ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL KRINDGES

A PROPOSED ARCHITECTURE FOR IOT DATA MARKET

Porto Alegre
2019

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

A PROPOSED ARCHITECTURE FOR IOT DATA MARKET

RAFAEL KRINDGES

Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Fabiano Passuelo Hessel

Porto Alegre
2019

Ficha Catalográfica

K92p Krindges, Rafael

A Proposed Architecture for IoT Data Market / Rafael Krindges . – 2019.

87.

Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fabiano Passuelo Hessel.

1. IoT. 2. blockchain. 3. Data marketplace. 4. Ethereum. 5. Internet of things. I. Hessel, Fabiano Passuelo. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

RAFAEL KRINDGES

A PROPOSED ARCHITECTURE FOR IOT DATA MARKET

This Thesis has been submitted in partial fulfillment of the requirements for the degree of Master of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on August 27th, 2019.

COMMITTEE MEMBERS:

Prof. Dr. Jorge Luís Victória Barbosa (PPGCA/Unisinos)

Prof. Dr. César Augusto Mission Marcon (PPGCC/PUCRS)

Prof. Dr. Fabiano Passuelo Hessel (PPGC/PUCRS - Advisor)

Dedico este trabalho a meus pais e minha esposa Pauline.

UMA PROPOSTA DE ARQUITETURA PARA O MERCADO DE DADOS EM IOT

RESUMO

O aumento no número de dispositivos IoT e conseqüentemente os dados gerados pelos mesmos fizeram com que se percebesse ali um novo mercado em potencial. Empresas e indivíduos estão buscando novas receitas financeiras através de estruturas já existentes e a venda de dados gerados por dispositivos móveis inteligentes é uma delas.

Porém, desafios precisam ser superados para que esse mercado aconteça. Questões como formato dos dados, remuneração dos entes envolvidos, como encontrar a informação desejada no meio de tantas disponíveis são apenas algumas delas. Este trabalho apresenta uma definição de arquitetura para um mercado de dados para dispositivos IoT, quebrando assim os silos de dados que existem. O trabalho detalha os módulos presentes na arquitetura proposta e, uma prova de conceito foi realizada, utilizando MQTT e Ethereum. A prova de conceito validou todas as etapas envolvidas numa transação de venda de dados, desde a sua geração, até o momento que o vendedor recebe o pagamento. Os problemas e desafios encontrados também são detalhados.

Palavras-Chave: IoT, internet das coisas, blockchain, mercado de dados.

A PROPOSED ARCHITECTURE FOR IOT DATA MARKET

ABSTRACT

The increase in the number of IoT devices and consequently, in the data generated by them created a new potential market. Companies and individuals are seeking new financial revenue through existing devices, and the sale of data generated by these devices is one of them.

However, some challenges exist to make this happen. An agreement in what format to exchange the data, how to get paid for it, how to find the desired information in a world with millions of devices are just a few.

This work aims to present an architecture that helps in defining a data marketplace focused on IoT devices that help to break the data silos that exist today. A detailed explanation of the proposed data marketplace modules is presented, and a proof of concept was implemented, using MQTT and Ethereum. The proof of concept involved all the phases in a data trading transaction, since the generation until the data seller payment. The issues encountered and the challenges that exist in the area are also presented.

Keywords: IoT, internet of things, blockchain, data marketplace, Ethereum.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1. INTRODUCTION

One of the most valuable assets in the past century was oil, but this is changing. Today, the data is considered the commodity more valuable, it is the oil of the digital era [17]. Companies and people are looking for ways to make a profit with this data, especially because 2.5 quintillion bytes of data are generated every day [23]. Some amount of these data are generated by the Internet of Things (IoT) devices, and data marketplaces designed especially for them are emerging.

A data marketplace or data market is a place where people can buy/sell data. Imagine being able to buy images generated from webcams; data sensed and collected from Medical IoT devices (MIoT) wearable by patients for research purpose, data collected from various public IoT devices in Smart Cities. All of these are an example of data that can be monetized and available in IoT data market. Moreover, the amount of data available will grow up especially with the estimation of having 20 billion IoT devices in 2020 [22].

In a traditional data marketplace, we have at least the data owner and the data consumer. In the data market for IoT devices, this is quite the same. Since IoT devices can be defined as sensors, a device that detects events or changes in the environment and sends the information to the digital world, the interaction between the ones that own the data and the ones that want to buy created a vision and model called Sensing as a Service [36].

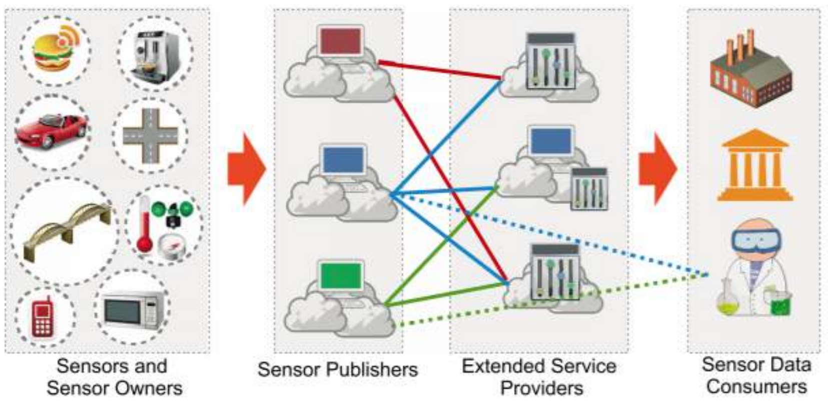


Figure 1.1 – The sensing as a service model. Source [38]

Figure 1.1 shows the four conceptual layers of Sensing as a Service model: *(i) sensors and sensor owners; (ii) sensor publishers (SPs); (iii) extended service providers (ESPs)*; and *(iv) sensor data consumers.*

Sensors and Sensor Owners Layer: consists of the sensor devices that detects, measures or sense a physical event such as temperature, humidity and so on. The sensor owner has the control of a specific sensor at a given moment.

Sensor Publishers Layer: this layer detects the available sensors, communicate with the sensor owners and obtain approval to publish the data sensors in the cloud.

Extended Service Providers Layer: can be considered the most intelligent of all four layers. It can interact with multiple Sensor Publishers Layer and retrieve the desired information for the Consumers.

Sensor Data Consumer Layer: these are the data consumers, for example, governments, business organizations, academic institutions and so on. They do not communicate directly with the sensors or sensors owners but through the or ESPs.

Some architectures proposed by [33], [8], [15], take advantage of this model but always add a man in the middle: at least the Sensor Consumer should be a person that selects the best data source and perform the contract agreement.

1.1 MOTIVATION

There are economical and environmental factors behind the re-usage of existing devices. For example, if temperature sensors across the city already exist owned by Company A, and another Company B want this data for a particular street, why to deploy new devices (that will exist in duplication) if you can buy the data from the Company A? The current infrastructure already available can be used or shared with multiple partners creating new value with a minimal investment. If data owners can simple catalog what type of data they have and the price want to sell an autonomous system can perform the match and find a buyer for it.

In addition to the use of IoT devices that serve as general purpose business, users can gain benefits from the user data market. Companies may want to know the users better and can do that if the users make their data available. Doing that, companies will be able to optimize their operations, saving costs, and create new products and services that meets individuals needs [5]. Example of this scenario is a Smart Home with an Internet-connected refrigerator. The refrigerator manufacturer maybe wants to know the usage data, temperature settings, number and frequency of door openings, and all of these can help in better refrigerator designs. The refrigerator owner may allow this data collection if paid for it.

By analyzing existing data market architectures, there is a need for a definition and study for a solution that covers the end-to-end process, since the data generation, the agreement between buyers and sellers, the money exchange and the solution scalability.

1.1.1 CONTRIBUTIONS

The original contribution of this work is the definition of an architecture and all components that automate the selling process of IoT data without human intervention. With the aim to answer the research questions, this work proposes the following specific contributions:

- Development of a data matching mechanism that enables buyers to find sellers automatically.
- Integration of a Payment Management and a Contract Management subsystem with the proposed architecture.
- Integration of a message broker subsystem with the proposed architecture.
- Performance evaluation of the proposed architecture.

2. THEORETICAL BACKGROUND

This chapter presents definitions regarding this work proposal. Section 2.1 presents the Internet of Things concept, section 2.2 presents Blockchain, section 2.3 presents Smart Contract, section 2.4 presents Data Market and section 2.5 presents MQTT Protocol.

2.1 INTERNET OF THINGS

The internet evolved in five phases as illustrated in Figure 2.1 [35]. All started with two computers connected together and ended in what we know today as Internet of Things. In a primary definition, Internet of Things agents are smart, connected devices with Internet access and some processing capacity. Such appliances can usually sense and send data to centralized servers. The popularity of the term introduced in the late 1990s is increasing due to the promises of according [48] allowing "people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any Service." It will help to create a better world for all, where objects around us know what we like, what we want, and what we need and act accordingly without explicit instructions [37].

