University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Engineering and Management of Business Systems

RESEARCH REPORT

Blockchain for hardware supply-chain business procedures

Scientific Adviser:
Dr.Eng. Gabriel Neagu

Author:
Eng. Adriana Dincă

Bucharest, 2019

Contents

List of Figures

Chapter 1

Introduction

1.1 Project Description

In this report we aim to prove the advantages of using Hyperledger Fabric by building a supply chain system that is transparent for all the entities involved in this process and to reduce the time consumption of the acquisition procedures. Ixia has an exhausting CAPEX process for acquiring the hardware devices needed by the development and QA teams so we believe that building a system based on Blockchain that solves the transparency and bottleneck issues is the perfect fit for replacing the existing procedures.

Hyperledger Fabric is a opensource framework developed by IBM and supported by the Linux Foundation that offers a permissioned private Blockchain solution. The framework is developed for private companies that know exactly the type of entities that are involved in their processes and what are their roles. Therefore the Hyperledger Fabric is suitable for solving the hardware devices supply chain transparency and bottleneck issues and to restrict the access of the participants involved in accordance with their position in the company. The hardware supply chain is CAPEX process that requires special approvals and priority analysis which is time consuming and often hard to track.

1.2 Project Objectives and Motivation

Hardware supply is an important aspect of the Ixia's employee daily work. Many software applications developed by Ixia teams run on dedicated hardware and in many situations only one employee can use that hardware at a time. The existing CAPEX procedure is very complicated and the time between ordering a new device and receiving it can be really long. It may happen that the team doesn't need it anymore at the time the device was received. The dedicated hardware is expensive (some chassis may cost up to $50000) so not receiving it on time may cost the department a lot of money and it can also delay the releases.

On each quarter, the manager has the responsibility of determining what are the hardware needs of the team and send the list to her/his superior for approval. Also, the manager has to check the price of the devices requested and if the costs are over the CAPEX budget, the hardware list should be prioritized. After completing the list and set the priorities if needed, the manager has to fill an *Internal Sales Order* on an internal platform. The hardware manufacturing managers have to check if there're are new ISOs for their teams to build or if they have the devices on stock. If the hardware machines are on stock they will be delivered to the department/manager that requested them. If there's no hardware with the desired specification on stock, the manager has

to schedule it for manufacturing. Also, the manufacturing managers need to make sure that the financial transaction was done successfully before delivering the hardware so they have to discuss with the financial department and validate that everything is ok. A huge disadvantage of this procedure is the lack of transparency regarding the status of the order and the time estimation until receiving the order. In addition to this the costs may differ due to the international tax changes.

The ***Hardware Supply-Chain*** project is a supply-chain business network that solves both the transparency and the access control restrictions. This system is using Hyperledger Composer framework for modeling the business network: participants, transaction and assets and connects to the Hyperledger Fabric via its API to add transactions in the Blockchain database and to manage participants' access accordingly with their roles in the network.

Chapter 2

Study of Hyperledger Projects

2.1 Hyperledger Project

Hyperledger is an open source project focused on Blockchain technology that aims to bring IoT, supply chain, finance, banking and manufacturing together. Is is hosted by the Linux Foundation and includes leaders from all the mentioned areas.

The Blockchain is a peer-to-peer distributed network, each participant of the network has its own copy of the ledger, and any transaction is validated by the majority of participants. The validation is done by solving Byzantine Generals Problem via consensus. The Hyperledger project offers solutions based on Blockchain with additional features such as the *smart contracts* and it has a large number of assistive tools to facilitate the embrace of Blockchain. All projects developed under the umbrella of Hyperledger and Linux Foundation are applications that use a ledger of transactions to establish transparency, accountability and trust. These projects are for a wide variety of business areas by providing the infrastructure, tools and frameworks to build applications based on Blockchain easily and in a short period of time. Additionally these applications follow the legal constrains by keeping the network closed to authorized participants that can be made accountable for their actions.

The project was launched in 2016 under the guidance of many important corporations such as IBM, Intel, etc. The first codebases that were released to the public were Hyperledger Fabric - a product that was the work result of three organizations (OpenBlockchain from IBM, libconsensus from Blockstream and the smart contracts from Digital Asset) and Hyperledger Sawtooth from Intel. These two frameworks were offering support for growing the development of Blockchain business solutions. Later on, there were other projects released that continued to offer assistive tools to support this growth. All projects that were released under the Hyperledger project were supervised by the Hyperledger Technical Steering Committee, a group of eleven specialists that were elected from the technical contributor community. In May 2016 the Linux Foundation and the corporate members involved in Hyperledger project elected an Executive Director to make sure that this idea has all the resources needed to be successful. In this position was placed the co-founder of Apache Software Foundation - Brian Behlendorf. The project became so successful so at the end of the next year the Hyperledger counted seven more projects and the number of members increased to more that 200.

The most successful projects were Hyperledger Sawtooth - multi-language support for distributed ledger, Hyperledger Fabric - the distributed ledger written in GO language, Hyperledger Composer - framework for accelerating the process of developing Blockchain applications, Hyperledger Iroha - the distributed ledger written in C++ and Hyperledger Indy - distributed ledger for decentralized identities. All these frameworks and tools are used in real-life applica-

tions except for Hyperledger Composer which is still in incubation. The Hyperledger Composer is an assistive tool for developers to facilitate the process of building a business network using the Hyperledger Fabric. It was proven that Hyperledger Composer is hard to maintain so the Hyperledger board decided to stop the development of new features and to keep the existing functionalities compatible with the new versions of Hyperledger Fabric. The main focus of the community is to add more features to Hyperledger Fabric framework which has a modular architecture so adding new features can be done quickly and without affecting the pre-existent functionalities.

In the following sections we are going to describe the Hyperledger Fabric and Composer frameworks. The Blockchain technology promises a revolution as big as the Web and the Hyperledger group understood its potential so they formed a Blockchain incubator to offer support to any bright idea related with Blockchain technology, smart contracts and the business world.

2.2 Hyperledger Fabric

2.2.1 General aspects

Hyperledger Fabric is an implementation of a distributed ledger developed mostly in Golang. The framework uses also other languages and technologies such as Javascript, Go or Java for smart contracts (chaincode), SDK in Node.js, Python, Java, Go and Rest. It is a solution for a secure, high-performance and permissioned Blockchain based network that has a modular architecture that allows plugins for any new features with no impact on the core functionalities.

Before getting into more details about the features provided by this framework it is important to mention that it is developed for enterprise use cases so the solution must solve the identity issue for transaction of type know-your-customer or the money laundering issue by offering transparency to authorities as participants with advanced permissions in the network. Thus this framework is permissioned which means that the participants are not anonymous, their identity is known by the other participants so they can be made accountable for their actions. The framework network is private and the transaction content is confidential so it can contain business sensitive information. In [3], Elli Androulaki and others are presenting the Hyperledger Fabric as a distributed operating system with all characteristics from the following paragraphs.

