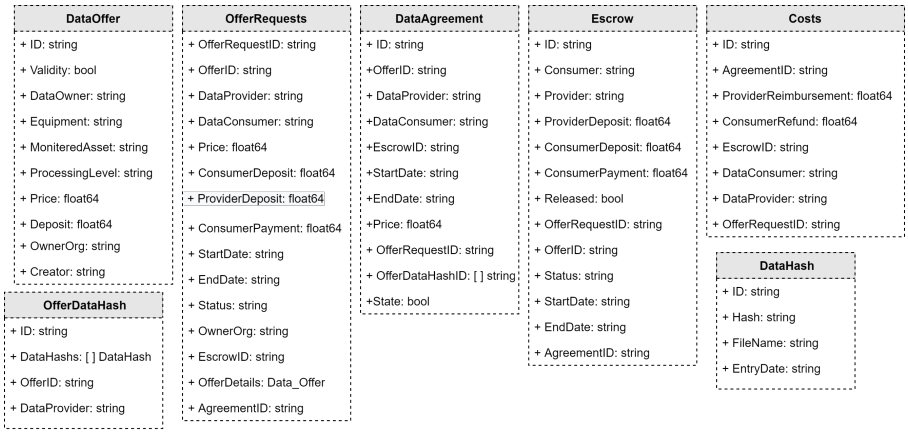

Figure 7: Entities as used by the chaincode model.

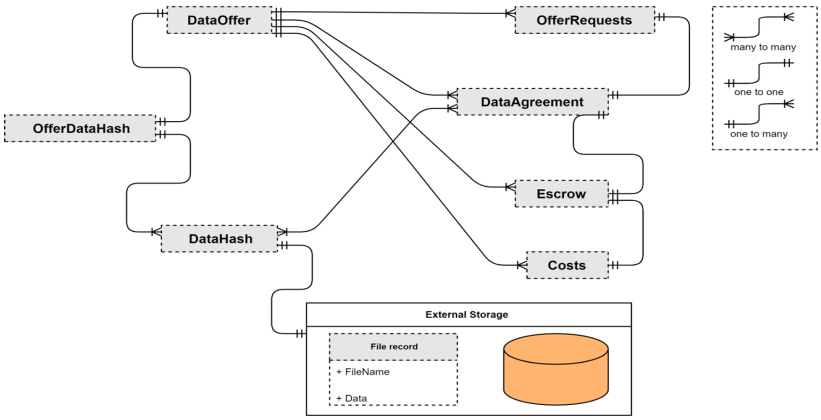

Figure 8: Relationships between chaincode entities and how it is linked to the external storage.

Most of the transactions will be initiated via the browser by sending HTTP/HTTPS request to the web server. Then, the browser sends the requests in the JSON format, and the web server routes all those requests to the application server. The application server is running service which is built on nodeJS and is responsible for processing and sending those requests to Hyperledger fabric blockchain with the help of Hyperledger Fabric SDK and the connection profiles. An Event emitter is put in place which is receiving the events emitted by the block and forwarding those events in the form of JSON to the application server. The application server is receiving those events and then persisting those events in the off-chain storage for logging purposes and to the browser to show requests responses. The application server is also interacting with the off-chain DB to access the user credentials and file storage and retrieval. Figure 9 shows the general flow of requests processing.

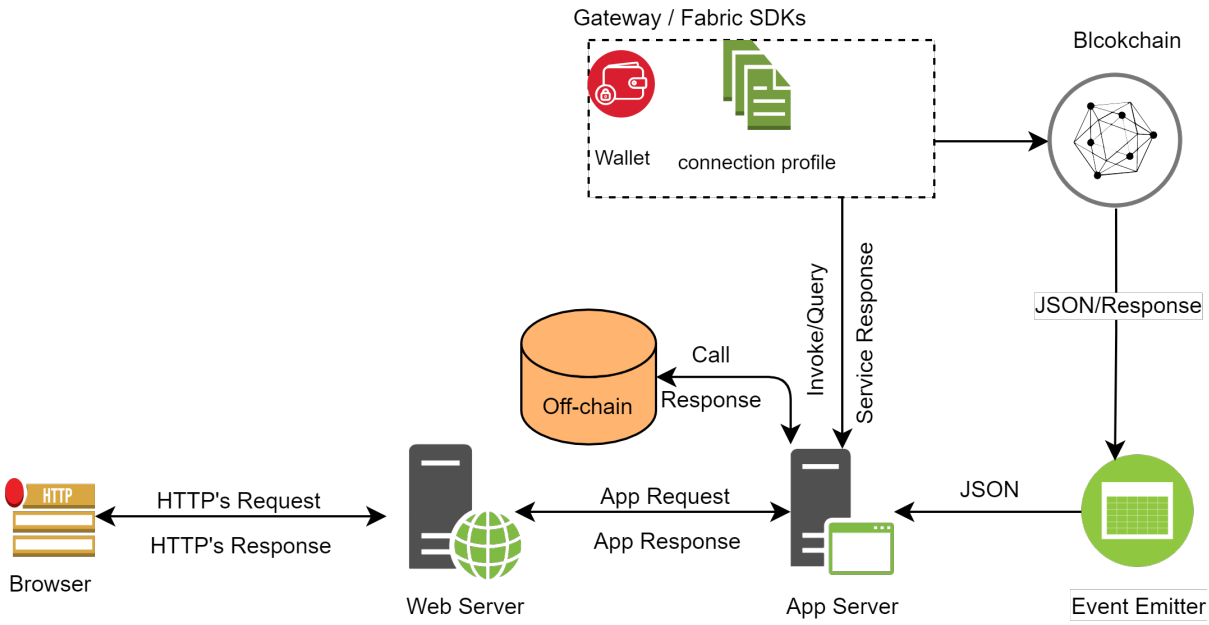

Figure 9: Transactional flow through the system.

Since the storage of data at rest is the responsibility of the provider, within the B4CM framework we are proposing that shared data is passed to the shared storage in an unencrypted state. In [11] the authors have provided a framework for the sharing of industrial data on public clouds where that data is encrypted. This would be a recommended evolution of future releases of the platform, and the work is structured such that it can be easily adopted by any blockchain platform useing Elliptic Curve Cryptography (ECC) for the digitally signing of transactions (including the Fabric platform on which B4CM is based).

6.1 Transaction Registration

Users should register and obtain private / public pair to gain access to the system services. Many restrictions according to each organization policies could be added at this level to restrict unauthorized users to register. One way is to dedicate the registration process to the organization's admin to maintain a list of eligible users which could be modified periodically when new users are enrolling to the organization or leaving the organization. The admin in each organization will be responsible of communicating with providers and consumers, investigating their identities, and issuing their certificates and keys.

For testing purposes, a sign-up form is implemented to register the users and to issue the key pair for each user but in future, for production phase, this process will be assigned to the admin identity in each organization.

6.2 Use of Transactions by Providers

The provider, through his interface, will be able to perform the following transactions as shown in Figure 10 with the same numbering order:

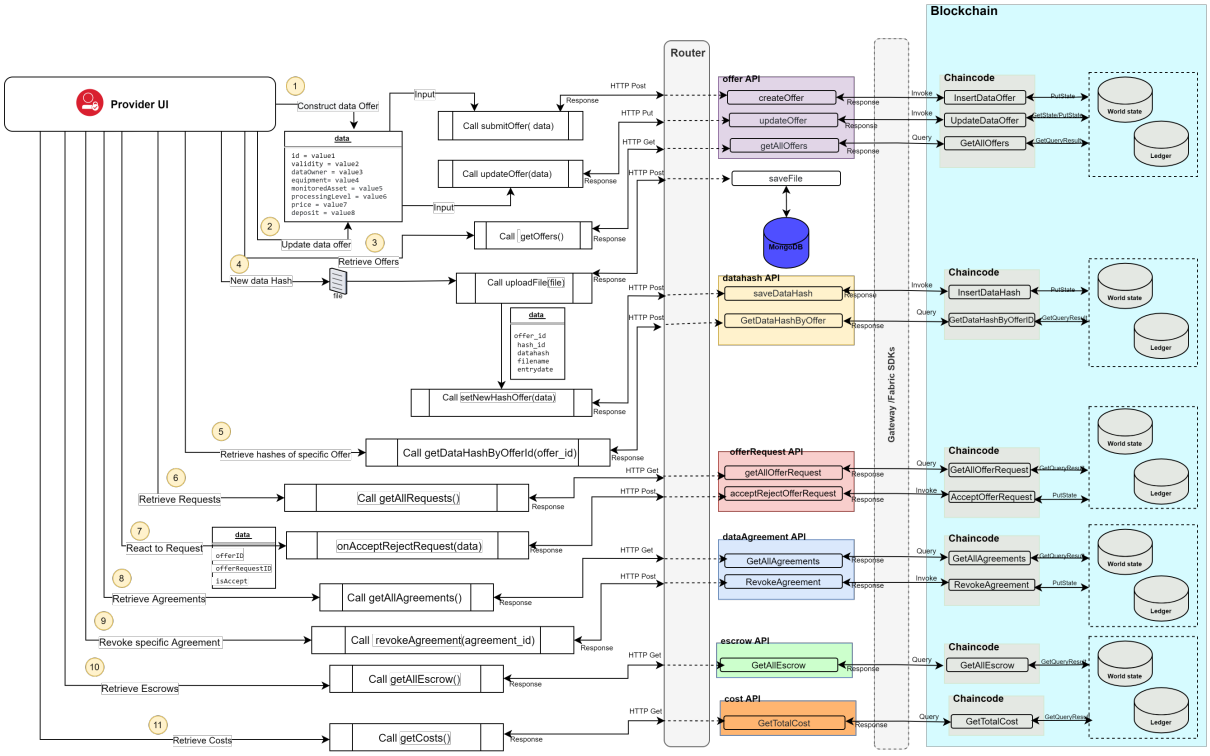

Figure 10: Provider-side transactional flow.

Step 1: Construct Data Offer is an essential transaction that simulates advertising data offers which the provider has to the consortium network. The provider will initiate this process by setting all the needed attributes and then pushing the offer to the blockchain using the Chaincode function *InsertDataOffer*.

Step 2: Update data offer will enable the provider to update any offer attribute at any time without affecting the ongoing agreements of the same offer. The updated attributes will take effects with new requests. The offer record will be updated in the blockchain through the Chaincode function *UpdateDataOffer*.