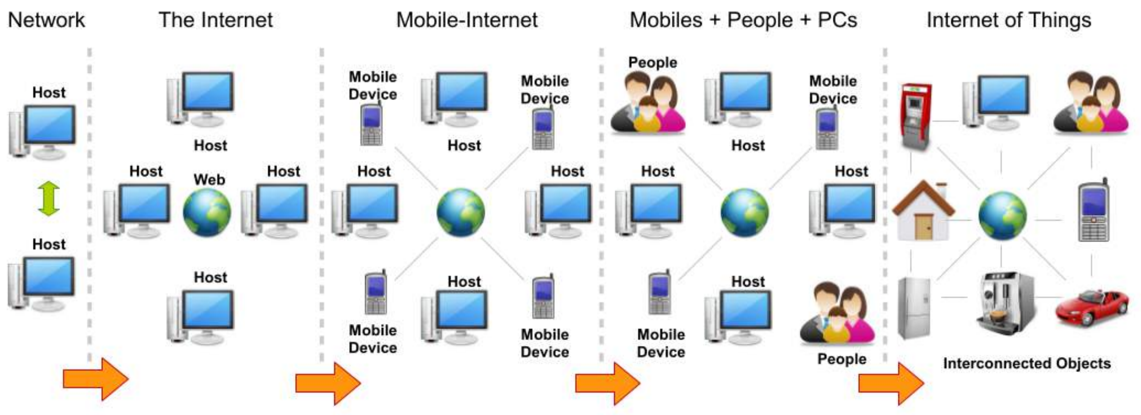


Figure 2.1 – Evolution of the Internet in five phases. Source [35]

The Internet of Things is the pillar for building smart energy systems, smart transportation, smart healthcare, smart buildings, and smart cities [28]. An example is the Array of Things project, where multiple devices are installed to monitor a set of cities environment [31].

2.2 BLOCKCHAIN

Satoshi Nakamoto is considered the father of the technology we knew as blockchain. In his paper “Bitcoin: A Peer-to-Peer Electronic Cash System” the mathematical foundation for the bitcoin cryptocurrency is present. The blockchain can be considered a stable digital ledger system implemented in a distributed way and usually without a central authority. It enables users to record transactions in a ledger public such that no transaction can be changed once published [56].

There are three types of blockchains network: private, public and permission-based (or consortium). The public ones enable users to join and contribute to an open network, where anyone can join and work in any core activity. Bitcoin and Ethereum are examples of these networks.

Private business companies generally run the private blockchains, and a participant can join only through an official and verified invitation. The private blockchain owner controls who is allowed to participate in the network, who can make transactions and who can validate/authenticate the blockchain changes [43].

The permission-based network is a mix of public and private networks. In this kind of network, a participant may not need permission to join the network, but needs permission to transact with another participant [9].

The table 2.1 provides a comparison among these three blockchain systems.

Table 2.1 – Blockchain networks properties. Adapted from [24]

Property	**Public blockchain**	**Consortium blockchain**	**Private blockchain**
Consensus determination	All miners	Selected set of nodes	Within one organization
Read permission	Public	Public or restricted	Public or restricted
Immutability level	Almost impossible to tamper	Could be tampered	Could be tampered
Efficiency (use of resources)	Low	High	High
Centralization	No	Partial	Yes
Consensus process	Permissionless	Needs permission	Needs permission

The core components of any blockchain architecture are [24]:

- Node: user or computer within the blockchain architecture (each has an independent copy of the whole blockchain ledger)
- Transaction: smallest building block of a blockchain system (records, information, etc.) that serves as the purpose of blockchain
- Block: a data structure used for keeping a set of transactions which is distributed to all nodes in the network
- Chain: a sequence of blocks in a specific order

- Miners: specific nodes which perform the block verification process before adding anything to the blockchain structure
- Consensus (consensus protocol): a set of rules and arrangements to carry out blockchain operations

A blockchain consists of blocks containing details of transactions that occurred. Each of these blocks contains two parts, the header, and the body. The transaction itself is in the body, and the header contains the identifier of the previous block and other fields, making the blocks connected in a chain as shown in Fig 2.2 [2].

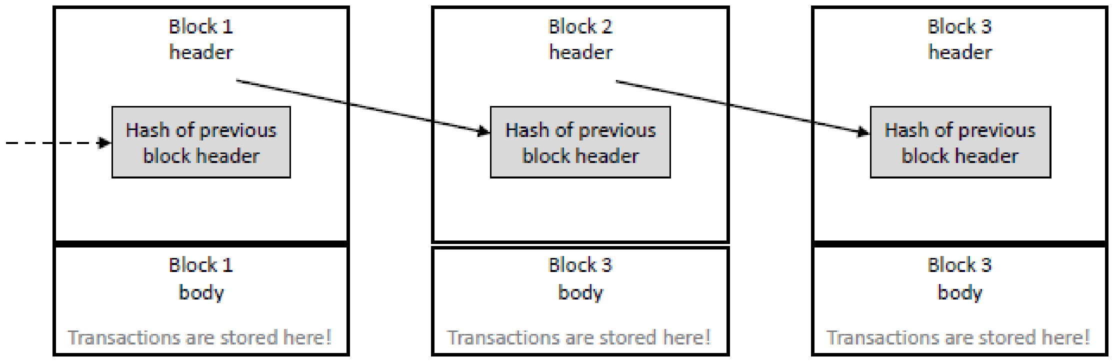

Block 1 header
Hash of previous block header
Block 1 body
Transactions are stored here!
Block 2 header
Hash of previous block header
Block 3 body
Transactions are stored here!
Block 3 header
Hash of previous block header
Block 3 body
Transactions are stored here!

Figure 2.2 – Logical representation of a blockchain. Source [2].

Bitcoin is the most famous blockchain platform but there are others. Authors [40] identified the most popular and most suitable platforms for IoT domains and categorized them accordingly certain characteristics.

Table 2.2 – Blockchain platforms for creating blockchain applications. Adapted from [40]

Platform	Blockchain	Crypto currency	Smart contracts
Ethereum	Public and permission-base	Ether (ETH)	Yes
Hyperledger Fabric	Permission-based	None	Yes
Multichain	Permission-based	Multi-currency	Yes
Litecoin	Public	litecoins (LTC)	No
Lisk	Public and permission-base	LSK	Yes
Quorum	Permission-based	ETH	Yes
HDAC	Permission-based	Multiasset	Yes

2.3 SMART CONTRACT

Nick Szabo defined in 1993 the term Smart Contract as "A computerized transaction protocol that executes the terms of a contract". In other words, we can say it is a code that validates a negotiation and immediately brings a contract into effect, without any intermediaries [16].

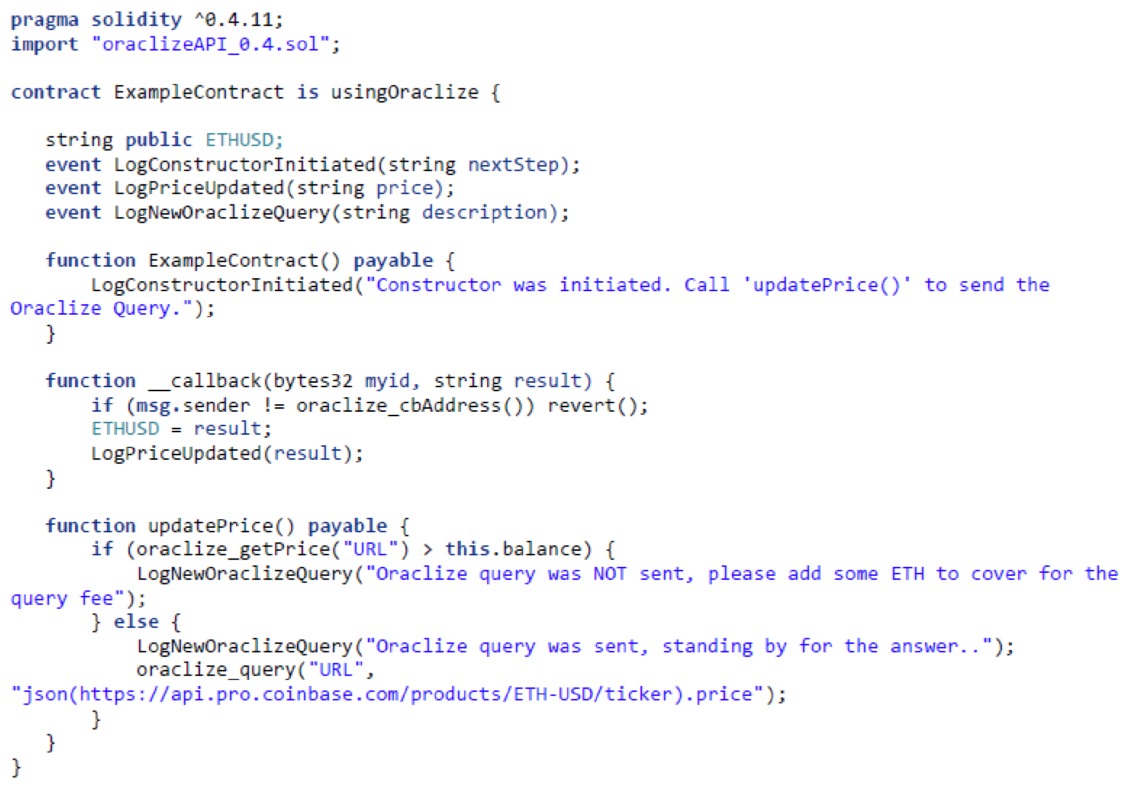
The smart contract is deployed in the blockchain, and each contract is identified by a unique address, and it contains multiple functions that can be called by any user, just sending transactions to this address. One of the most prominent blockchain platforms with smart contract support is Ethereum [7]. It offers a Turing-complete programming language. Developers usually code smart contracts in high-level languages (e.g., Solidity) and compile them in a low-level, stack-based bytecode language, referred to as "Ethereum virtual machine code" or "EVM code".