More than that the framework offers the possibility of selecting a consensus protocol based on the business needs. For example, if the business network is used by a single enterprise or it is governed by a authorized identity there's no need for using the Byzantine Fault Tolerant (BFT) protocol so it may sufficient to go with a simplified version such as Crash Fault Tolerant (CFT) consensus protocol. As a result, the Hyperledger Fabric eliminates the low latency of transaction validation and improves the network performance. The framework doesn't required CPU power for mining or for smart contract execution so the cost is similar with any distributed system.

The Hyperledger Fabric has a wide community of developers and activists that help with the development of new features and maintain the Fabric codebase. The number of contributors has grown to more than 200 members and the organizations involved to 35 so the project benefits of a diverse set of skills. With such a support the future of this project is really promising. It has modular and pluggable architecture so it can bring innovation to many industries.

2.2.2 Technical aspects

Modularity is one of the main characteristics of Hyperledger Fabric. It was designed to have a modular architecture so it can be used by a large number of industries from manufacturing to

finance.

The project was designed to be modular so all the components can be plugged in/out. The list of components (according to [2]) is formed of:

- an *ordering service* that makes sure that the order of transaction is correct and that all peers receive the blocks of transactions via broadcast; the ordering service can be plugged in/out and it runs in an independent environment (e.g Docker container named *hyperledger/fabric-orderer*);
- a *membership service provider* that enables only authorized entities to take part in the network(e.g a Docker container named *hyperledger/fabric-ca*); any entity that wants to connect to the business network has to have a cryptographic identity created by the business network Certification Authority (CA);
- *smart contracts* are applications that run independently in docker containers or other isolated environments and store agreements between network members; these smart contracts are named *chaincode* and they can be written in almost any programming language;
- a variety of *DBMS* options to store the ledger; a common option is CouchDB that runs in a Docker container named *hyperledger/fabric-couchdb*;
- more application's constrains for approval and validation policy;
- an optional *peer-to-peer gossip service* to leverage messages to all the peers by broadcast; makes sure that everyone receives up-to-date information about the ledger transactions and fills in any gaps if one node missed some of the broadcasts.

The chaincode (smart contract) is an application that facilitates and verifies the execution of a digital agreement between one or more network participants. The ledger is responsible for executing the digital contract at the time that it need to be applied. In a business network one or more digital contracts can co-exist and everyone that is connected to that network can create a contract. The network is responsible for validating that application code and it should be consider invalid until it is validated using order-execute architecture. In most Blockchain systems the smart contracts follow the rule order-execute so to reach consensus the order-execute design should be deterministic. Building a deterministic architecture is quite challenging and the code should be written in domain-specific programming languages (e.g Ethereum developed a DSL - Solidity for their decentralized applications) in most of the cases. Additionally this architecture brings performance issues due to the fact that transactions are added sequentially in the ledger.

The Hyperledger Fabric tries to solve the overhead of using the order-execute approach by proposing a new architecture type - execute-order-validate. This new design is composed by three stages. In the first stage the transactions are checked for correctness and if so they are immediately executed. After that, they are ordered using a consensus protocol and in the last stage they are validated according to an endorsement policy and added to the ledger. As a result of this approach the Hyperledger Fabric solves the issue of using DSL and non-determinism by removing any inconsistency and reduces the performace overhead. By removing any non-deterministic problem, the smart contracts can be written in any standard programming language. The first supported languages are Go and Node.js but the board of Hyperledger plans to support more popular programming languages such as Java in the future releases.

Permissioned Blockchain refers to restricting the access to the network to a known group of participants that are worth it to be trusted. It is not necessary that participants may know each other but all of them should be vetted by a trustful authority. The Fabric is using a permissioned Blockchain system so each member of the business network can easily be identified and if his/her actions (e.g deploying a smart contract or changing the configuration of the network) are malicious that member can be made accountable for them. In a permissionless

Blockchain system it is very hard to identify the real person that stood behind a certain action so Hyperledger Fabric is more suitable for the business world. The Certification Authority is one instrument that can vet for the business network members and any respectful business can easily obtain the authorization to participate in the network.

Privacy and confidentiality is key for most of the business use-cases therefore any solution used has to guarantee that the data cannot be available to the public. In permissionless systems the transactions' data is available to every node in the Blockchain network so such a system fails to fulfill the privacy business requirements. The main approach of permissionless Blockchain systems is to encrypt their data. However, this solution is far from perfect because other organizations can hold enough computational power to break any encryption in a reasonable amount of time. The approach of permissioned systems is to restrict the distribution of private data to only the involved participants via a channel. The Hyperledger Fabric allows any member of the network to create channels for different purposes so they can share confidential data with the authorized members. Another solution that can solve this issue for both for permissionless and permissioned Blockchains is the Zero Knowledge Proofs (ZKP) protocol but it is still under research so in the meantime creating a channel to share data with authorized entities is a good alternative.

As mentioned above, the Hyperledger Fabric offers support for pluggable components for many of its features. The ordering service is a pluggable module that is responsible of achieving consensus of ordering transactions. In the literature, there are two popular consensus protocols: Crash Fault Tolerant (CFT) and Byzantine Fault Tolerant (BFT). Each of them can have one or more implementations. For example, the Hyperledger Fabric has two possible implementations for the CFT algorithm: Kafka (based on Zookeeper) and *etcd* library from Raft.

The performance of Hyperledger Fabric depends on a variety of parameters: the number of nodes of the network, the number and size of transactions and hardware limitations so it is difficult to generalize the performance of the Hyperledger Fabric project. In order to determine the performance of a Business Network based on Fabric, the Hyperledger team has developed another project called Hyperledger Caliper. The community is contributing to define the set of measures that affect performance and scalability to help Caliper team build a powerful framework for benchmarking.

Taking all these into account, the Hyperledger Fabric is one of the best solution for permissioned distributed systems that integrates the Blockchain technology. The Hyperledger community is really involved in improving and building strong solutions for Blockchain business networks and the Hyperledger Fabric project has the biggest support from the community. Fabric supports a wide range of industries: from banking, manufacturing, supply chain to retail, healthcare, etc.

2.3 Hyperledger Composer

The Hyperledger Composer framework is one of the assistive tools developed by Hyperledger team that focuses on helping developers accelerate the process of creating Blockchain systems for business. Building Blockchain applications from scratch require high technical skills from networking protocols, cryptography and consensus algorithms and the development period can extend to one or more years. Having these issues in mind, Hyperledger community started a project to reduce the time and the skills set required to build Blockchain based systems. The project is called Hyperledger Composer and it is really appreciated by developers by reducing the time and effort from years to months.