Step 3: Retrieve offers is the process by which the provider will access all his offers. The function *GetAllOffers* in the Chaincode will process this transaction.

Step 4: New data hash process simulates uploading the raw data to the external storage (MongoDB) and uploading the hash value of the same data to the blockchain accordingly. This will be accomplished by adding the new hash value to all *DataAgreement* records that are not expired and not revoked for the same *offer_id*. In addition, the new hash value will be appended to the *OfferDataHash* record which hold all the hash values of specific *DataOffer* record, see Figure 11.

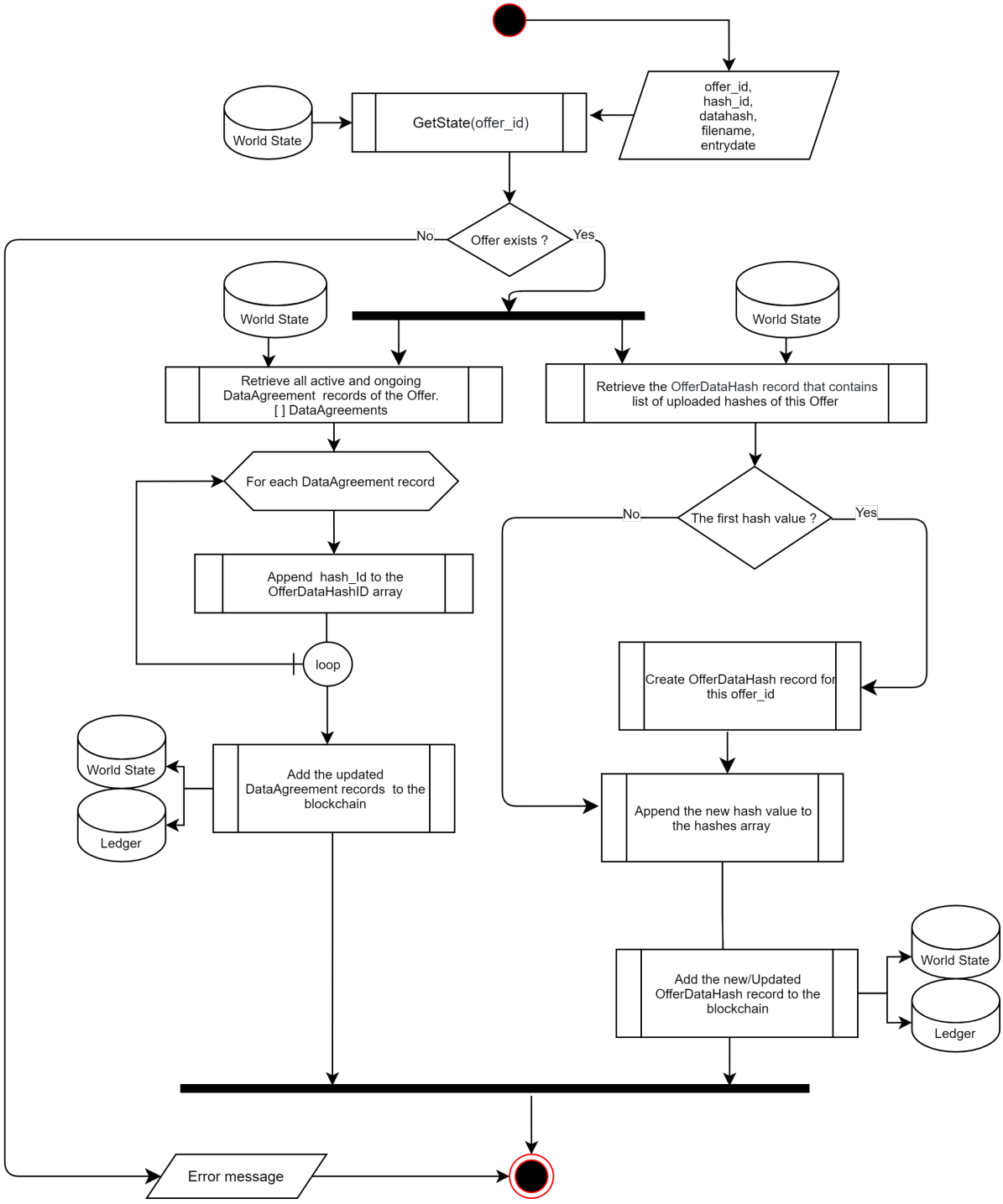

Figure 11: Flowchart illustrating the insertion of new hashes.

Step 5: Retrieving the hash values of specific offer will be accomplished by querying the *OfferDataHash* records and retrieving record that matches the *offer_id* and which contains all uploaded hashes for the same offer.

Step 6: Retrieving requests will show all the requests that has been sent from consumers to the provider and stored in *OfferRequest* records, then, the provider will be able to react against these requests.

Step 7: React to Requests simulates the case of capturing the provider responsiveness of the received requests. As a result, this will trigger the transactions of updating the *Escrow* and *OfferRequest* records accordingly based on the provided details and payments. In Figure 12, the logical flow of accepting and rejecting request transaction as implemented in the Chaincode function *AcceptOfferRequest* is illustrated.

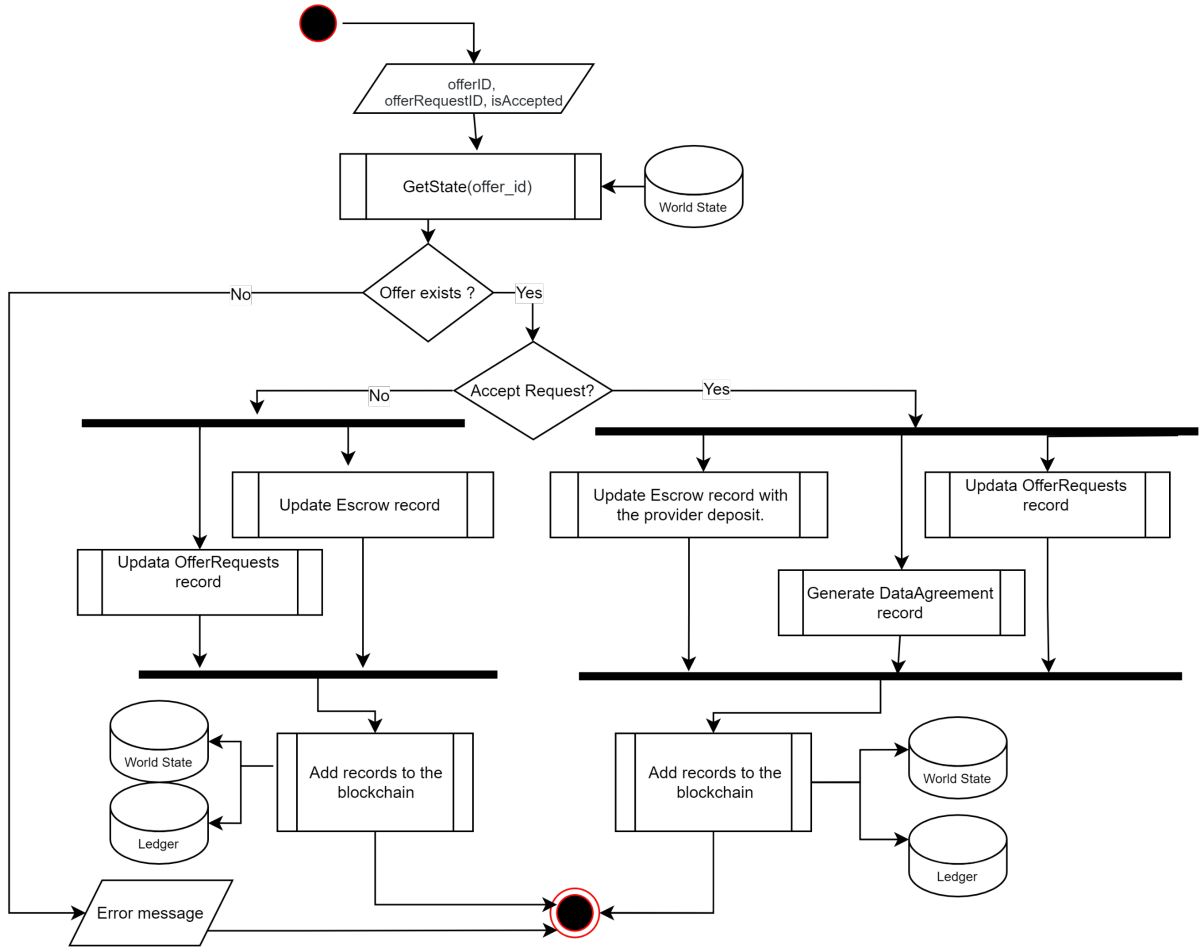

Figure 12: Flowchart illustrating the response to offers.

Step 8: Retrieve Agreements enables the provider to explore all his settled agreements and this process will be accomplished through querying *DataAgreement* records in the Chaincode function *GetAllAgreements*.

Step 9: Revoke specific agreement will lead to releasing the *Escrow* record and generating the cost attribution of the agreement, see Figure 13. Accordingly, *DataAgreement*, *Escrow*, and *Costs* records will be updated and appended to the blockchain in the Chaincode function *RevokeAgreement*.