To deploy a smart contract, the owner sends a transaction to the Ethereum blockchain, with the bytecode. After deployed, any user can invoke the smart contract by sending a transaction whose recipient is the contract.

Smart contracts can access data from real-world events using oracles but it adds more complexity, since authentication, security, and trust in oracles have to be provided [57]. Also, bugs become critical due to the irreversibly and immutable nature of the system. A smart contract once deployed cannot be changed.

The figure 2.3 shows an example of Smart Contract in Solidity language. This contract uses Oraclize to fetch the last ETH/USD from Coinbase Pro APIs. The update process starts every time the function updatePrice() is called. For ease development, it does not charge a contract for its first request of data. The next requests will require the contract to pay the Oraclize fee and the ether necessary to pay for the callback transaction. Both are taken from the contract balance. If the contract does not have enough funds in his balance, the request will fail, and Oraclize will not return any data [32].

In February 2019 the Oraclize price for each URL call request is 0.01 ETHER that is equivalent to US$1.02 as of February 6th 2019.



```
pragma solidity ^0.4.11;
import "oraclizeAPI_0.4.sol";

contract ExampleContract is usingOraclize {

   string public ETHUSD;
   event LogConstructorInitiated(string nextStep);
   event LogPriceUpdated(string price);
   event LogNewOraclizeQuery(string description);

   function ExampleContract() payable {
       LogConstructorInitiated("Constructor was initiated. Call 'updatePrice()' to send the
Oraclize Query.");
   }

   function __callback(bytes32 myid, string result) {
       if (msg.sender != oraclize_cbAddress()) revert();
       ETHUSD = result;
       LogPriceUpdated(result);
   }

   function updatePrice() payable {
       if (oraclize_getPrice("URL") > this.balance) {
           LogNewOraclizeQuery("Oraclize query was NOT sent, please add some ETH to cover for the
query fee");
       } else {
           LogNewOraclizeQuery("Oraclize query was sent, standing by for the answer..");
           oraclize_query("URL",
"json(https://api.pro.coinbase.com/products/ETH-USD/ticker).price");
       }
   }
}
```

Figure 2.3 – Example of Smart Contract with oracles. Source [32].

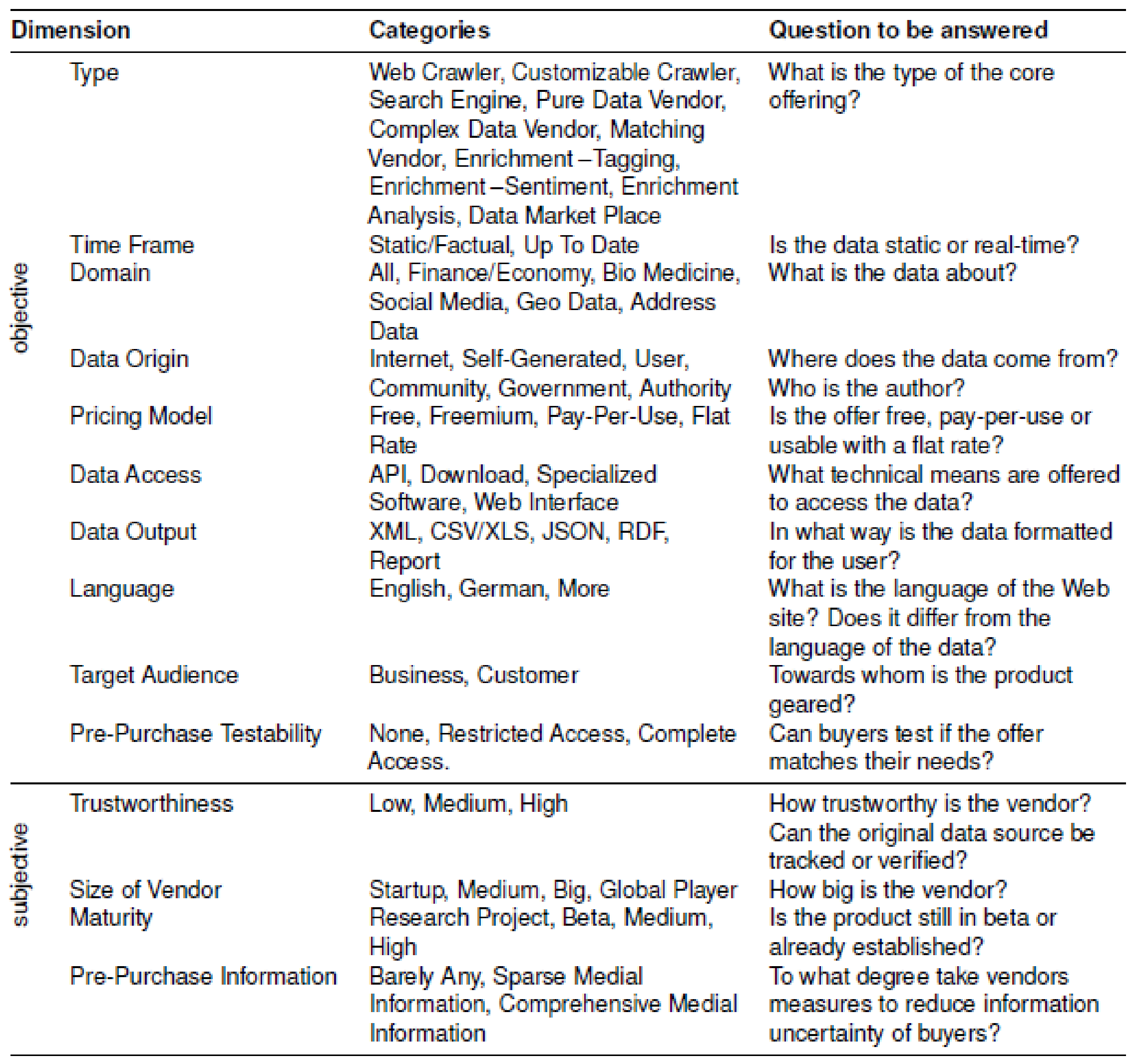
2.4 DATA MARKETPLACE

In a neo-classical economic model, the market is where the interaction of buyers and sellers define the price and the quantity of a good/service, and marketplace is where multiple market participants execute transactions, i.e., it provides the infrastructure for trading [46]. Now, the term data marketplace was probably first used in 1998 by [3]. The authors modeled trading of information between digital libraries, with a focus on the motivation and behavior of participants and identifying factors that affect cooperation [42]. A survey conducted by [45], found the data market and marketplaces can be categorized into twelve dimensions – all listed in Figure 2.4.

The survey shows a lot of different ways the information can be available and charged in a data marketplace. For example, the Pricing Model, a data can be available without charge (Free); available at no cost for a limited amount of time and paid later (Freemium); can make unlimited use for a limited time (Flat Rate). It also shows different formats the data can be available, like XML, CSV, JSON, RDF, and other formats.

Figure 2.5 contains an underlying representation of a data marketplace ecosystem.

- **Data Providers/Sellers:** Individuals/companies that own information and want to make it available for a specific price



	Dimension	Categories	Question to be answered
objective	Type	Web Crawler, Customizable Crawler, Search Engine, Pure Data Vendor, Complex Data Vendor, Matching Vendor, Enrichment – Tagging, Enrichment – Sentiment, Enrichment Analysis, Data Market Place	What is the type of the core offering?
	Time Frame	Static/Factual, Up To Date	Is the data static or real-time?
	Domain	All, Finance/Economy, Bio Medicine, Social Media, Geo Data, Address Data	What is the data about?
	Data Origin	Internet, Self-Generated, User, Community, Government, Authority	Where does the data come from? Who is the author?
	Pricing Model	Free, Freemium, Pay-Per-Use, Flat Rate	Is the offer free, pay-per-use or usable with a flat rate?
	Data Access	API, Download, Specialized Software, Web Interface	What technical means are offered to access the data?
	Data Output	XML, CSV/XLS, JSON, RDF, Report	In what way is the data formatted for the user?
	Language	English, German, More	What is the language of the Web site? Does it differ from the language of the data?
	Target Audience	Business, Customer	Towards whom is the product geared?
	Pre-Purchase Testability	None, Restricted Access, Complete Access.	Can buyers test if the offer matches their needs?
subjective	Trustworthiness	Low, Medium, High	How trustworthy is the vendor? Can the original data source be tracked or verified?
	Size of Vendor	Startup, Medium, Big, Global Player	How big is the vendor?
	Maturity	Research Project, Beta, Medium, High	Is the product still in beta or already established?
	Pre-Purchase Information	Barely Any, Sparse Medial Information, Comprehensive Medial Information	To what degree take vendors measures to reduce information uncertainty of buyers?

Figure 2.4 – Dimensions List. Source [45].

- **Data Marketplace:** The data marketplace is managed by an owner (company) that provides services that allow the integration between Data Providers and Data Users.

  - **Database with metadata:** Contains the data providers description used by the Buyers to find the Sellers.

  - **Data Services:** Services offered that provide secure access, billing management, contract management, data access, and others.