The Hyperledger Composer framework can be easily integrated with Fabric via a Loopback REST API. Before getting into more details on how we can integrate the two projects it is important to describe the idea behind Hyperledger Composer.

This framework is using a Business Network Archive (BNA) to store information about the participants of the Blockchain network, the assets they can store in the ledger via transactions. Each business requires a custom BNA which defines the type of members, their roles and permissions and the transactions that can be done in the Blockchain. For example, in the supply-chain industry we need one or many manufacturers, buyers/clients, distributors and regulators. All these participants have restricted access to submit transactions or change the network configurations based on their role. For example a buyer can submit an *order* transaction but he/she cannot submit a *delivery* transaction.

The BNA has a well defined structure and each component should be defined using a custom Composer syntax. According to Hyperledger Composer Community [1] the BNA has to follow this structure:

- one or more Model files - written using Hyperledger Composer Modeling Language, an object oriented language that enables developers to define three elements: the namespace (only one for all the resources declared in the model), the list of entities of the network: participants, assets, events and transactions and the possibility to add entities defined in other namespaces;
- one or more Transactions Functionalities file - each transaction should have a metadata, decorators and a JavaScript function; these functions run when a participant submits a transaction; the REST API exposed by Hyperledger Fabric is invoked in the transaction function via *getNativeAPI* and the Hyperledger Composer API via *getAssetRegistry*; also these functions can issue events that other external systems can listen to via the *composer-client* library;
- an Access Control role file - written using Hyperledger Composer Access Control language that enables setting permissions for all the entities defined in the model; the rules that can be enabled/disabled are Create Read Update and Delete (C.R.U.D.); there are two types of access controls: business and network; both types are defined in the Access Control files and the business control is applied after the network control;
- a Query Definition file - it is an optional component that requires the *composer-rest-server* to be up and running; there are two types of queries: named and dynamic; the named queries are static, defined in the BNM and exposed by the REST server via GET endpoints; the dynamic queries can be defined in transaction functions or directly in the client application.

The Figure 2.1 outlines the BNA structure with all the components and a brief description for each of them. .

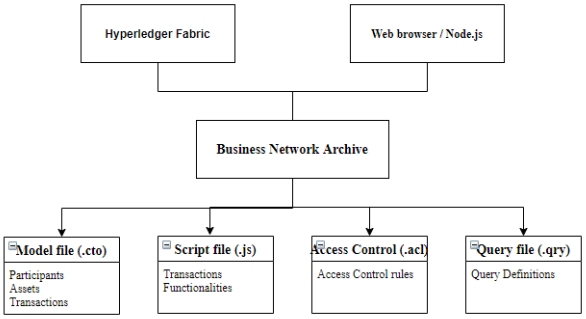

Figure 2.1: Hyperledger Composer Business Network Archive

There two possible ways to use this framework: the playground and the CLI tools. The first

solution offers a web user-interface to define, deploy and test business network directly on cloud or locally on the personal machine. The other solution is for Blockchain developers that want access to all the Fabric functionalities and that can be easily integrated with Visual Studio Code IDE and Atom editor.

More than that, the Hyperledger Composer has a specialized storage mechanism which decouples transactions information from any other information such as participants and assets data. All transactions are stored in the Blockchain ledger, everything else is located in the state database associated with that ledger. Another interesting feature is the Historian Registry that stores the transactions as *HistorianRecord* assets of the business network. These assets' definition is available in the Hyperledger Composer Model and they store information about the transaction that created them, the identity of the participant and the participant that invoked the transaction.

Another concept related with the Composer framework is the Connection Profile. Each Blockchain Network has a Connection Profile that defines the system and it is created by the network administrator to enable members to connect to the system. The Hyperledger Composer uses the Identity concept to ensure that only authorized members connect to the network. An Identity has a private key, a digital certification and a metadata file that is optional and may store the network name and other details about it. A Connection Profile and an Identity form a Business Network Card that offers a member access to the network. It is worth mentioning that only one identity can be stored in a Business Network Card and that identity may be associated with one of the participant that exists in the network. The concept of Business Network Cards was introduced to enable groups of participants with the same access rights to access the network so it is related with the network access control management.

Chapter 3

Implementation

3.1 General Aspects

After analyzing the existing Blockchain solutions we reached the conclusion that Hyperledger Fabric and Composer are suitable for the enterprise usage and we propose an application build using these two frameworks that solves the Capital Expenditures (CAPEX) related issues in the division Ixia Solution Group (ISG) of Keysight Technologies Inc. The proof of concept is focused on hardware supply chain for research and development teams of Network Visibility Solution (NVS) department.

The project idea is using the Blockchain strengths to solve the transparency problem so any authorized member of the network can check the status of a CAPEX order and it also improves the overall time of order processing. The Blockchain protocol uses proof-of-work to guarantee that only valid blocks of transactions are added in the ledger so once that data is added there's no way to modify it without being invalidated by the other members. As a result, all the orders added in the CAPEX supply chain are visible to all the authorized members so any attempt of crime is a fail.

The CAPEX procedure involves also some difficult and time consuming discussions to get the approval for a certain CAPEX order. The employees that order a hardware device need to get approval from their manager and the manager must check the CAPEX budget before approving it. All these issues can be easily managed in a permissioned blockchain network where the participants have different levels of access based on their roles. This type of Blockchain solves the accountability issue by allowing access to the network only to authorized members.

After some research work, we discovered that Hyperledger Composer is a good solution to limit the network access based on the participant role in the business. This framework offers two layers of control access: network and business. The network layer is the first level of access so each participant has to use unique identification network card to connect to the Blockchain network. The second level of access control is the business layer that restrict participants to do certain actions based on their role in the business. For example, a participant that wants to order a hardware doesn't have access to schedule that hardware for manufacturing.

Taking all of these into account, we build a project that uses Hyperledger Composer for business modeling, Hyperledger Fabric for accessing the Blockchain protocol via an API and solves all the issues described above.

3.2 Technical Aspects

In order to develop the hardware supply chain over the Blockchain protocol without too much effort we choose the frameworks developed by Hyperledger that facilitate the adoption of the Blockchain technology. The usage of these frameworks reduce the time and effort to develop the supply chain consistently.

The Business Network Model (BNM) is a concept introduced by the Hyperledger Composer project that facilitates the access to all the Fabric features such as introducing members with different roles, developing business functionalities in JavaScript language. For defining the business logic we can use the Hyperledger Composer API to connect to the Fabric or invoke the Fabric native functions. In most of the enterprise use case we have to provide a mechanism to restrict the access to the network and to have a transparent view of everything that happens in the system. These two frameworks fulfill really well the requirements of the business world so it is the obvious solution to select them for any Blockchain for business project.