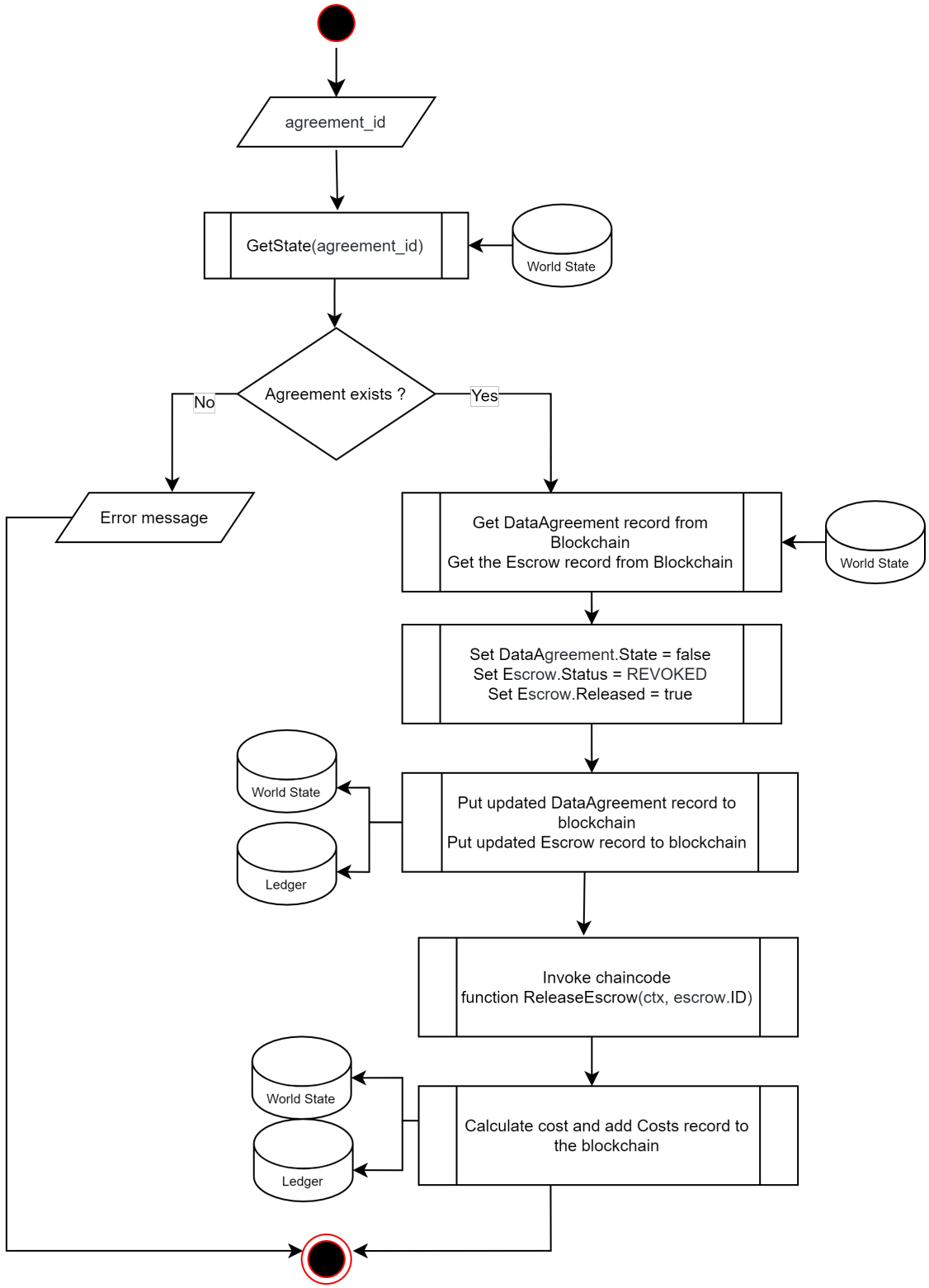

Figure 13: Flowchart illustrating the revocation of an agreement.

Step 10: Retrieve Escrows enables the provider to see all active escrows through querying *Escrow* records in the Chaincode function *GetAllEscrow*.

Step 11: Retrieve costs showing the costs distribution based on the released escrows through querying *Costs* records in the Chaincode function *GetTotalCost*.

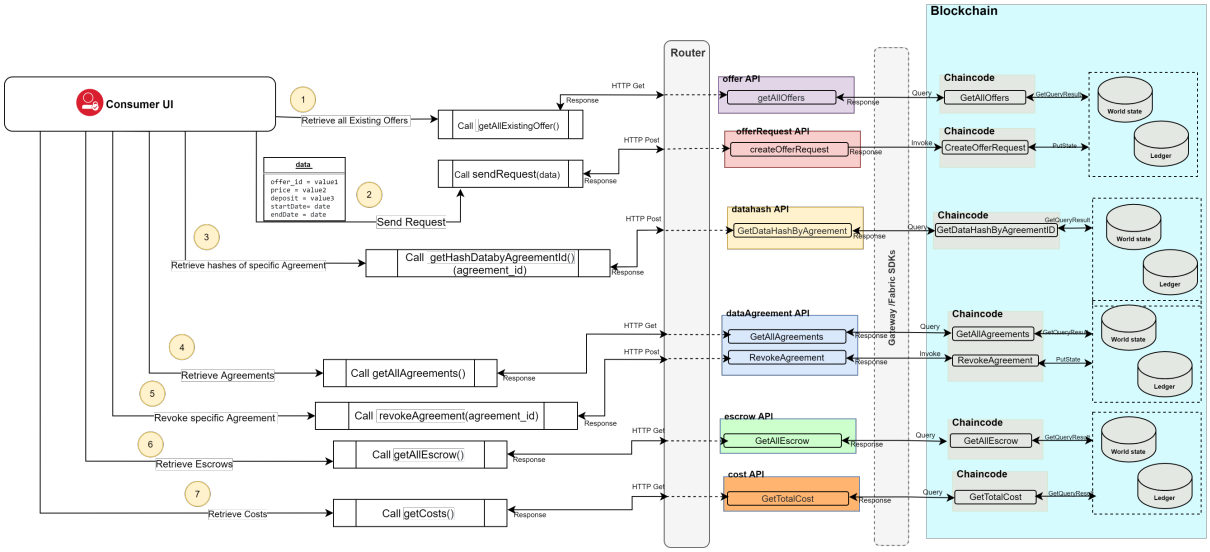

Figure 14: Consumer-side transactional flow.

6.3 Use of Transactions by Consumers

The consumer, through his interface, will be able to perform the following transactions as shown in Figure 14 with the same numbering order:

Step 1: Retrieve all Existing offers will send a query that will be processed by retrieving all the offers from the world state database by the Chaincode function *GetAllOffers*. The result of this query will list all available offers in the blockchain and the response will be reflected to the consumers portal and each consumer has the ability then to request any available offer.

Step 2: Send Offer Request is representing the first step in building an agreement by sending a request to the provider in which the consumer specifies the duration of the agreement and place the data price and the deposit in Escrow. In the Chaincode, the function *CreateOfferRequest* will use the sent attribute *offer_id* to retrieve the offer details and check its existence before generating new *Escrow* and new *OfferRequest* records then append them to the blockchain (see Figure 15).

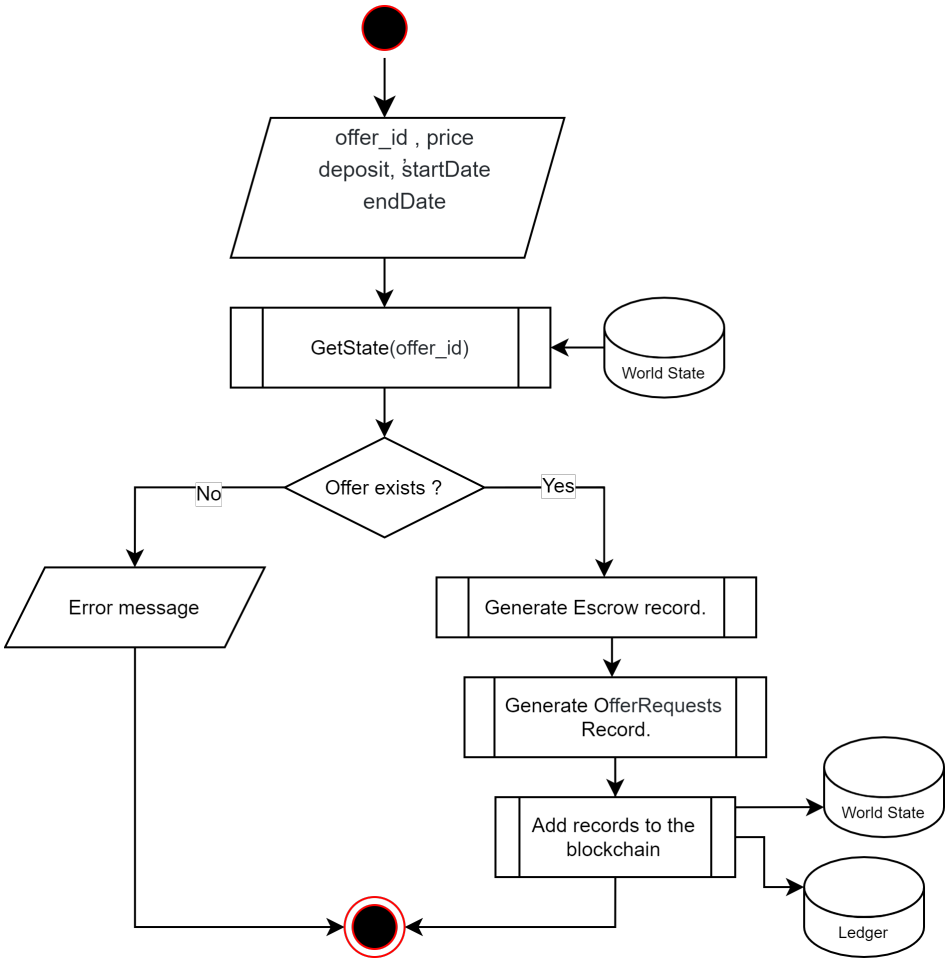

Figure 15: Flowchart illustrating the generation of an offer.

Step 3: Retrieve the hash value array of each agreement which is updated with every new hash entry when the provider uploads new data to the system as illustrated above in **Error! Reference source not found.**. The function *GetDataHashByAgreementID* in chaincode will access the hashes IDs stored in *OfferDataHashID* attribute of *DataAgreement* record.

Step 4: Retrieve all agreements the consumer has by applying a query on *DataAgreement* records in the Chaincode function *GetAllAgreements*.

Step 5: Revoke agreement transaction will terminate the agreement before reaching the agreed end date. the escrow will be released, and the final costs attribution will be calculated in similar way to the revoking transaction discussed in the provider's transactions. see Figure 13.

Step 6: Retrieve escrows will show the escrows of all agreements by applying a query on *Escrow* records in the Chaincode function *GetAllEscrow*.

Step 7: Retrieve all Costs will be showing the costs distribution based on the released escrows through querying *Costs* records in the Chaincode function *GetTotalCost*.