- **Data Users/Buyers:** Individuals/companies that can pay for data that match certain criteria.

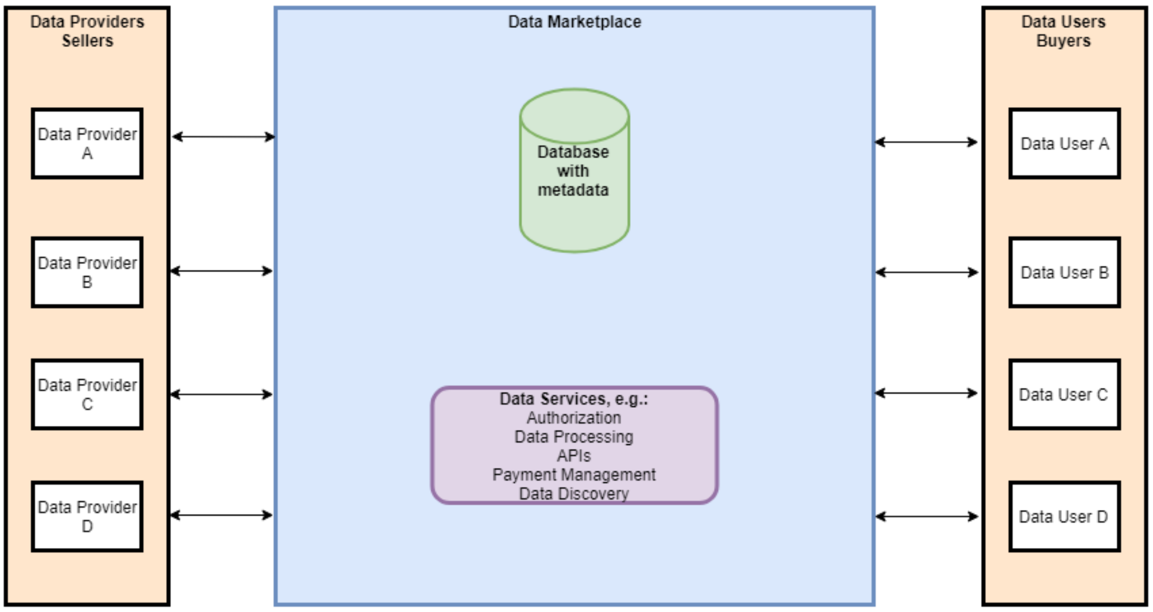


Figure 2.5 – Elements of a data marketplace ecosystem. Adapted from [58].

2.5 MQTT

MQTT stands for MQ Telemetry Transport, and it is a simple and lightweight messaging protocol, useful for devices with low-memory and low-bandwidth available. The protocol is ideal for the IoT world and mobile applications [29]. Dr. Andy Stanford-Clark of IBM invented the protocol in 1999, and version 5 is OASIS standard.

The protocol works in a publish/subscribe model (Figure 2.6).

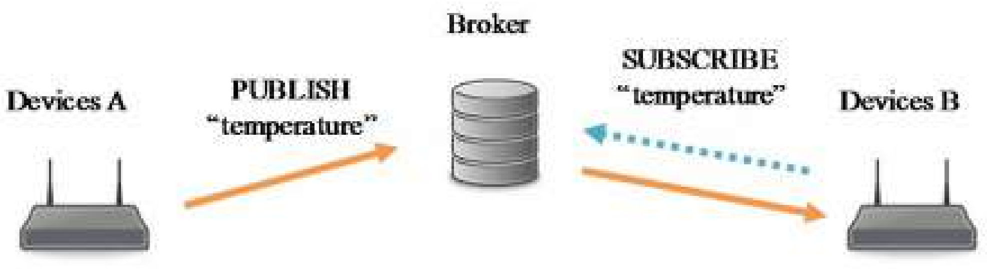


Figure 2.6 – MQTT System [41]

The Devices A publish the message to a topic hosted in a central server (broker). The Devices B can subscribe to specific topics and receives the particular messages delivered.

To guarantee a message is delivered correctly or not, the MQTT defines three levels of Quality of Service (QoS).

QoS 0 - at most once

This level guarantees a best-effort delivery. The recipient does not acknowledge receipt of the message (Figure 2.7) [49].

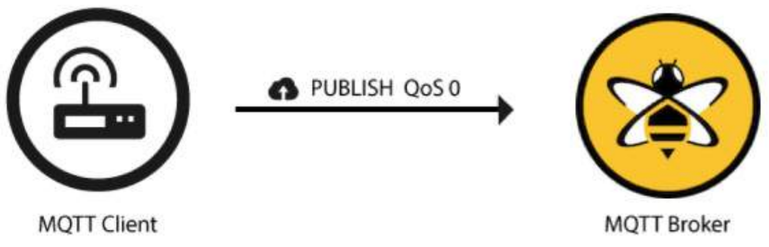


Figure 2.7 – QoS 0 - Source [49]

QoS 1 - at least once

This level guarantees the message is delivered at least one time to the receiver. The sender waits for a PUBACK packet from the receiver (Figure 2.8) [49].

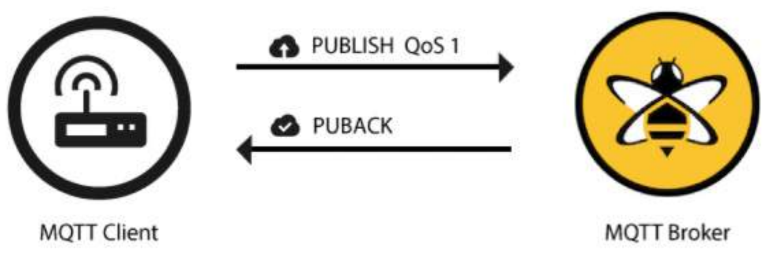


Figure 2.8 – QoS 1 - Source [49]

QoS 2 - exactly once

This level guarantees that each message is received only once by the intended recipients (Figure 2.9) [49].

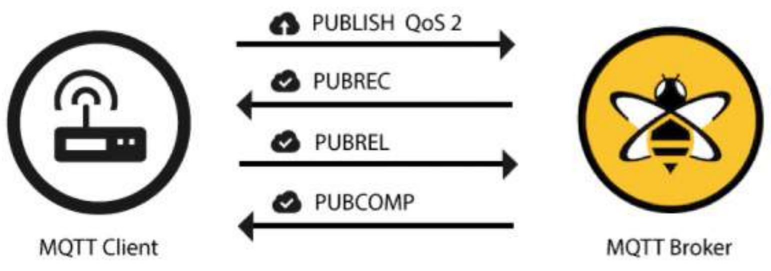


Figure 2.9 – QoS 2 - Source [49]

3. RELATED WORK

A state-of-the-art study regarding existing data monetization architectures focused on IoT devices was conducted.
The table 3.1 lists papers and existing products that explorer the data marketplace for IoT.

Table 3.1 – List of Platforms and Papers

Platform / Paper	Blockchain platform	Public Ledger	Development Level
IOTA's data marketplace [20]	Yes	Tangle	Pre-Product
Wibson's data marketplace [54]	Yes	Ethereum	Product
Project XBR [55]	Yes	Ethereum	Pre-Product
Databroker DAO [12]	Yes	Ethereum	Pre-Product
Datum [14]	Yes	Ethereum	Pre-Product
Weeve [53]	Yes	IOTA, Ethereum, Hyperledger	Pre-Product
Terbine [51]	Yes	Not available	Pre-Product
Datapace [13]	Yes	Hyperledger	Development
Slock [44]	Yes	Not available	Pre-Product
Filament [19]	Yes	Not available	Product
Dajie [11]	Yes	Not available	Product
Mind my value: a decentralized infrastructure for fair and trusted iotdata trading [25]	No	Not applicable	Proof of concept
Towards a Decentralized Data Marketplace for Smart Cities [39]	Yes	Ethereum	Proof of concept
When money learns to fly: Towards sensing as a service applications using bitcoin [30]	Yes	Bitcoin	Theory
Data marketplace for internet of things [27]	No	Not applicable	Proof of concept
Internet of things cloud mediator platform [26]	No	Not applicable	Proof of concept
MARSA: A Marketplace for Realtime Human Sensing Data [8]	No	Not applicable	Proof of concept
Valorising the IoT Databox: creating value for everyone [36]	No	Not applicable	Proof of concept
On Supporting Contract-aware IoT Dataspace Services [4]	No	Not applicable	
IDMoB: IoT Data Marketplace on Blockchain [33]	Yes	Ethereum	Proof of concept
A Peer-to-Peer Architecture for Distributed Data Monetization in Fog Computing Scenarios [15]	Yes	Hyperledger, Tangle	Proof of concept
Monetization of IoT data using smart contracts [47]	Yes	Ethereum	Proof of concept

3.1 INDUSTRIAL PLATFORMS

IOTA [20] is a distributed ledger that it is based on a directed acyclic graph consensus structure called the 'Tangle' rather than a blockchain. Instead of requiring special participants—miners—to perform computational proof-of-work and validate blocks of transactions in exchange for newly minted tokens, network participants themselves perform consensus by validating two previous transactions each time they wish to make a transaction [50]. At the time this work was done the platform was still under testing. It was possible to get data from devices, but no real-world payments were implemented.