The project idea is to handle the CAPEX procedures so it has to follow a certain business model. In the NVS department, a CAPEX process involves many actors from team leaders, managers, manufactures and external authorities. Keysight Technologies is an international company and it has to make sure that all the processes conducted internally and externally are complied with the laws where it operates.

In the first stage we focused on defining a supply-chain solution that handles only the CAPEX processes conducted internally due to a lack of visibility about the external procedures of the delivery stage. Therefore we have identified three types of participants: *Employee*, *Manufacturer* and *Regulator*, the assets traded are the *Order* and *Chassis* and the type of transactions submitted are *PlaceOrder* and *UpdateOrderStatus*.

As described in section 2.3 we developed a Business Network Archive the follows the structure required by the Hyperledger Composer framework (see Figure 3.1). In the following paragraphs we are going to get into the implementation aspects of the supply-chain project in order to outline the advantages of integrating the Hyperledger frameworks and tools in the CAPEX business process.

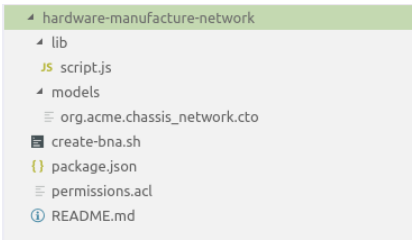

Figure 3.1: Structure of the *hardware-manufacture-network.bna* file

The Business Network Model is defined in *org.acme.chassis_network.cto* file and contains information about participants, assets, transactions, events, enums and concepts. All of these refer to the network resources and require to be associated with a namespace. The namespace used by all these resources is *org.acme.chassis_network*. A BNA can import resources from other namespaces and this practice is used when more than one organization is connected to the network. In our case there's only one organization that has access to the network and there's no need for other namespaces.

The first group of resources is the class definition group and it includes the class types of Participant, Asset and Transaction. In the supply chain we have three types of participants so there are four classes definition of type participant: *Employee*, *Division* and its subclasses *Manufacturer* and *Regulator*, two types of assets so there are two classes definition of type

asset: *Chassis* and *Order* and two types of transactions so there are two classes definition of type transaction: *PlaceOrder* and *UpdateOrderStatus*.

In the composer modeling language, classes of type *Participant* must be defined using the keyword *participant* as it can be seen in Figure 3.2. Each participant type requires an identification property that is set using the phrase *identified by* after the class name. For example, the *Employee* participant is identified using its identification number called *employeeId* and has a set of properties defined after the *o* syntax. This modeling language is object-oriented and we can make use of the inheritance property so both the *Manufacturer* and *Regulator* are subclasses of the participant *Division*. A participant of type *Regulator* is someone from the division's management team (e.g a project manager).

```
15 participant Employee identified by employeeId {
16   o String employeeId
17   o String name
18   o String email
19 }
20
21 abstract participant Division identified by name {
22   o String name
23 }
24
25 participant Manufacturer extends Division {
26 }
27
28 participant Regulator extends Division {
29 }
```

Figure 3.2: Hardware Supply-Chain: Participants

Another type of resources are the assets. In our business network we have identified two types of assets: *Order* and *Chassis*. An asset is defined using the reserved keyword *asset* followed by the asset's name, the phrase *identified by* and the name of the variable used for identification. The *Order* asset is used to start the ordering procedure and have a mechanism to track all the intermediate steps previous to register the new hardware device. The *Chassis* asset is used to track the hardware devices that were registered. In the Keysight hardware divisions, the chassis is identified using a serial number. The other properties of this asset are the *chassisDetails* - a field of type concept (which is an abstract class with no identification key), the *chassisStatus* - an enum that stores information about the life-cycle stage of a device (never used, active and scrapped) and the *order* - the employee that submitted the order (it is referenced with a reserved syntax –>). In Figure 3.3 and Figure 3.4 are defined the two type of asset resources used by the supply-chain.

The model has to contain also the type of transactions that can be submitted in the network and the events connected with them. In our business model we determine the need of two type of transactions: *PlaceOrder* and *UpdateOrderStatus*. These two types are required in order to have a complete flow of the CAPEX ordering process. In the first stage an employee needs to place an order, the order is than approved by the regulator and send to the manufacturer. During manufacturing, the order status is changed from scheduled for manufacturing to serial number assigned and then delivered. In the CAPEX procedures the manager is responsible to receive the orders and assign an owner for each of the devices. In Figure 3.5 and Figure 3.6 we outline the transactions definitions and the events associated: *PlaceOrderEvent* and *UpdateOrderStatusEvent*.

Each participant has a network card to connect to the supply-chain system that is provided by an administrator. In addition to the network access control we can restrict the actions done by a member of the network by defining rules for all the possible actions. The possible type of operations are create, read, update and delete (CRUD) and we can use this operations to create rules for all types of participants in report with other resources.

In the supply-chain network the *Employee* has to follow three rules: *EmployeeMakeOrder*, *Em-*

```
concept ChassisDetails {
  --> Manufacturer make
  o String chassisType
  o Integer noLineCards
  o String processorType
  o Integer noProcessors
  o Integer noCore
}

asset Chassis identified by serialNumber {
  o String serialNumber
  o ChassisDetails chassisDetails
  o ChassisStatus chassisStatus
  --> Employee owner optional
}

enum ChassisStatus {
  o ACTIVE
  o OFF_THE_ROAD
  o SCRAPPED
}
```

Figure 3.3: Hardware Supply-Chain: Asset of type *Chassis*

ployeePlaceOrder and *EmployeeReadOrder* (see Figure 3.7). In these rules the participant is *org.acme.chassis_network.Employee*, the resources used are *org.acme.chassis_network.PlaceOrder* and *org.acme.chassis_network.Order* and the rules specify if it is allowed or not to perform one of the operations of CRUD.

For the participant *Manufacturer* we had defined rules that involves the asset of type *Chassis* (see Figure 3.8) and the asset of type *Order* (see Figure 3.9). A participant of type *Manufacturer* can create new chassis and view the existing devices manufactured in the hardware department. Also this type of participant can update the status of order and read the existing orders assigned to him/her.

The last rule is associated with a participant of type *Regulator* who has extended permissions in the network. He/She can see all the orders submitted, who is the employee that ordered it and he/she has the possibility to assign a chassis to one of the employees (see Figure 3.10). This type of participant has a managerial role in the division so he/she is authorized to make changes in the network, to validate or invalidate a certain order and track the cause of orders' delays or other issues.