6.4 Use of Transactions by Automation

As noticed above there are some dependencies in transactions invocations where some transactions are invoked based on the responses of other transactions invocation or based on external events. The following transactions are invoked by other transactions or scheduled to be invoked at specified date and time.

Step 1: Create Escrow transaction is invoked twice. The first time is when the consumer sends a request to the provider to hold his payment and invokes the function *CreateOfferRequest* in Chaincode as shown in Figure 15. The second time is when the provider responds to the request by invoking the function *AcceptOfferRequest* to update the Escrow record with the provider deposit in the case of acceptance or to set his deposit to 0 in case of rejection, see Figure 12.

Step 2: Create Agreement as shown in Figure 12, this transaction will be invoked only when the provider sends an Accept response to the *AcceptOfferRequest* function.

Step 3: Agreement expiration transaction is implemented as scheduled job that emits an expiration event when the end date is due to invoke the function *ReleaseEscrow* in the Chaincode.

Step 4: Release Escrow as discussed before will be triggered automatically when the agreement's end date is due or when the agreement is revoked by provider or consumer at any time. Releasing the escrow will evaluate the presence of latency or falsified data claims before calling the *InsertCost* function as illustrated in Figure 17.

Step 5: Create costs transaction will be invoked when the escrow is released to find the cost attribution by invoking *InsertCosts* function. The payments to provider and consumer will be calculated as illustrated in Figure 18 and Figure 16.

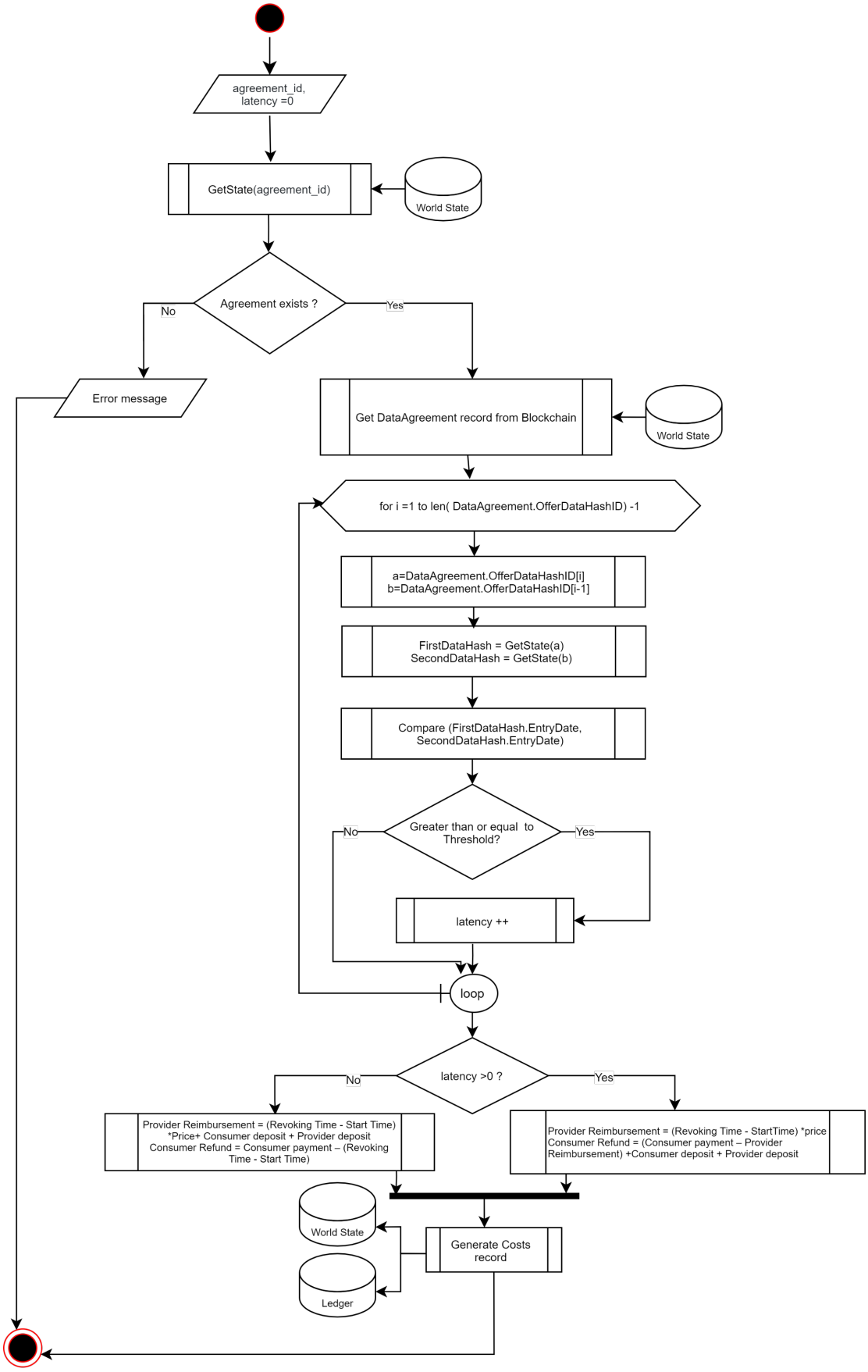

Figure 16: Flowchart illustrating cost calculation and the impact of latency.

Algorithm 1: Conducting releaseEscrow function.

Result: latency, falsified
initialization:
Index $i \in \{a..b\}$ of hash records in ledger for that particular agreement between StartDate and (endDate or revokingTime);
Index $j \in \{x..y\}$ of data records in the shared storage for that particular agreement between StartDate and (endDate or revokingTime);
latency=0, falsified=0;
if *consumer requests latency verification* **then**
  for $i \leftarrow a$ *to* b **do**
    Compare timestape of the hash record in the ledger with index i;
    if *record[i].EntryDate - record [i-1].EntryDate* $\geq$ *threshold* **then**
      latency ++
    end
  end
end
if *consumer requests falsified verification* **then**
  for $i \leftarrow a$ *to* b *and* $j \leftarrow x$ *to* y **do**
    Compare data of each hash record in the ledger with the hash value of data in the shared storage;
    if *record[i].data* $\neq$ *hash (record[j].data)* **then**
      falsified ++
    end
  end
end

Figure 17: Release of Escrow.

Algorithm 2: Costs Attribution.

Result: providerReimbursement, consumerRefund
if *latency* $>$ *0* $\vee$ *falsified* $>$ *0* **then**
  providerReimbursement = (revokingTime-startTime) *price;
  consumerRefund = (Cpayment – providerReimbursement) +Cdeposit + Pdeposit;
else
  if *revokingTime is initiated by provider* **then**
    providerReimbursement = (revokingTime-startTime) *price;
    consumerRefund = (Cpayment – providerReimbursement) +Cdeposit + Pdeposit;
  else
    if *revokingTime is initiated by consumer* $\wedge$ *latency=0* $\wedge$ *falsified=0* **then**
      providerReimbursement = (revokingTime-startTime) *price+ Cdeposit + Pdeposit;
      consumerRefund = Cpayment – (revokingTime-startTime) *price;
    else
      providerReimbursement = Cpayment + Pdeposit;
      consumerRefund = Cdeposit ;
    end
  end
end

Figure 18: Attribution of costs.

7. Demonstration Application

As a demonstration of the functionality of the proposed application, a skeleton web app has been developed using Angular 12.0.1 to interact with the written REST APIs. The Chaincode is written in Golang and use the standard Fabric SDKs (Fabric-Network and Fabric-ca). The implementation of Chaincode, APIs, and simulation are available as open source in GitHub at *https://github.com/B4CMProject/B4CMProjectSoftwareReleases with documentation available in the "Documents" subfolder.* As a brief introductory guide to the structure of the code, some notes are included below.

The top-level structure of the project is broken down into three main sections, "api-server", "Front-end", and "network", with the latter being the most important of these as it contains the basis for the local blockchain deployment used by the demonstration. "Front-end" contains the angular interfaces shown in the figures throughout the remainder of this section, while "api-server" provides the javascript APIs used for communication between the interface components and chaincode assets.

Under the "network" channels are then used to implement peer organisations and orderer organisations, with 3 of each being deployed for the demonstration in a process scripted using YAML files. Each of the peer organisations within the network is also provided with an instance of a CouchDB database deployed (again via the YAML script) using docker. The overall structure is shown in Figure 19.

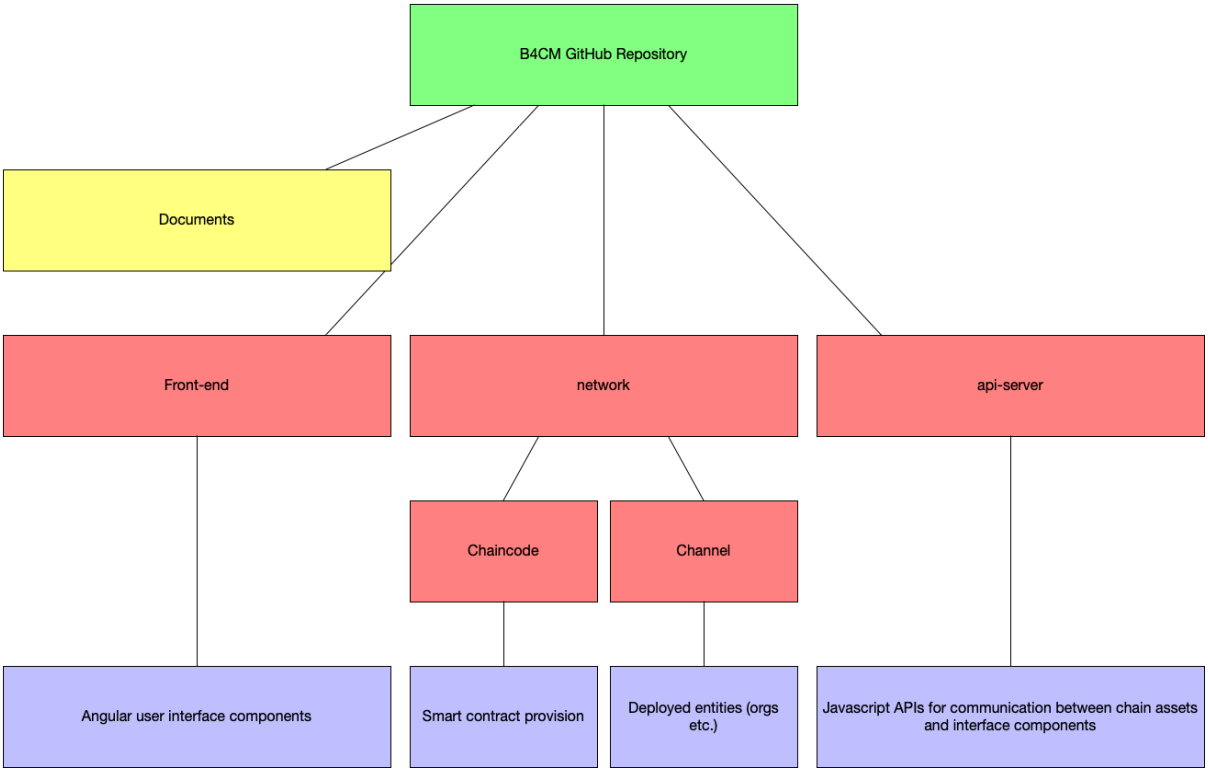

Figure 19: Structure of the code in the project repository.