The XBR Project [55] is also under development with a planned go-live on June 2021. It aims at monetizing the access to data services via API. The payments are performed using an XBR cryptographic token based on the Ethereum blockchain [34]. With the promise to be released this year, Databroker DAO [12] is another marketplace to buy and sell sensor data. It is also based in the Ethereum network and has his unique tokens named DTX that will serve as the credit to buy and sell sensor data within the platform.

The Datum project is still under development but is another platform for data storage, and uses BigchainDB and IPFS together with smart contracts and blockchain, allowing data sharing and a marketplace, enabling individuals to monetize their data on their terms [21]. Weeve envisions the evolution of the IoT into the Economy of Things having an unfalsifiable, scalable and secures data marketplace supported by its platform. The marketplace enables IoT devices to index, process, and trade digital assets in a verifiable manner [34]. It is also developing a new communication protocol for IoT devices called TEE-MQTS, that extends MQTT protocol with new cryptographic protections and is creating a new operation system called Weeve OS for IoT devices [52]. A platform that is in alpha phase is Terbine [51], providing IoT data generated by public agencies, universities, and corporations around the world. It will provide a marketplace using blockchain-based transaction logging and tracking. Datapace is a platform under development that will provide a global-scale marketplace for IOT sensor data. It has his token named TAS and will use Cosmos[1] for interoperability with other blockchains [13]. Besides that, it uses Mainflux[2] as IoT cloud platform which enables IoT sensor and gateway connectivity and management.

Other platforms like Dajie[11] uses the blockchain and the IoT for management energy produced in local homes, allowing people to exchange energy peer to peer in a local neighborhood at a better price than what they can get feeding that energy back to the grid. The Filament[19] company offers hardware, software, training, to allow the connectivity between IoT devices and distributed ledger.

A specific data marketplace for personal information is Wibson[54]. The users install a smart phone application and can share some personal data, like Facebook account and mobile device location, and sell the data in exchange for Wibson tokens.

3.2 ACADEMIC PROJECTS

The work presented in [30] explains the Sensing-as-a-Service concept and the benefits of using Bitcoin in the data integration and payment. It is a purely theoretical work, with the essential ideas: the data requestor sent the request using the blockchain platform and the sensor reply the data back, all using the blockchain. The authors also highlight the problems of this implementation and suggest the data sensors information be hosted in a decentralized server and not in the blockchain.

[1]https://cosmos.network/
[2]https://www.mainflux.com

Authors in [27] envision a cloud application where device owners register their sensors along with standardized device data (like what they measure, location information) and the data consumers query the system and retrieve the data. The authors show a diagram (figure 3.1) with the proposed architecture and also evaluated the device owners profit with the pricing of measurements, application acquisition rates, and query performance.

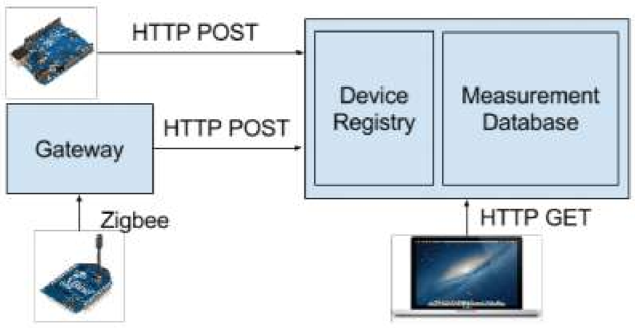


Figure 3.1 – System Architecture. Source [27]

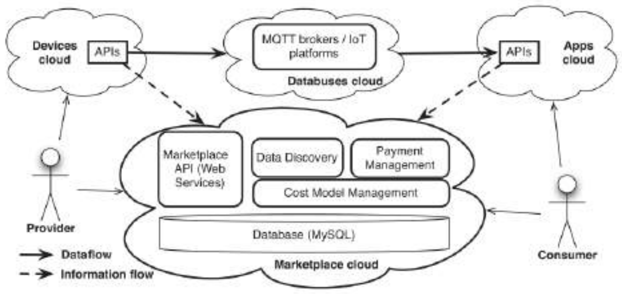


Figure 3.2 – MARSA Prototype. Source [8]

A similar architecture is defined by [26] with the addition of a module responsible for performing the negotiation between the data buyers and sellers, but the authors emphasize how the proposed platform should behave with a considerable amount of devices. The application architecture is presented in figure 3.3.

Another architecture is developed by [8] in the MARSA: A Marketplace for Realtime Human Sensing Data, including a prototype of the architecture (see figure 3.2). The authors present techniques for selecting data types and managing data contracts based on different pricing models, quality of data and incentive models. They also full implemented the Data Discovery and Cost Model Management making the code source available.

A business model target that explores the usage of blockchain and smart contracts transactions for IoT data trading is explored by [25]. Authors perform some experimental evaluation and describe the challenges and lessons learned in using such technologies.

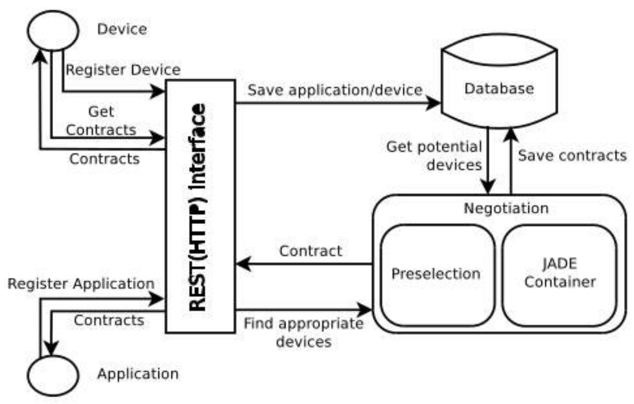


Figure 3.3 – Architecture overview. Source [26]

Motivated by the potential valorization opportunities of personal data, the authors on [36] define the term Databox as a protective container for personal data where data may locate in different geographical location. The Databox is the system (can be physical or not) responsible for handling the interactions between the Data Owner (usually a person) and the Data Buyers (companies that want to buy the data). The major phases in a Databox's life cycle are despite in the figure 3.4 and explained in the original paper.

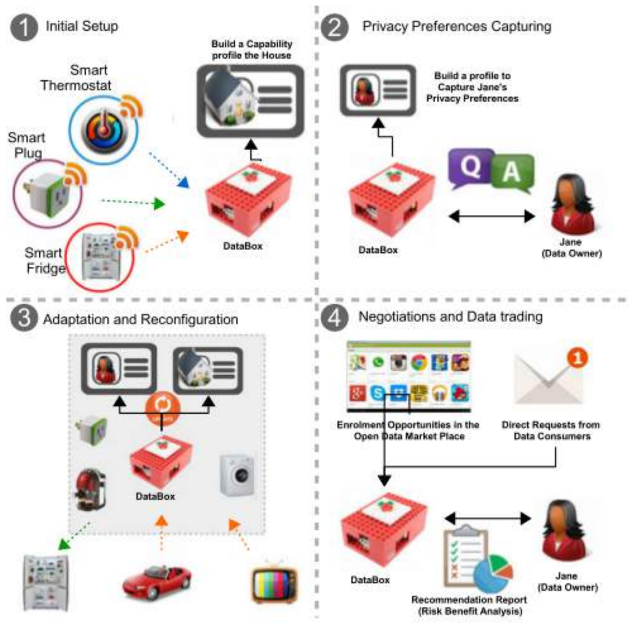

1 Initial Setup
Build a Capability profile the House
Smart Thermostat
Smart Plug
Smart Fridge
DataBox
2 Privacy Preferences Capturing
Build a profile to Capture Jane's Privacy Preferences
DataBox
Q A
Jane (Data Owner)
3 Adaptation and Reconfiguration
DataBox
4 Negotiations and Data trading
Enrolment Opportunities in the Open Data Market Place
Direct Requests from Data Consumers
DataBox
Recommendation Report (Risk Benefit Analysis)
Jane (Data Owner)

Figure 3.4 – Major phases in a Databox's life cycle. Source [36]

Moved by the limitations that exist in current data contract management, authors [4] developed a new extensible framework for data contract management, which takes into account both technical and business aspects to allow the monitoring of individual IoT data contracts. Focused on Data contract elements, like Data Rights, Quality of Data, Quality of Service, Pricing Model, Purchasing Policy, Control and Relationship, the authors implemented a prototype (following the architecture from figure 3.5) and made the code available.

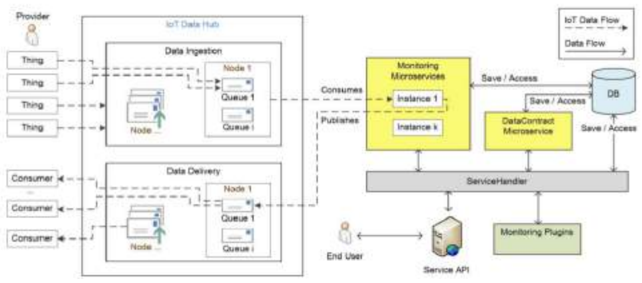


Figure 3.5 – Architecture of Contract-aware Dataspace Service Platform. Source [4]

Using the benefits or peer-to-peer platforms like Ethereum and Swarm authors [33] developed a proof-of-concept data marketplace with the code available on Github. Smart Contracts deployed in Ethereum platform performs the data access validation, and the content is accessible on Swarm. The core functions description is available. A data flow sequence envisioned is presented in figure 3.6.