Additionally to all of these, we need to set some default rules for the system administrator that is responsible with network and business access control. The admin can create network cards for new members or deploy a new version of the *.bna*. We can also set some generally applied rules for the network members. The rules of the system administrator and network members are described in Figure 3.11. All participants can see the other members of the network. We think this rule is really useful in the early days of the network or for new members to learn about the system participants and be able to connect to them if they are in a situation that requires information from someone else in the business network.

The Hyperledger Composer has defined a default namespace *org.hyperledger.composer.system* that has some predefined participants that can be used to enable the network administrator special privileges. In our business, all participants have access to the system resources and only the *org.hyperledger.composer.system.NetworkAdmin* has full access to the user resources.

Another important aspect of the project implementation is the transactions definition. Transactions hold all the business logic and they use the Hyperledger APIs to connect to the Fabric and register the submitted transactions in the ledger.

For each transaction we have to define a function that holds the business logic. The *placeOrder*

```
asset Order identified by orderId {
  o String orderId
  o ChassisDetails chassisDetails
  o OrderStatus orderStatus
  o Options options
  --> Employee orderer
}

concept Options {
  o String trim
  o String interior
  o String[] extras
}

enum OrderStatus {
  o PLACED
  o SCHEDULED_FOR_MANUFACTURE
  o SERIAL_NUMBER_ASSIGNED
  o OWNER_ASSIGNED
  o DELIVERED
}

concept ChassisDetails {
  --> Manufacturer make
  o String chassisType
  o Integer noLineCards
  o String processorType
  o Integer noProcessors
  o Integer noCore
}
```

Figure 3.4: Hardware Supply-Chain: Asset of type *Order*

```
transaction PlaceOrder {
  o String orderId
  o ChassisDetails chassisDetails
  o Options options
  --> Employee orderer
}

event PlaceOrderEvent {
  o String orderId
  o ChassisDetails chassisDetails
  o Options options
  --> Employee orderer
}
```

Figure 3.5: Hardware Supply-Chain: Transaction of type *PlaceOrder* and its event

function (see Figure 3.12) is executed every time a transaction of type *PlaceOrder* is submitted. In this function we used some of the Hyperledger Composer API calls. The list of API calls used is:

- *getFactory* - returns the *module:composer-runtime.Factory* which is used to create, update resources and to create relationships between resources;
- *factory.newResource* - creates an asset;
- *factory.newRelationship* - creates a relationship between two resources;
- *factory.newEvent* -creates an event for the transaction;
- *emit* - sends an event for the transaction;
- *getAssetRegistry* - returns a *Promise* and it tries to get the asset registry with the specified identification name; if the registry exists the call return *@link module:composer-runtime.AssetRegistry AssetRegistry*, otherwise it returns an error.

```
transaction UpdateOrderStatus {
  o OrderStatus orderStatus
  o String serialNumber optional
  --> Order order
}

event UpdateOrderStatusEvent {
  o OrderStatus orderStatus
  o Order order
}
```

Figure 3.6: Hardware Supply-Chain: Transaction of type *UpdateOrderStatus* and its event

```
rule EmployeeMakeOrder {
    description: "Allow Employees to create and view orders"
    participant(e): "org.acme.chassis_network.Employee"
    operation: CREATE
    resource(o): "org.acme.chassis_network.Order"
    transaction(tx): "org.acme.chassis_network.PlaceOrder"
    condition: (o.orderer.getIdentifier() == e.getIdentifier())
    action: ALLOW
}

rule EmployeePlaceOrder {
    description: "Allow Employees to place orders and view they've done this"
    participant(e): "org.acme.chassis_network.Employee"
    operation: CREATE, READ
    resource(o): "org.acme.chassis_network.PlaceOrder"
    condition: (o.orderer.getIdentifier() == e.getIdentifier())
    action: ALLOW
}

rule EmployeeReadOrder {
    description: "Allow Employees to place orders and view they've done this"
    participant(e): "org.acme.chassis_network.Employee"
    operation: READ
    resource(o): "org.acme.chassis_network.Order"
    condition: (o.orderer.getIdentifier() == e.getIdentifier())
    action: ALLOW
}
```

Figure 3.7: Hardware Supply-Chain: Employee Rules

The *placeOrder* function creates an asset of type *Order*, register it in the database state and emits an event for this transaction.

The second function called *updateOrderStatus* is executed when a member submits a transaction of type *UpdateOrderStatus*. We used the same API calls to interact with the Hyperledger Composer and Fabric frameworks. This function has a more complex business logic. In the case a member changes the order status to *SERIAL_NUMBER_ASSIGNED* a new asset of type *Chassis* is created and registered. If the order status is changed to *OWNER_ASSIGNED* the chassis is assigned to an owner of type *Employee* and the chassis status is set to *ACTIVE*. In all cases, we update the order status with the new value (see Figure 3.13).

```
rule ManufacturerCreateChassis {
    description: "Allow manufacturers to create and view their chassis"
    participant(m): "org.acme.chassis_network.Manufacturer"
    operation: CREATE
    resource(c): "org.acme.chassis_network.Chassis"
    transaction(tx): "org.acme.chassis_network.UpdateOrderStatus"
    condition: (c.chassisDetails.make.getIdentifier() == m.getIdentifier() && tx.orderStatus == "SERIAL_NUMBER_ASSIGNED")
    action: ALLOW
}

rule ManufacturerReadChassis {
    description: "Allow manufacturers to create and view their chassis"
    participant(m): "org.acme.chassis_network.Manufacturer"
    operation: READ
    resource(c): "org.acme.chassis_network.Chassis"
    condition: (c.chassisDetails.make.getIdentifier() == m.getIdentifier())
    action: ALLOW
}
```

Figure 3.8: Hardware Supply-Chain: Manufacturer Chassis related Rules

```
rule ManufacturerUpdateOrder {
    description: "Allow manufacturers to view and update their own orders"
    participant(m): "org.acme.chassis_network.Manufacturer"
    operation: UPDATE
    resource(o): "org.acme.chassis_network.Order"
    transaction(tx): "org.acme.chassis_network.UpdateOrderStatus"
    condition: (o.chassisDetails.make.getIdentifier() == m.getIdentifier())
    action: ALLOW
}

rule ManufacturerUpdateOrderStatus {
    description: "Allow manufacturers to update order statuses and view they've done this"
    participant(m): "org.acme.chassis_network.Manufacturer"
    operation: CREATE, READ
    resource(o): "org.acme.chassis_network.UpdateOrderStatus"
    condition: (o.order.chassisDetails.make.getIdentifier() == m.getIdentifier())
    action: ALLOW
}

rule ManufacturerReadOrder {
    description: "Allow manufacturers to view and update their own orders"
    participant(m): "org.acme.chassis_network.Manufacturer"
    operation: READ
    resource(o): "org.acme.chassis_network.Order"
    condition: (o.chassisDetails.make.getIdentifier() == m.getIdentifier())
    action: ALLOW
}
```