Process 1 – Provider and Consumer Enrolment: In production deployment, this process should be under the responsibilities of the organization admin as mentioned before in Section 4. For testing purpose, this process is simulated by a sign-up form to receive the enrolment requests. Then, enrolled participants will login to the application to perform all processes through the Login form as shown in Figure 20.

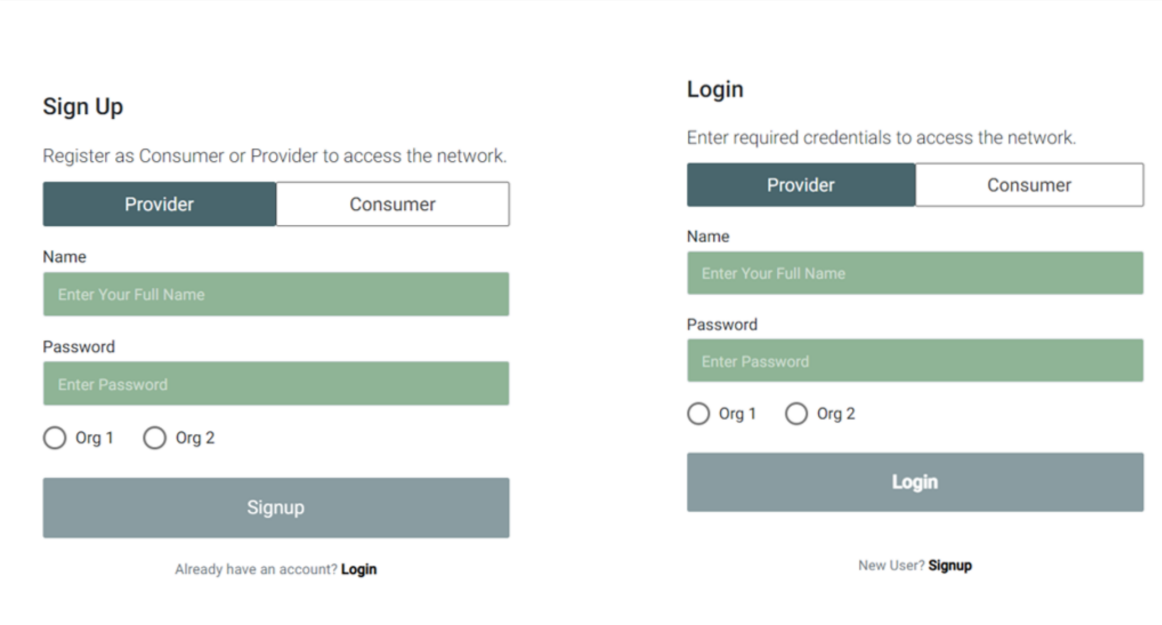

Figure 20: Enrolment process.

In our simulation, providers and consumers might belong to Org1 or Org2 to trade data within the same organization or with different organizations in the same network. The distribution of providers and consumers among organizations is flexible and could be redesigned depending on the application domain in which this framework is integrated.

For testing, we have created three different providers: Siemens, and Virgin trains that are members in Org1 while Network Rail is a member in Org2. Created consumers are: South West Trains, First Great Western, Network Rail, and Serco and all are members in Org2.

Process 2 – Creation of Data Offer: This process simulates advertising data offers to the network and the provider will initiate this process by setting all the needed attributes and pushing offers to the blockchain. As shown in Figure 21 (b) and (c), providers (e.g., Siemens) will generate and update offers through their portal and each consumer (e.g., First Great Western), in Figure 21 (a) has the ability to see all offers and request any available one.

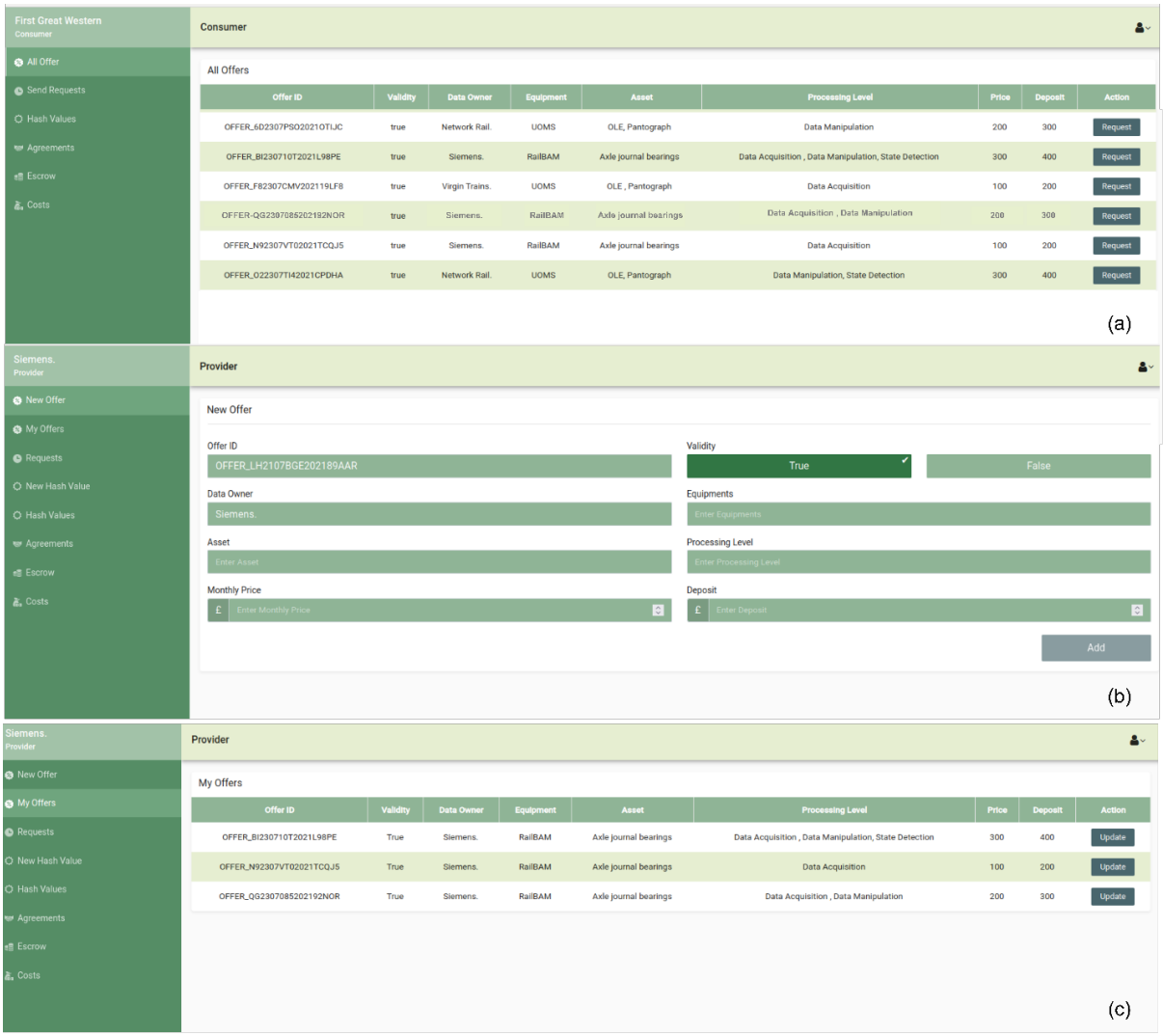

Figure 21: Creation of data offers from available sets.

Process 3 – Upload of Data: This process simulates uploading the raw data to the external storage and uploading the hash value of the same data to the blockchain. As shown in Figure 22, when the provider (e.g. Network Rail) uploads the data, the hash value will be generated consequently. Therefore, uploading the same data will generate the same hash value which will be taken as evidence when consumers raise a claim in future. The consumer on the other side (e.g., Serco), will be able to access only the raw data and hashes related to specific agreement as illustrated in Figure 22 (d).