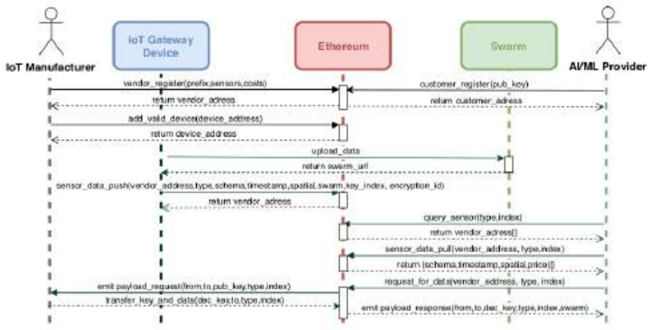

IoT Manufacturer
IoT Gateway Device
Ethereum
Swarm
AI/ML Provider
vendor_register(prefix,sensors,costs)
return vendor_adress
customer_register(pub_key)
return customer_adress
add_valid_device(device_address)
return device_address
upload_data
return swarm_url
sensor_data_push(vendor_address,type,schema,timestamp,spatial,swarm,key_index, encryption_id)
return vendor_adress
query_sensor(type,index)
return vendor_adress[]
sensor_data_pull(vendor_address, type,index)
return (schema,timestamp,spatial,price)[]
request_for_data(vendor_address, type, index)
emit payload_request(from,to,pub_key,type,index)
transfer_key_and_data(dec_key,to,type,index)
emit payload_response(from,to,dec_key,type,index,swarm)

Figure 3.6 – Data Flow Sequence Diagram. Source [33]

The work proposed by [39] uses IPFS[6] as the peer-to-peer platform to store the data generated by the IoT devices, and implement other mechanisms like a Data Quality mechanism that allows buyers and sellers to rate each other.

Authors in [15] presents a distributed peer-to-peer data marketplace in a fog computing scenario, using FIWARE technology for the fog nodes and blockchain for the communication. There are two communication layers: one for the data and the business layer. Each layer uses different ledgers: the business layer uses Hyperledger, and the data layer uses IOTA. The authors don't provide any source code.

Other author like [47] also uses Smart Contract for the data monetization. In their paper, the Ethereum platform is used to demonstrate the integration between Device owner and Customers. The system overview is available in the figure 3.7 and the authors published their code.

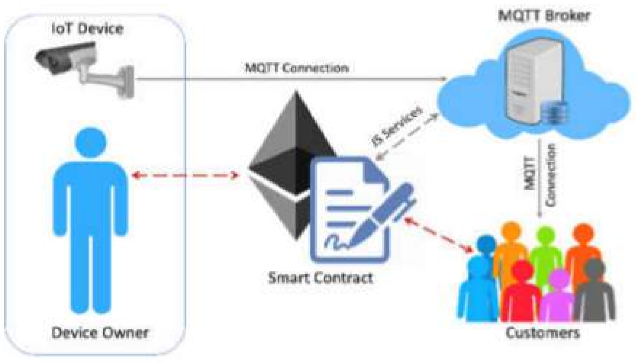


Figure 3.7 – A system overview for automating the monetization of IoT data using Ethereum Smart Contracts. Source [47]

3.3 OUR PROPOSAL

Table 3.2 contains some of the features available or evaluated in the commercial platforms and papers.

Commercial projects like [20], [54], citexbr and [12], released new cryptocurrencies to support their initiatives. It is a valid approach, but, at the same time, the success of the product will be linked directly with the success of the coin. Using an existing and well-established cryptocurrency like Bitcoin or Ether reduces this risk and is the strategy used by this proposal and by [33], [47], [25], [30].

Another critical feature a data market needs to handle is how to exchange the money between all the involved players. Most of the academic papers like [26], [27] , [8], [36] and[4], don't make any reference for financial side. The current proposal defines how the money exchange will happen (using Ethereum Smart Contracts and Ether has the cryptocurrency).

With just a few exceptions ([14], [51], [33], [30]), the authors add support for real-time messages in their proposals but, there are no references related with the delay that exists between the data generation and the data consumer. This proposal tackles this area together with an important one, the Data Matching mechanism.

Only one article [26] references a data matching mechanism, using autonomous agents, and in their model, the buyer makes agreements with specific sellers. In our model, the agreement is to the data type. The proposed architecture will try to deliver the message desired and, it can be for different vendors along the time.

Table 3.2 – List of Platforms and Papers comparison

Platform	Cryptocurrencie	Money exchange mecanism defined	Real-time Support	Auto Data Matching	Performance Evaluation	Year
Solution Proposal	ETHER	Yes	Yes	Yes	Yes	2019
IOTA's data marketplace [20]	New	Yes	Yes	Not available	Not available	2015 /
Wibson's data marketplace [54]	New	Yes	Yes	No	Not available	2018
Project XBR [55]	New	Yes	Yes	Not available	Not available	2021
Databroker DAO [12]	New	Yes	Yes	No	Not available	2018
Datum [14]	New	Yes	No	Not available	Not available	2017 / 2021
Weeve [53]	New	Yes	Yes	Not available	Not available	2017
Terbine [51]	Not Available	Not Available	No	No	Not available	2016
Datapace [13]	New	Yes	Yes	No	Not available	2017
Dajie [11]	Not Available	Yes	Yes	Not applicable	Not available	Not available
IDMoB: IoT Data Marketplace on Blockchain [33]	ETHER	Yes	No	No	No	2018
A Peer-to-Peer Architecture for Distributed Data Monetization in Fog Computing Scenarios [15]	Any	Multiple	Yes	No	No	2018
Monetization of IoT data using smart contracts [47]	ETHER	Yes	Yes	Not applicable	No	2018
Towards a Decentralized Data Marketplace for Smart Cities [39]	Any	Yes	Yes	No	Not applicable	2018
Mind my value: a decentralized infrastructure for fair and trusted iot data trading [25]	ETHER	Yes	Yes	No	Not applicable	2017
On Supporting Contract-aware IoT Dataspace Services [4]	Not applicable	No	Yes	No	Yes	2017
Valorising the IoT Databox: creating value for everyone [36]	Not applicable	No	Yes	No	No	2016
MARSA: A Marketplace for Realtime Human Sensing Data [8]	Not applicable	No	Yes	No	Yes	2016
Data marketplace for internet of things [27]	Not applicable	No	Yes	No	Yes	2016
When money learns to fly: Towards sensing as a service applications using bitcoin [30]	Bitcoin	Yes	No	Not applicable	No	2014
Internet of things cloud mediator platform [26]	Not applicable	No	Yes	Yes	No	2014

4. SYSTEM ARCHITECTURE

In this chapter, the characteristics and definitions of the data marketplace platform of this work are presented. The developed Data Marketplace Platform has the modules and interactions depicted by Figure 4.1.