Figure 3.9: Hardware Supply-Chain: Manufacturer Order related Rules

```
rule RegulatorAdminUser {
    description: "Let the regulator do anything"
    participant: "org.acme.chassis_network.Regulator"
    operation: ALL
    resource: "**"
    action: ALLOW
}
```

Figure 3.10: Hardware Supply-Chain: Regulator Rule

```
rule ParticipantsSeeSelves {
    description: "Let participants see themselves"
    participant(e): "org.hyperledger.composer.system.Participant"
    operation: ALL
    resource(r): "org.hyperledger.composer.system.Participant"
    condition: (r.getIdentifier() == e.getIdentifier())
    action: ALLOW
}

rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule System {
    description: "Grant all full access to system resources"
    participant: "org.**"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

Figure 3.11: Hardware Supply-Chain: Hyperledger Composer predefined participants Rules

```
/**
 * Place an order for a chassis
 * @param {org.acme.chassis_network.PlaceOrder} placeOrder - the PlaceOrder transaction
 * @transaction
 */
async function placeOrder(orderRequest) { // eslint-disable-line no-unused-vars
    console.log('placeOrder');

    const factory = getFactory();
    const namespace = 'org.acme.chassis_network';

    const order = factory.newResource(namespace, 'Order', orderRequest.orderId);
    order.chassisDetails = orderRequest.chassisDetails;
    order.orderStatus = 'PLACED';
    order.orderer = factory.newRelationship(namespace, 'Employee', orderRequest.orderer.getIdentifier());
    order.options = orderRequest.options;

    // save the order
    const assetRegistry = await getAssetRegistry(order.getFullyQualifiedType());
    await assetRegistry.add(order);

    // emit the event
    const placeOrderEvent = factory.newEvent(namespace, 'PlaceOrderEvent');
    placeOrderEvent.orderId = order.orderId;
    placeOrderEvent.chassisDetails = order.chassisDetails;
    placeOrderEvent.options = order.options;
    placeOrderEvent.orderer = order.orderer;
    emit(placeOrderEvent);
}
```

Figure 3.12: *PlaceOrder* Function

```
/**
 * Update the status of an order
 * @param {org.acme.chassis_network.UpdateOrderStatus} updateOrderStatus - the UpdateOrderStatus transaction
 * @transaction
 */
async function updateOrderStatus(updateOrderRequest) { // eslint-disable-line no-unused-vars
    console.log('updateOrderStatus');

    const factory = getFactory();
    const namespace = 'org.acme.chassis_network';

    // get chassis registry
    const chassisRegistry = await getAssetRegistry(namespace + '.Chassis');
    if (updateOrderRequest.orderStatus === 'SERIAL_NUMBER_ASSIGNED') {
        if (!updateOrderRequest.serialNumber) {
            throw new Error('Value for serialNumber was expected');
        }
        // create a chassis
        const chassis = factory.newResource(namespace, 'Chassis', updateOrderRequest.serialNumber );
        chassis.chassisDetails = updateOrderRequest.order.chassisDetails;
        chassis.chassisStatus = 'OFF_THE_ROAD';
        await chassisRegistry.add(chassis);
    } else if(updateOrderRequest.orderStatus === 'OWNER_ASSIGNED') {
        if (!updateOrderRequest.serialNumber) {
            throw new Error('Value for serialNumber was expected');
        }

        // assign the owner of the chassis to be the Employee who placed the order
        const chassis = await chassisRegistry.get(updateOrderRequest.serialNumber);
        chassis.chassisStatus = 'ACTIVE';
        chassis.owner = factory.newRelationship(namespace, 'Employee', updateOrderRequest.order.orderer.employeeId);
        await chassisRegistry.update(chassis);
    }

    // update the order
    const order = updateOrderRequest.order;
    order.orderStatus = updateOrderRequest.orderStatus;
    const orderRegistry = await getAssetRegistry(namespace + '.Order');
    await orderRegistry.update(order);