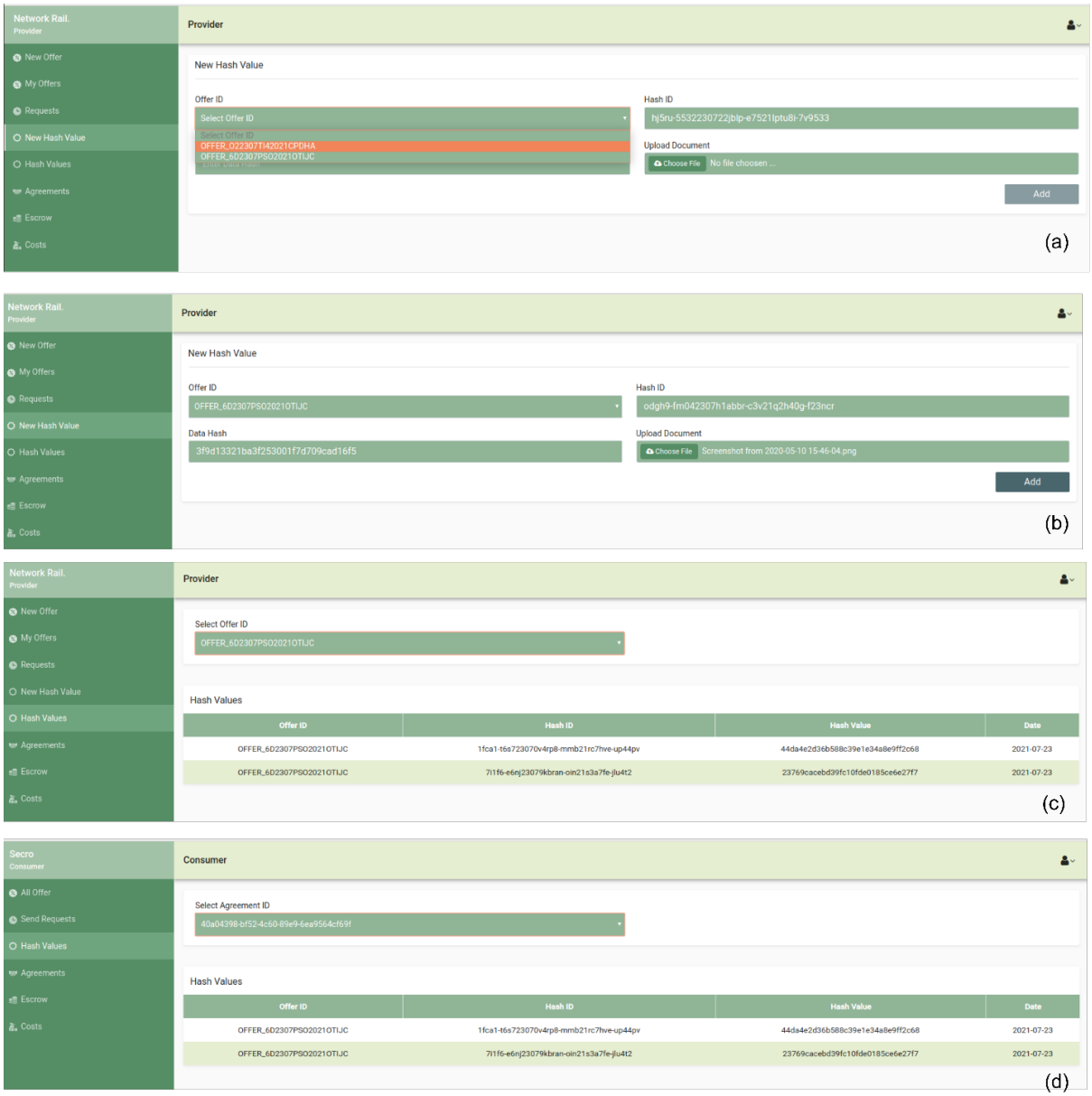

Figure 22: Upload of data & assignment of hashes.

Process 4 – Requesting Data: This process simulates the case of sending a request to the provider to start data trading and will be initiated by consumer. In Figure 23 (a) and (b), Serco and First Great Western as consumers are sending request to Network Rail and Siemens respectively after specifying the duration and doing their part of payments.

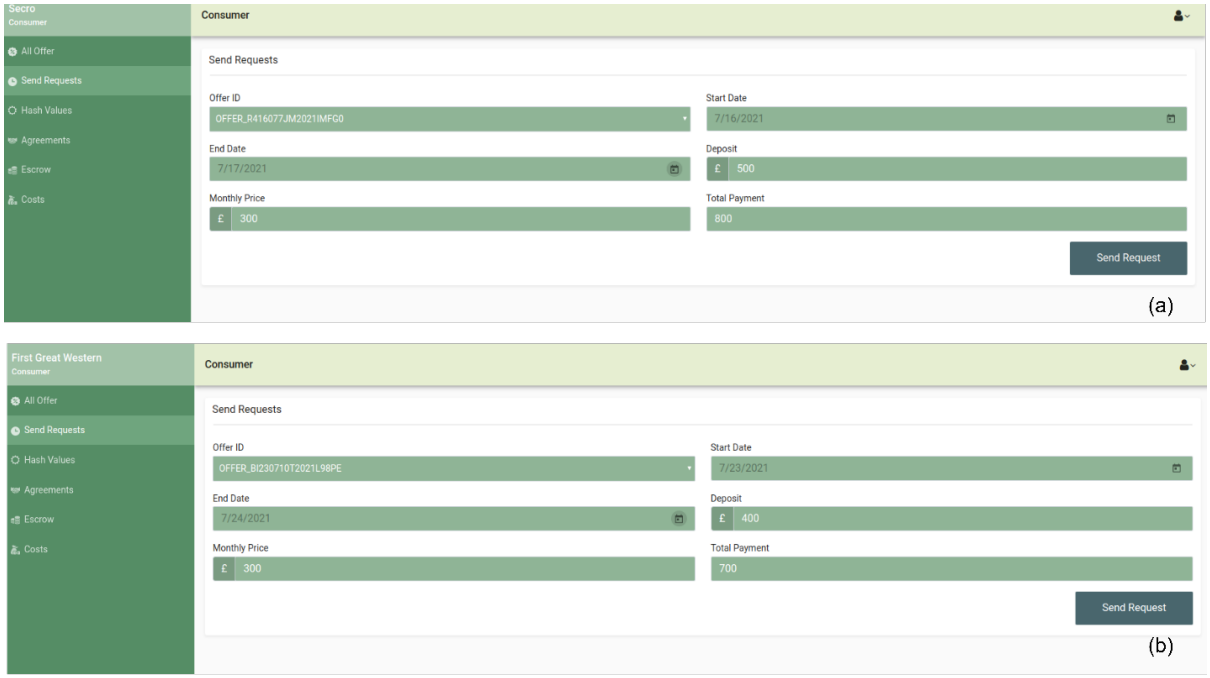

Figure 23: Issue of data requests to providers.

Process 5 – Issuing of Response to Data Request: This process simulates the case of responding to the requests received from consumers. In Figure 24, Siemens and Network Rail are maintaining the received requests by deciding which one will be accepted or rejected based on the provided details and payments.

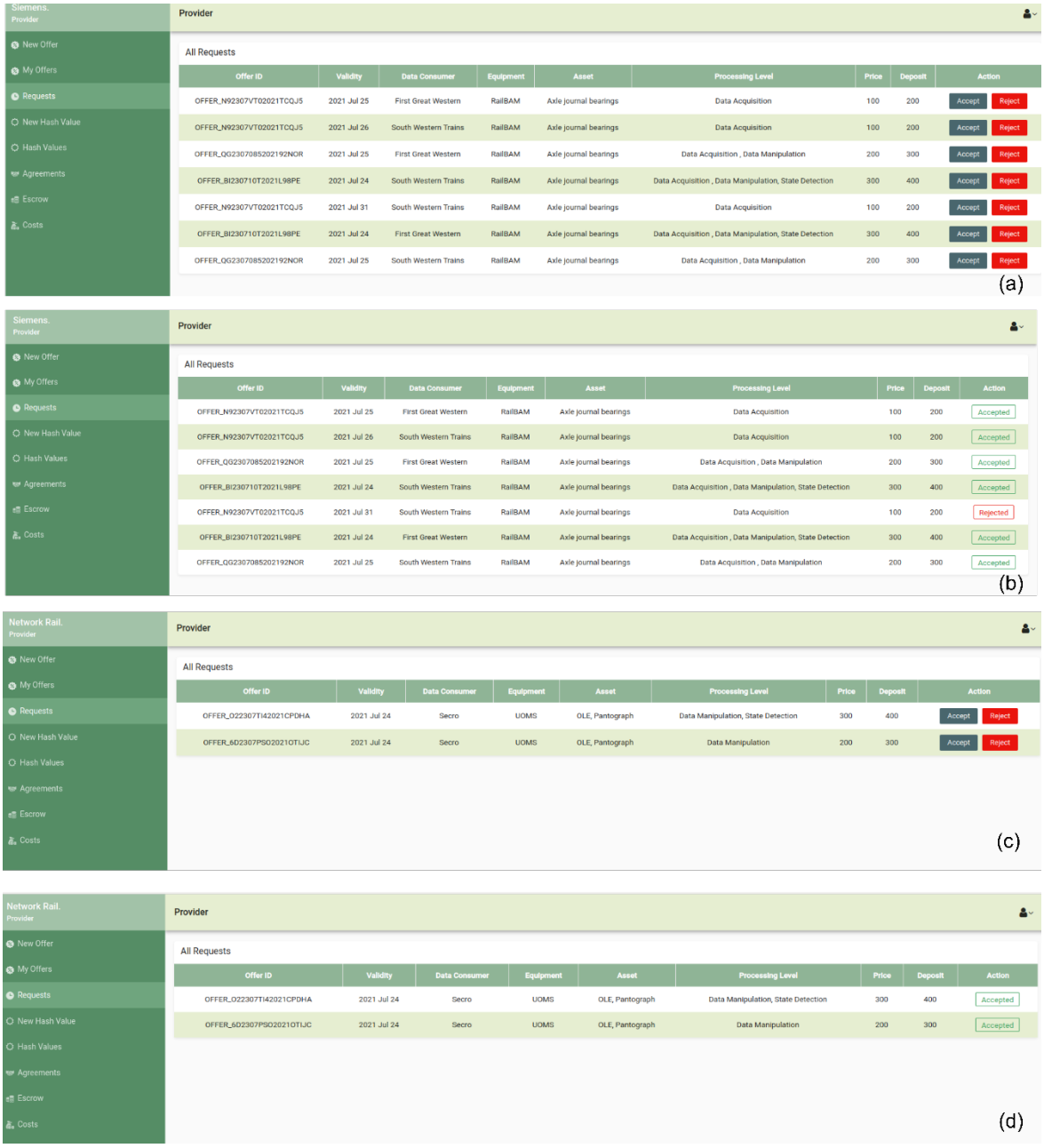

Figure 24: Management of requests.

Process 6 – Creating Agreements and Entering Escrow: This process simulates the case of generating the agreement and the escrow records automatically. When the request is accepted by provider, the escrow will be locked holding all payments and the agreement will be activated between provider and consumer. In Figure 25 (a) and (b), the agreement and escrow records are generated after Siemens had accepted the requests that sent by First Great Western. While Figure 25 (c) and (d), shows agreements and escrows records that generated on the Siemens side right after maintaining all requests received from First Great Western and South Western Trains.