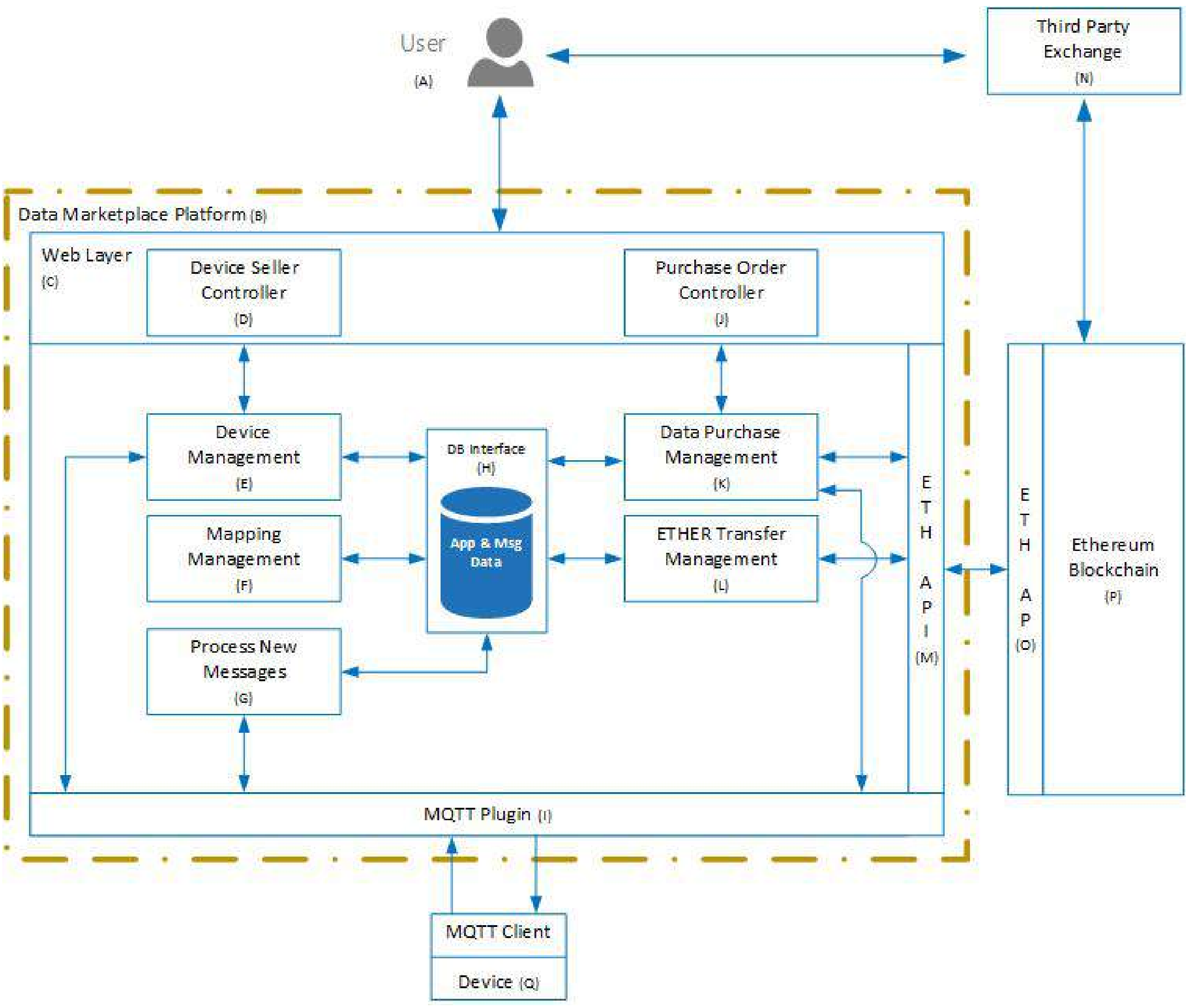


Figure 4.1 – System Architecture

The system is divided into sellers and buyers. If you want to sell data, you need to register the device that contains the data at the marketplace platform, as well as the account that you desire to receive the incomings. The procedures are similar in case you wish to buy data; you must register the type of data you want to buy. Next paragraphs explain these procedures.

The first step consists of a User (Figure 4.1.A) interacting with the platform throw the web pages or APIs (Figure 4.1.C). The next step is to register a device for selling data (Figure 4.1.D) or register a Purchase Order for buying data (Figure 4.1.J).

To register a Device for selling data, the User needs to inform (throw the Device Seller Controller - Figure 4.1.D) a Name for the device, a friendly Description, the Ethereum Account that wants to receive the credits for the data selling, the type of data that will sell and the amount in Gwei[1] that want per message sold. Also, it is necessary to inform if the device is fixed or mobile. If fixed, need to specify the GPS location and, if mobile, needs to inform the geographical area the device will be in 90% of the time (example, the city or the country). The Device Management module (Figure 4.1.E) collects all these data and saves it into the internal database (Figure 4.1.H). At this time, the device has an internal ID, and the database module sent back the ID to the Device Management module. With that information, the MQTT Plugin is executed (Figure 4.1.I), and creates an user specifically for this device, setting the password and the topic permissions. Subsequently, the user receives back from the Device Seller Controller (Figure 4.1.D) the MQTT credentials and the message template to be used when delivering the data.

In order to register a Purchase Order, the user interacts with the Purchase Order Controller (Figure 4.1.J), passing a Name for the Purchase Order, a friendly Description, and the characteristics of the messages that want to receive. The Data Purchase Management module (Figure 4.1.K) receives all these data and saves it into the internal database (Figure 4.1.H). Similar to what happens in the Device Registration process, the Purchase Order now has an internal ID, and an MQTT account is set. Later, the module calls the ETH API plugin (Figure 4.1.M) and with the information provided by the User, deploys a unique Smart Contract in the Ethereum Blockchain (Figure 4.1.P) for the Purchase Oder. Afterward, the User receives back from the Purchase Order Controller (Figure 4.1.J) the MQTT credentials and the message template to be used when receiving the data. Additionally, the User receives the Smart Contract address that he should transfer Ether, using a Third Party Exchange (Figure 4.1.N). The system will only deliver messages to the User if there are enough credits in the Smart Contract address. Inside the Data Purchase Management module (Figure 4.1.K) there is a function that executes every 2 hours and updates the Smart Contract balance in the internal database (Figure 4.1.H) with the amount from the Ethereum Blockchain (Figure 4.1.P) (see section 4.3.2).

After we have buyers and sellers in the system, the Mapping Management module (Figure 4.1.F) runs looking for potential sellers for each active Purchase Order (see section 4.4). If a matching happens, the Platform creates a mapping between the Purchase Order and the Seller and save it in the internal database (Figure 4.1.H)

The Device (Figure 4.1.Q) is the IoT equipment owned by the User, and it requires an MQTT Client installed to deliver or to receive the data messages.

The platform receives (Figure 4.1.I) and process (Figure 4.1.G) every message sent by the Sellers. For each message, the Process New Messages module (section 4.5) checks at the database (Figure 4.1.H) the existing mapping between Purchase Order and

[1]Gwei is a denomination for $10^{-9} Ether. 1 Gwei = 0.000000001 Ether$

Seller. For each mapping found, the module evaluates if the message should be delivered or not. If positive, the module calls the MQTT Plugin (Figure 4.1.I) publishing the message to the Purchase Order MQTT Topic where the Buyer Device (Figure 4.1.Q) can consume. Afterward, the module updates inside the database (Figure 4.1.H) the amount pending to transfer to the seller and the Platform (Sales Commission).

The Ether Transfer Management module (Figure 4.1.L) checks every 12 hours the amounts recorded in the internal database (off-chain) (Figure 4.1.H) that should be processed on-chain. The module calls the ETH API (Figure 4.1.M) and transfers from the Smart Contract to the Seller and Platform wallets only if some criteria are meet (section 4.6).

In this work, a private Ethereum Blockchain network (Figure 4.1.P) is in use. A node running Go Ethereum version 1.8.27 is setup which provides all necessary APIs (Figure 4.1.O).

4.1 Web Layer

The Web Layer is responsible for the interaction between the User and the platform. A User can manage a Seller Device or can administer a Purchase Order (a request to buy some data). All these interactions can happen through web pages or APIs. The Web Layer is develop using Microsoft ASP .Net Core and hosted as an App Service in Microsoft Azure platform. A GitHub project [2] was used. The basic functionalities of a web site platform (user registration, user permissions) are already there.

The APIs definition created for the platform is available in Appendix A.

4.1.1 Device Seller Controller

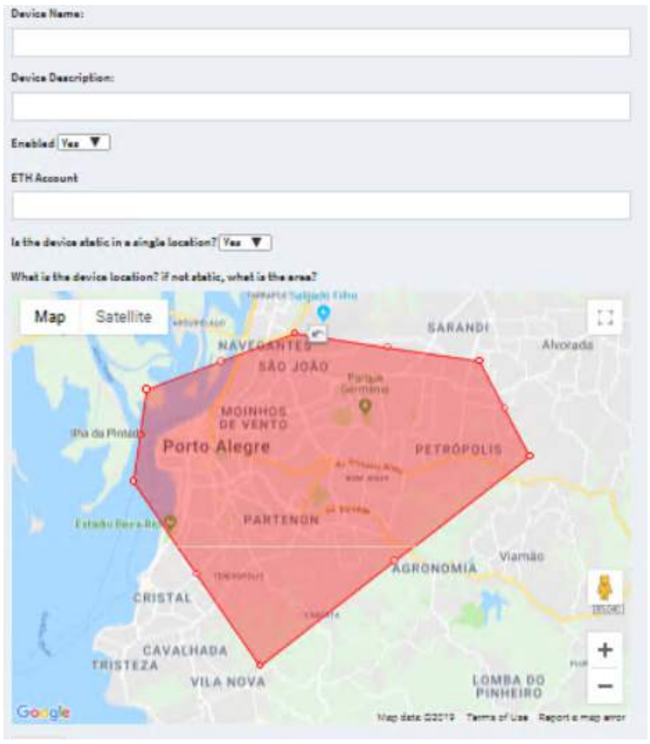
The Device Seller Controller contains the pages (and APIs) the user interacts with to register a new Seller device. The main two actions involved are the Device register itself and the process to associate the messages it can deliver.

Seller Device Registration

Figure 4.2 shows the page the User needs to fill when registering a new Device. The User should give a Device Name, a Device Description, indicates if the Device is Enabled or Not and, the ETH Account that wants to receive the ETHER credits. It also needs

[2]https://github.com/go2ismail/Asp.Net-User-Role-Membership-Example

to indicate if the Device is fixed in a single location or is mobile. Further, needs to indicate in the Google Maps the Device location (in case static) or the area the Device can be.


Device Name:

Device Description:

Enabled Yes ▼

ETH Account

Is the device static in a single location? Yes ▼

What is the device location? If not static, what is the area?

Figure 4.2 – Seller Device Registration Form

After submitting this form, the User receives back the MQTT information (user name, password, topic) to be used when delivering the messages.