    // emit the event
    const updateOrderStatusEvent = factory.newEvent(namespace, 'UpdateOrderStatusEvent');
    updateOrderStatusEvent.orderStatus = updateOrderRequest.order.orderStatus;
    updateOrderStatusEvent.order = updateOrderRequest.order;
    emit(updateOrderStatusEvent);
}
```

Figure 3.13: *UpdateOrderStatus* Function

Chapter 4

Business Procedures

4.1 ISG CAPEX Procedures

The *Hardware Supply-Chain* use cases are: ordering a new hardware device and sending the device for service investigation(RMA). Ixia Solutions Group (ISG) teams are focused on developing software applications for testing the network performance, getting network visibility by monitoring the traffic or finding security threats before compromising any data. Therefore teams need specialized hardware devices such as routers, gateways, switches to more powerful machines as packet brokers, ESXI servers, clusters, etc. The ISG division offers deep expertise in network testing, security and visibility which requires investing in powerful hardware equipment. In each quarter of the financial year the ISG division has a CAPEX budget to invest in hardware so each team can request new hardware devices. Each manager has the responsibility of identifying the hardware requirements by consulting his/her team and assuring that the total cost doesn't exceed the budget in which case the CAPEX list should be prioritized.

When the CAPEX requirements list is completed, the manager can proceed to order them. For ordering the equipment he/she has to send emails for approval and after the order is accepted by a superior, the manager can added it in an internal platform.

The RMA procedure can be performed by the team manager as soon as he/she is informed about the hardware malfunction. Most hardware devices have an warranty period where they can be replaced or repaired free of charges by the manufacturing divisions. Before sending it to RMA the employee has to get approval from his/her manager who also needs to ask for approval from the superior.

Taking all these into consideration, the CAPEX can become quite exhausting and if something happens with the hardware it is hard to find the root cause of the issue. In order to solve the issue of transparency when tracking a device and to reduce the time of getting all the required approvals we propose a Blockchain based solution that allows everyone to check the status of orders and hardware devices. More than that we want a solution that offers immutability so nobody can modify the history of a chassis which is one of the core features of the Blockchain protocol. The idea is to build a permissioned Blockchain application that solves the transparency and the immutability requirements and is able to offer support for the network access control problem. For reasons of competitiveness a business has to keep its internal procedures private and it is important to use a solution that enables only authorized members to get access to the system data. After some research work we found two frameworks developed by Hyperledger community specially for these business requirements so we decided to use them to build our hardware supply-chain system that can be used to replace the CAPEX complicated procedures.

4.2 Hardware Supply-Chain Use Cases

The hardware supply-chain offers support for two main uses cases. The first use case solves the new hardware ordering procedure and the second use case solves the situation when a device is due to service (RMA).

In order to prove the functionalities of this system we have created instances for all types of participants and three types of network card for each of them. In Figure 4.1 we can see the three network cards used by the three types of participants: *employee@keysight*, *hardware@keysight* and *regulator@keysight.com*.

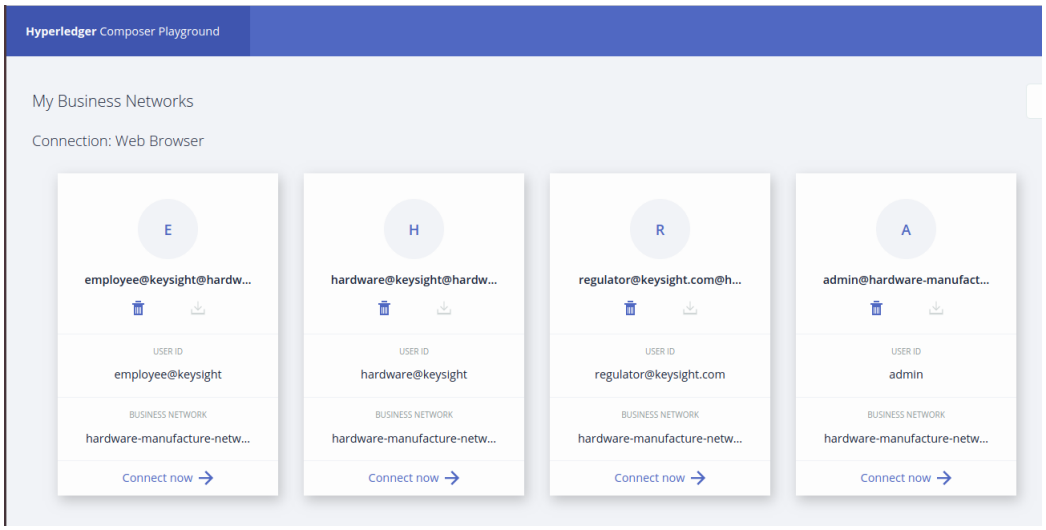

Figure 4.1: Hardware Supply-Chain: Playground

The first use case flow is triggered by an employee by creating an asset of type *Order* via a transaction of type *PlaceOrder* (see Figure 4.2).

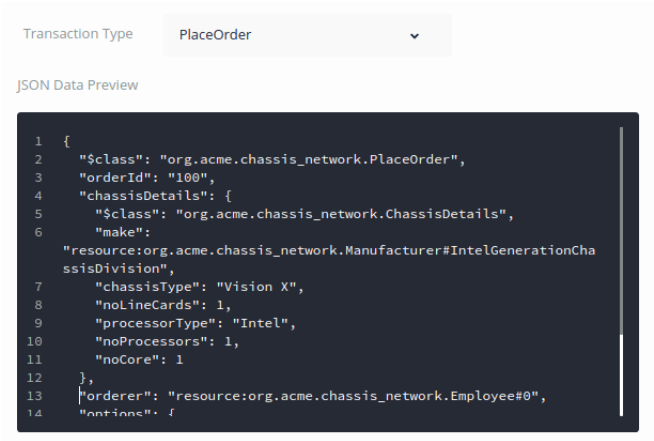

Figure 4.2: Hardware Supply-Chain: Example of *PlaceOrder* Transaction

Each order has assigned a manufacturer that is in charge and authorized to create the device. The manufacturer analyzes the order and checks if the requested device is available or need to be scheduled for manufacturing. If no device is available the manufacturer submits a transaction of type *UpdateOrderStatus* and update the order status in *SCHEDULED_FOR_MANUFACTURE* (see Figure 4.3).

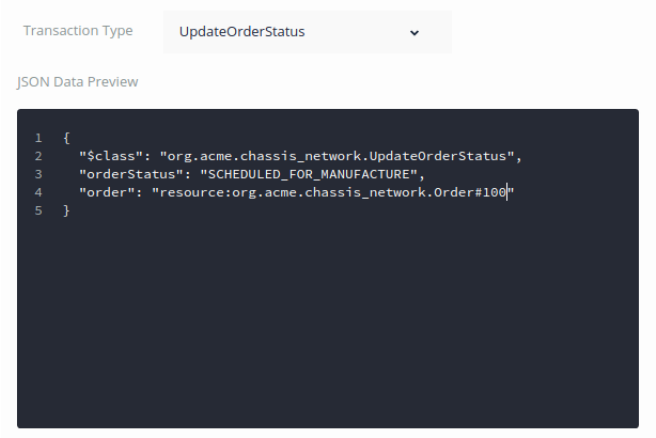

Figure 4.3: Hardware Supply-Chain: Update order status in *SCHEDULED_FOR_MANUFACTURE*

When he finished creating the hardware ordered, he/she sets a serial number for the device by submitting a transaction of type *UpdateOrderStatus* with the status *SERIAL_NUMBER_ASSIGNED* and a unique value for the field *serialNumber* (see Figure 4.4).

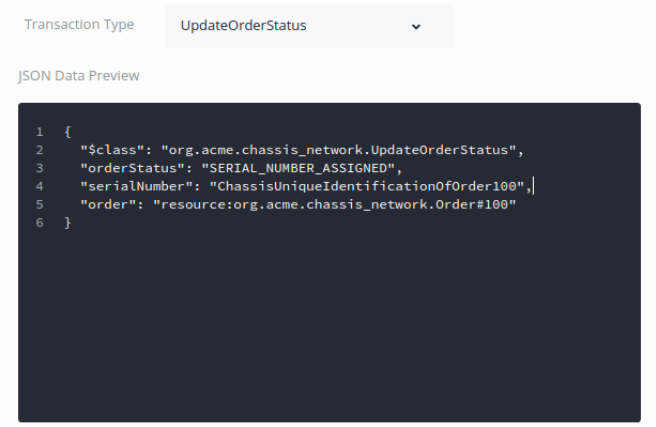

Figure 4.4: Hardware Supply-Chain: Update order status in *SERIAL_NUMBER_ASSIGNED*

In this moment, a new chassis is added in the list of assets stored in the Blockchain that can be easily identified via its serial number (see Figure 4.5).

In the end, the manufacturer has to send the device to the employee that ordered it. This step is completed by submitting a transaction of type *UpdateOrderStatus* and change the order status to *DELIVERED* (see Figure 4.6). The manufacturer is also responsible to correctly inform the delivery company all the legally required details about the packet and to set the recipient of the packet to be the direct manager of the employee that ordered the device. This use case flow ends when the designated manager receives the hardware and assigns it to the employee that ordered it via a transaction of type *UpdateOrderStatus* (see Figure 4.7). So the order changes its status in *OWNER_ASSIGNED* and the chassis has a new property set *owner* (see Figure 4.8).

All the submitted transactions are registered and stored in the Blockchain ledger so nobody can corrupt that data. Having an immutable ledger is one of the powerful advantages of the