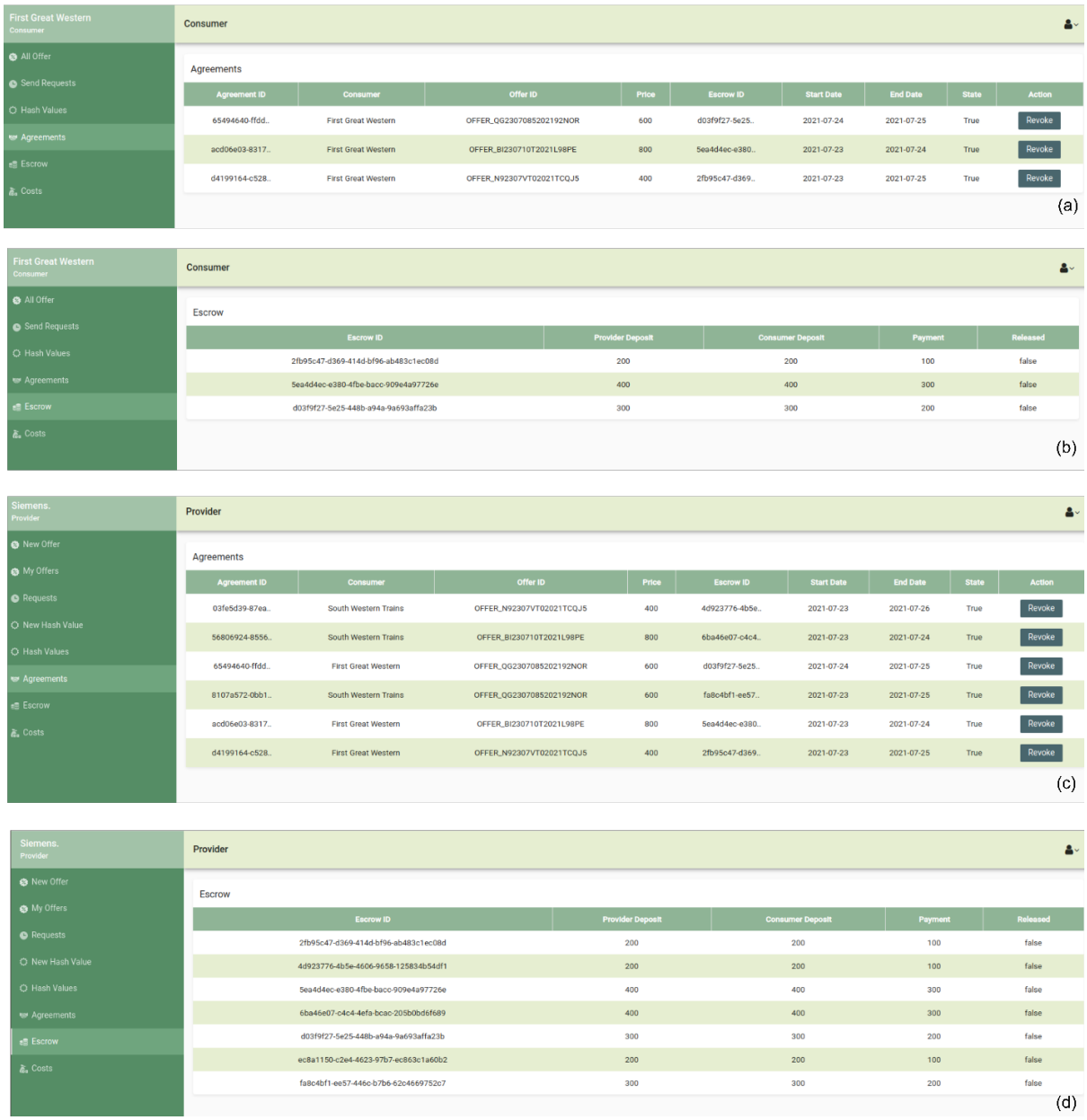

Figure 25: Generation of agreement and establishing Escrow.

Process 7 – Cost attribution: Each case in the cost attribution algorithm (Figure 18) is simulated and tested as follows:

- **Agreement is expired with no conflicts:** When the agreement is due, it will be flagged as expired, and the relevant escrow will be released to produce a cost distribution record. Figure 26 (a)-(c), shows how this is presented on the provider's side (e.g., Siemens), while Figure 26 (d)- (f) shows the consumer's side (e.g., First Great Western).

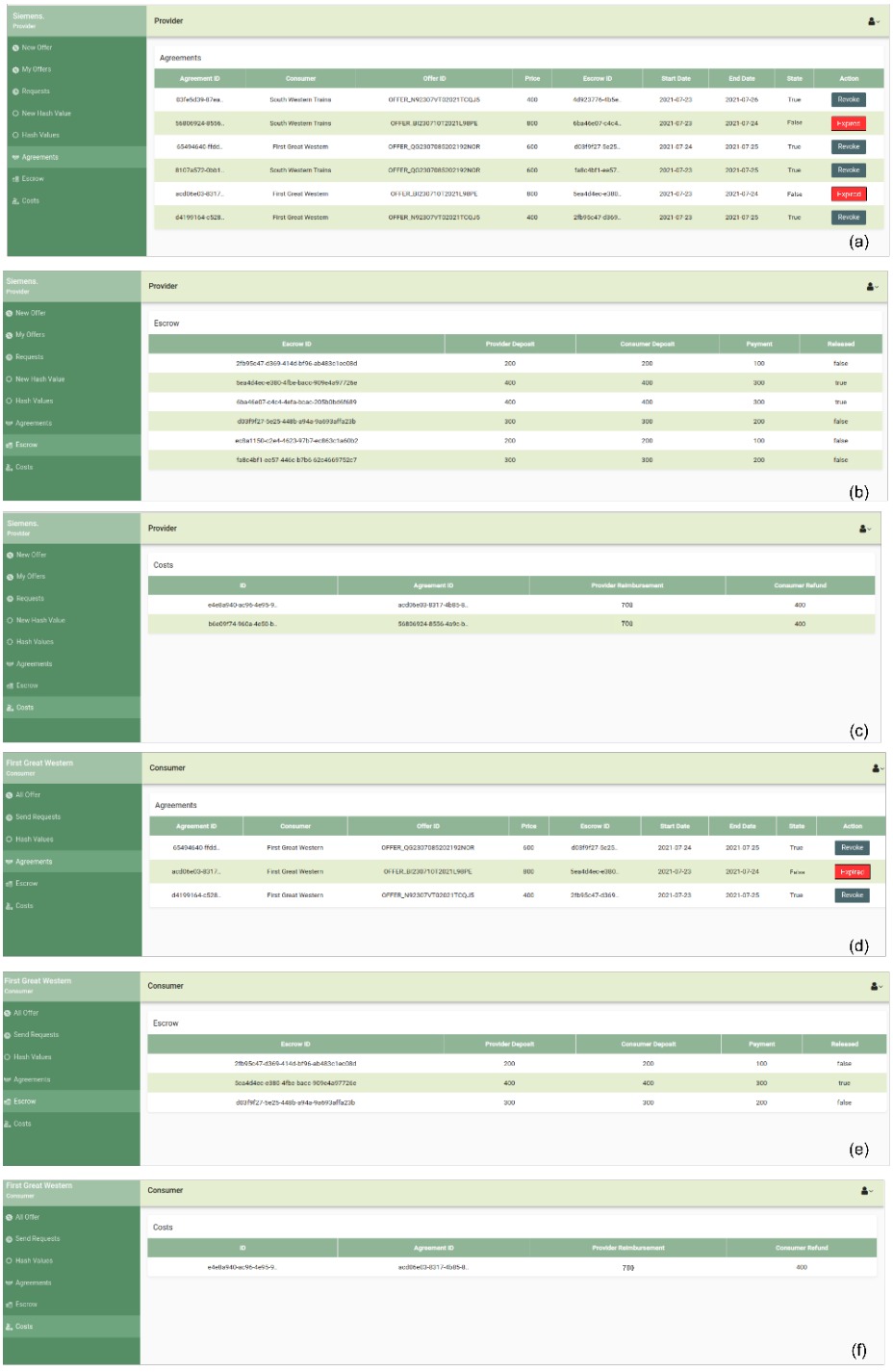

Figure 26: Contribution to costs on expiration of agreement.

- **Agreement is revoked by the provider:** In this case, the agreement will be flagged as revoked as shown in Figure 27 (a) and the provider will lose his deposit when the cost distribution record is generated as illustrated in Figure 27 (b). The consumer we get his deposit back in addition to compensating him by adding the provider's deposit to the refund.

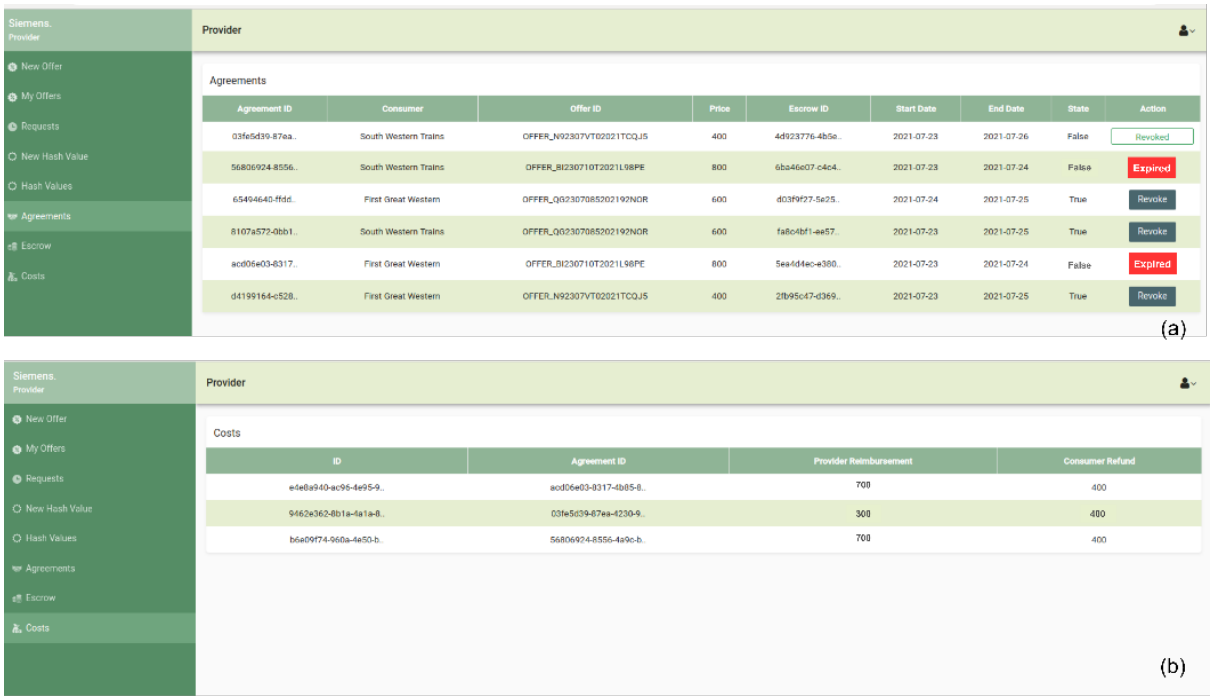

Figure 27: Contribution to costs on revocation by provider.

- **Agreement is revoked by consumer due to latency or falsified data:** In Figure 28 (a), the two agreements that Serco has with Network Rail have been revoked by the Serco. On the Network Rail side, a latency in appending the hash values and some redundancy was conducted to test this case as illustrated in Figure 28 (b) and (c). The result as presented in Figure 28 (d) shows that the consumer will be compensated with the provider's deposit to the refund along with his own deposit.

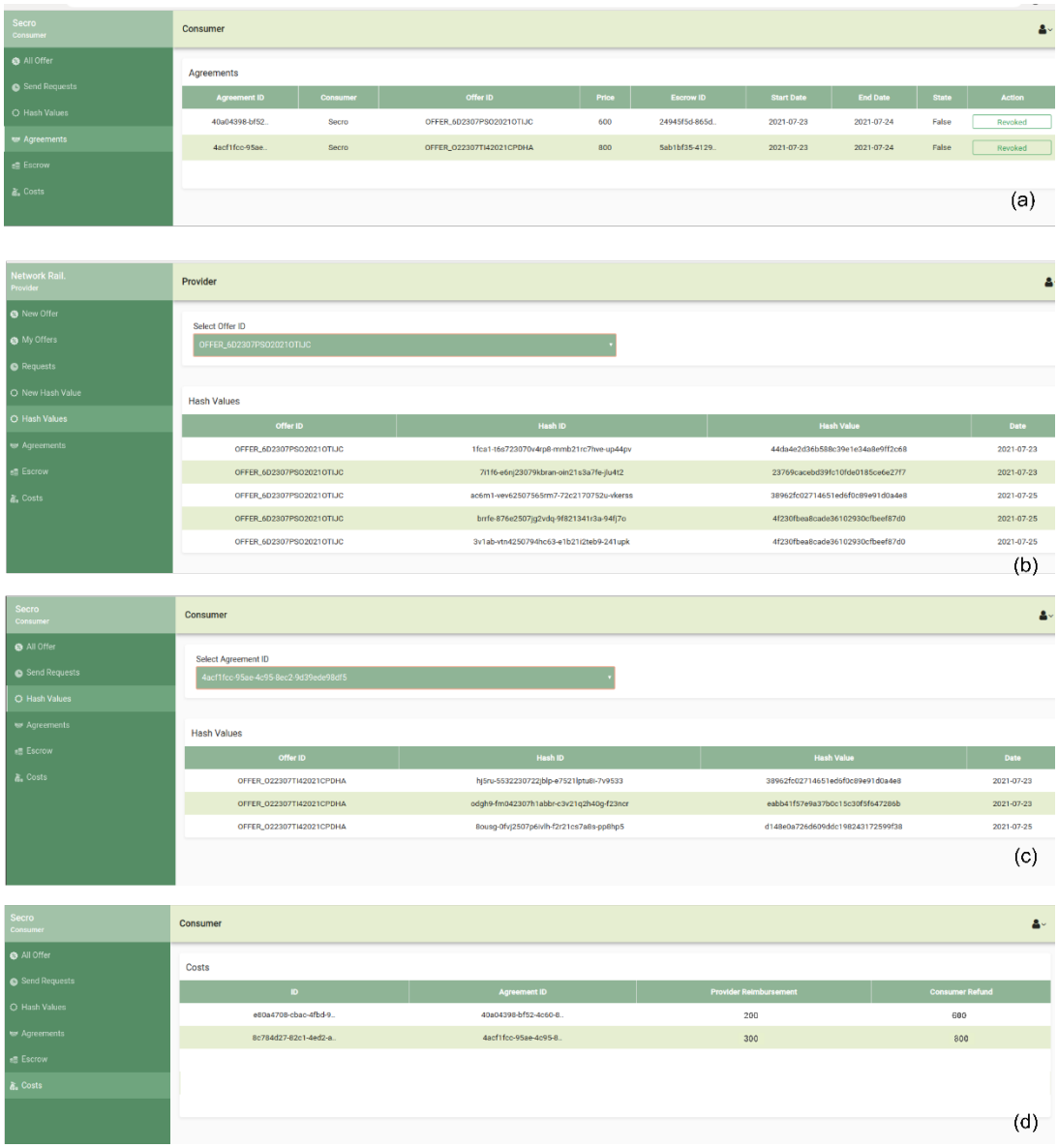

Figure 28: Consumer revocation of an agreement with a claim.

- **Agreement is revoked by the consumer and no latency or falsified data is proven:** In this case, the cost distribution will be calculated by adding the consumer's deposit to the provider's reimbursement as illustrated in Figure 29 (a) and (b).

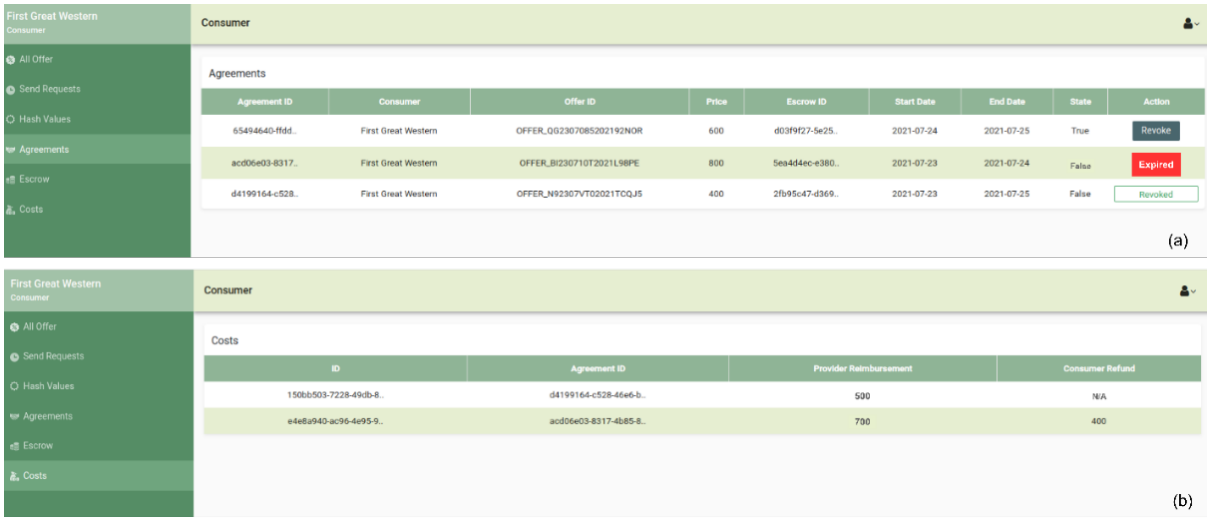

Figure 29: Unsubstantiated consumer revocation.

8. Conclusions

This document has presented an overview of the tooling, design rationale, and intended usage of the software framework being developed under the B4CM project. While the team expect that this will be substantively fleshed-out as we develop the specific use cases over the coming months, this overview was intended to provide good visibility of the work to date and the planned direction of travel with respect to the implementation. The project source code repository is publicly accessible, and will be updated over time as improved releases become available; the current content, although essentially only an alpha release, has been demonstrated in a toy industry context throughout Section 7.

References

[1] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” Technical report., [Online]. Available: https://gavwood.com/paper.pdf.

[2] V. Buterin, “Ethereum white-paper.,” [Online]. Available: https://ethereum.org/en/whitepaper/.

[3] E. Androulaki et al., “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” New York, NY, USA, 2018. doi: 10.1145/3190508.3190538

[4] K. Olson et al., “Sawtooth: An introduction – White paper.,” [Online]. Available: https://www. hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf.

[5] Hyperledger Iroha Community, “Iroha handbook: Installation, getting started, API, guides, and troubleshooting.”, [Online]. Available: https://iroha.readthedocs.io/_/downloads/en/1.1.3/pdf/.

[6] RSSB Research, “Detailed overview of selected RCM areas - Monitoring of overhead line integrity (T857 Report),” T857-4, 2012. [Online]. Available: https://www.sparkrail.org/Lists/Records/DispForm.aspx?ID=9919

[7] RSSB Research, “Detailed overview of selected RCM areas - Monitoring of pantograph integrity (T857 Report),” T857-3, 2012. [Online]. Available: https://www.sparkrail.org/Lists/Records/DispForm.aspx?ID=9918

[8] RSSB Research, “Detailed overview of selected RCM areas - Monitoring of axle journal bearings (T857 Report),” T857-01, 2012. [Online]. Available: https://www.sparkrail.org/Lists/Records/DispForm.aspx?ID=9916

[9] Hyperledger Fabric documentation (2020). [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/hyperledger-Fabric/latest/hyperledger-Fabric.pdf

[10] R. A. Alzahrani, S. J. Herko and J. M. Easton, "Blockchain Application in Remote Condition Monitoring," 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 2385-2394, doi: 10.1109/BigData50022.2020.9377895.

[11] J. D. Preece and J. M. Easton, "Towards Encrypting Industrial Data on Public Distributed Networks," 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 4540-4544, doi: 10.1109/BigData.2018.8622246.

[12] J. Tutcher, J. Easton, and C. Roberts (2017). “Enabling Data Integration in the Rail Industry Using RDF and OWL: the RaCoOn Ontology.” ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering, 3(2). https://doi.org/10.1061/AJRUA6.0000859