```
{
  "$class": "org.acme.chassis_network.Chassis",
  "serialNumber": "ChassisUniqueIdentificationOfOrder100",
  "chassisDetails": {
    "$class": "org.acme.chassis_network.ChassisDetails",
    "make": "resource:org.acme.chassis_network.Manufacturer#IntelGenerationChassisDivision",
    "chassisType": "Vision X",
    "noLineCards": 1,
    "processorType": "Intel",
    "noProcessors": 1,
    "noCore": 1
  },
  "chassisStatus": "OFF_THE_ROAD"
}
```


Figure 4.5: Hardware Supply-Chain: The manufactured chassis

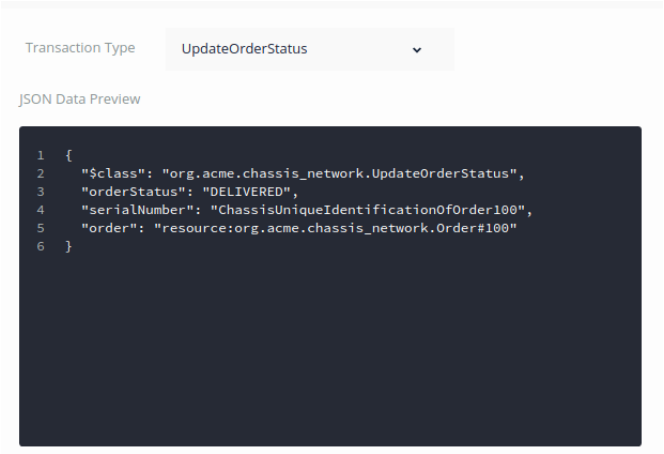

Figure 4.6: Hardware Supply-Chain: Update order status in *DELIVERED*

Blockchain usage. The Hyperledger Composer playground offers also the possibility to read the information stored in the ledger (see Figure 4.9).

The second use case is similar with the one described above with the mention that the stage of delivery may be missing. In this situation the status of a chassis is set to *SCRAPPED*.

The user interface was provided by Hyperledger Composer playground but the business model we defined can be integrated with other web frameworks via the *composer-rest-server* module. The business network can be deployed on this server and we can connect from any web client to the REST API it has exposed.

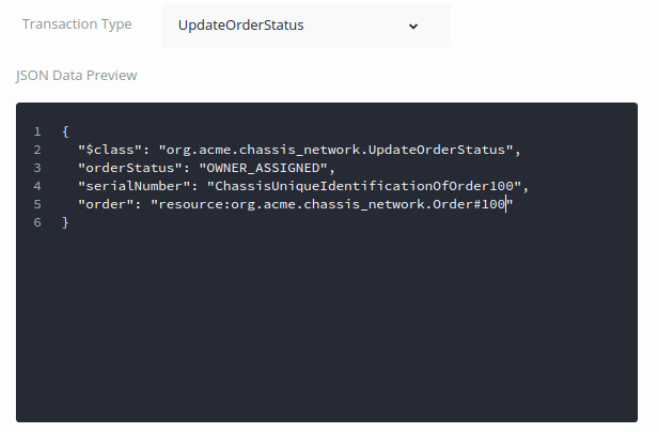

Figure 4.7: Hardware Supply-Chain: Update order status in *OWNER_ASSIGNED*

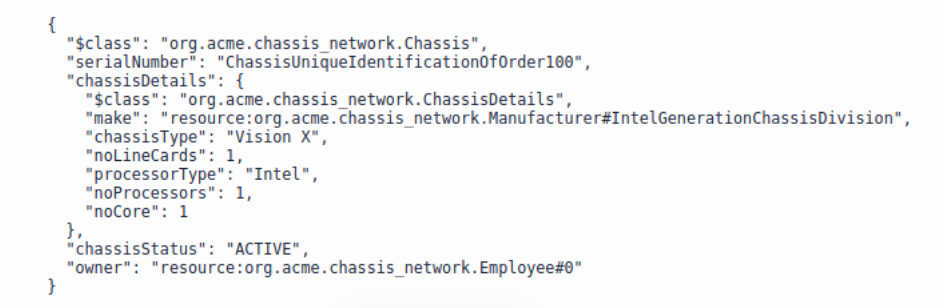

Figure 4.8: Hardware Supply-Chain: Chassis' Owner

Figure 4.9: Hardware Supply-Chain: Blockchain Ledger

Chapter 5

Conclusion

In conclusion, we created a proof-of-concept to solve the issues identified in tha CAPEX procedures conducted by the Network Visibility Solution department of Ixia Solutions Group using two business oriented Blockchain based projects: Hyperledger Composer and Hyperledger Fabric. The main motivation for this application is to outline the diversity of advantages the Blockchain protocol offers for the business world. In addition, we have explored some powerful solutions that facilitate the adoption of this technology. The two frameworks developed by the Hyperledger organization are programmer-friendly so anyone with some basic programming skills can manage to develop a Blockchain based application in a few months. In the future, we plan to generate the API for this business network and a basic web client to complete the Hardware Supply-Chain user experience.

Bibliography

[1] Hyperledger Composer Community. Hyperledger composer documentation. Published on Hyperldeger Github, https://hyperledger.github.io/composer/latest/introduction/introduction.html, 2019. Accessed Mar. 2019.

[2] Hyperledger Fabric Community. Hyperledger fabric documentation. Build with Sphinx and Read the Docs, https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html#, 2019. Accessed Mar. 2019.

[3] Vita Bortnikov Christian Cachin Konstantinos Christidis Angelo De Caro David Enyeart Christopher Ferris Gennady Laventman Yacov Manevich Srinivasan Muralidharan Chet Murthy Binh Nguyen Manish Sethi Gari Singh Keith Smith Alessandro Sorniotti Chrysoula Stathakopoulou Marko Vukolić Sharon Weed Cocco Jason Yellick Elli Androulaki, Artem Barger. Hyperledger fabric: A distributed operating system for permissioned blockchains. Appears in proceedings of EuroSys 2018 conference, https://arxiv.org/abs/1801.10228v2. Accessed Mar. 2019.

[4] LinuxFoundationX LFS171x. Blockchain for business - an introduction to hyperledger technologies. https://courses.edx.org/, 2017. Accessed: 2019-01-06.