Seller Device Message Association

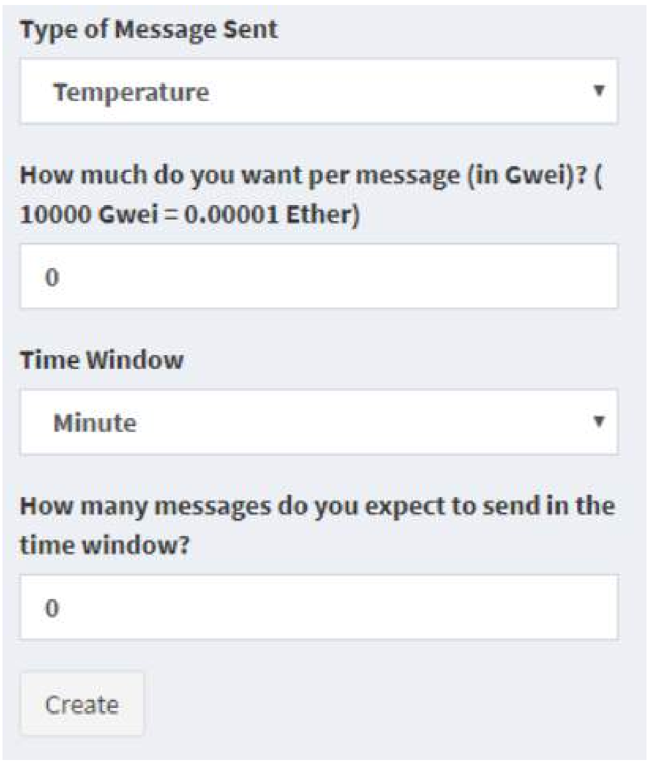
The User should associate one or more message types to the Seller Device. A message type defines what kind of information the User would like to sell. Currently, the following message types are in the platform:

- Application Usage - Facebook: Some applications monitor how many hours a user spent in a mobile app. It is possible to sell, for example, how many hours I utilized Facebook during a specific range of dates.
- Application Usage - WhatsApp
- Lumen
- People Location
- Temperature
- Voltage

Figure 4.3 shows the page the User needs to fill when associating a message type with a Device. Besides the message type, it needs to define the amount in Gwei that want to receive per message sold.

After submitting this form, the User receives back the JSON message template to be used when delivering the messages.

Appendix C contains more explanation about each message type.



Type of Message Sent

Temperature ▾

How much do you want per message (in Gwei)? (10000 Gwei = 0.00001 Ether)

0

Time Window

Minute ▾

How many messages do you expect to send in the time window?

0

Create

Figure 4.3 – Seller Device Message Association Form

4.1.2 Purchase Order Controller

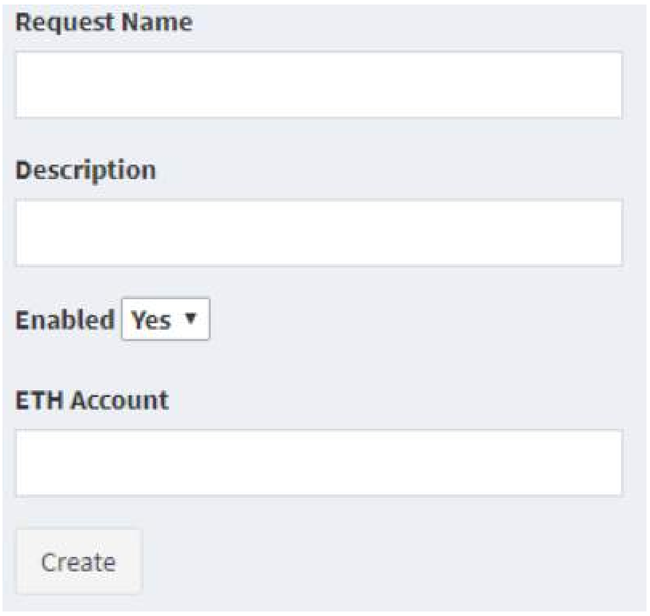
The Purchase Order Controller contains the pages (and APIs) the user interacts with to register a new Purchase Order (a new request to buy data). The user records the PO first and later associate the message types that want to receive.

Purchase Order Registration

When a User wants to buy some data, it goes to a form (Figure 4.4) and fills with a Name, a friendly Description, indicates if the request is Active or not, and the Ethereum account that will receive any Ether back when the Purchase Order ends.

After submitting this form, the User receives back the MQTT information (user name, password, topic) to be used for receiving the messages. Also, it receives the Smart

Contract address. Using a third party exchange, the user needs to transfer Ether to this address; otherwise, the platform will not deliver any message.


Request Name

Description

Enabled: Yes

ETH Account

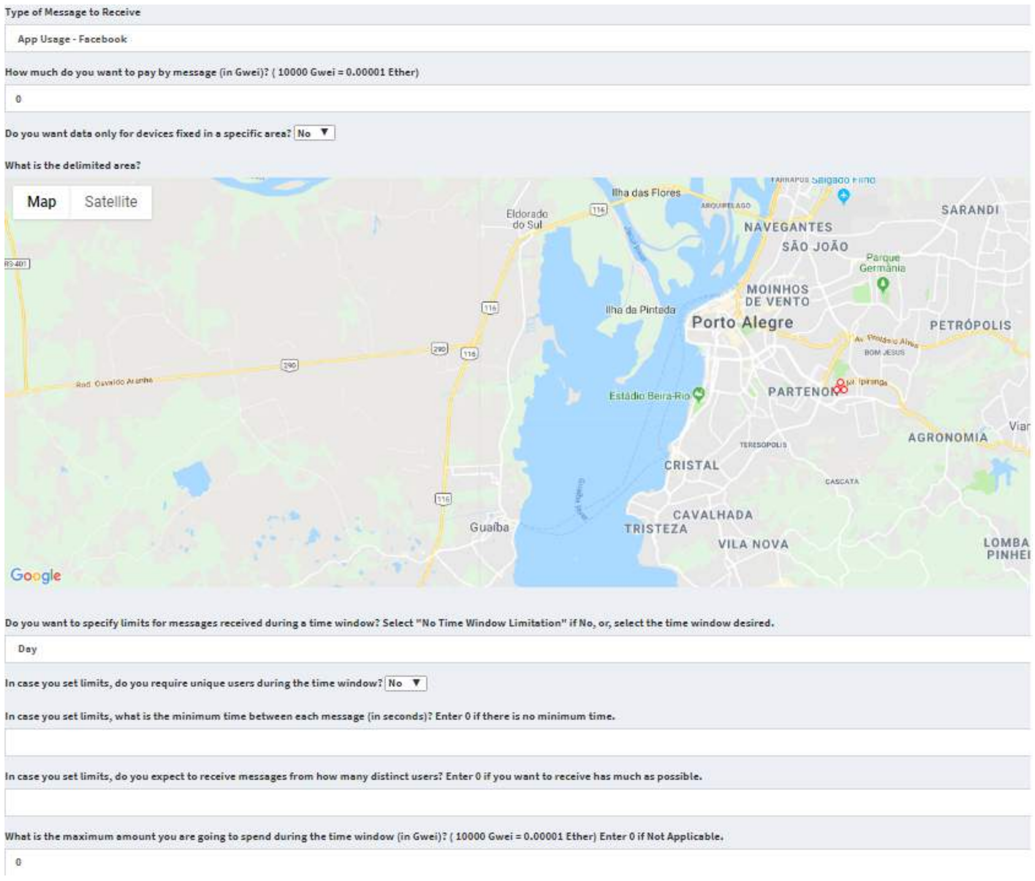
Figure 4.4 – Purchase Order Registration Form

Purchase Order Message Association

After the PO is created, the user can associate the messages want to receive. First, it is necessary to select the message type (same as from section 4.1.1) and define how much in Gwei wants to pay per message delivered. The Gwei amount specified is the maximum amount, and, the platform can deliver messages that cost up to this value (seller price plus platform commission). After defining the price, the Buyer should specify other settings:

- Delimited area of interest: the Buyer can set in the Google Maps (or throw GPS coordinates) the region that wants the messages. It can be a street, a city, an entire country or all the world;
- Static or mobile devices: if a region of interest is delimited, the Buyer can specify if he wants the messages from the devices that are installed in that area (like a fixed thermometer) or can be from mobile devices;
- Minimum time between messages: the Buyer can specify if he wants all messages that meet the selected criteria or if it should exist a minimal interval between each one;
- Limits for a specific Time Window: it is possible to define a time window (Minute, Hour, Day) and add additional restrictions:

- Can set to receive only messages from unique users. For example, if the time window is an hour and a Seller sent two messages, the Buyer will only receive the first one;
- Define the maximum number of unique users. For example, if the time window is an hour and the maximum number of unique users is 10, only the messages from the ten first users are delivered;
- Define the maximum spent amount. For example, if the time window is one day and the maximum amount is one Ether, once this amount is reached, no new message will be delivered until the next day.


Type of Message to Receive

App Usage - Facebook

How much do you want to pay by message (in Gwei)? (10000 Gwei = 0.00001 Ether)

0

Do you want data only for devices fixed in a specific area? No ▼

What is the delimited area?

Do you want to specify limits for messages received during a time window? Select "No Time Window Limitation" if No, or, select the time window desired.

Day

In case you set limits, do you require unique users during the time window? No ▼

In case you set limits, what is the minimum time between each message (in seconds)? Enter 0 if there is no minimum time.

In case you set limits, do you expect to receive messages from how many distinct users? Enter 0 if you want to receive has much as possible.

What is the maximum amount you are going to spend during the time window (in Gwei)? (10000 Gwei = 0.00001 Ether) Enter 0 if Not Applicable.

0

Figure 4.5 – Purchase Order Message Association Form

4.2 Device Management

Device Management (Figure 4.1.E) is responsible for receiving the information entered by the Users in the Web Layer related with the Seller Devices (section 4.1) and processing it. The primary goal is to register the Seller Device into the platform. Figure 4.6 shows this process.

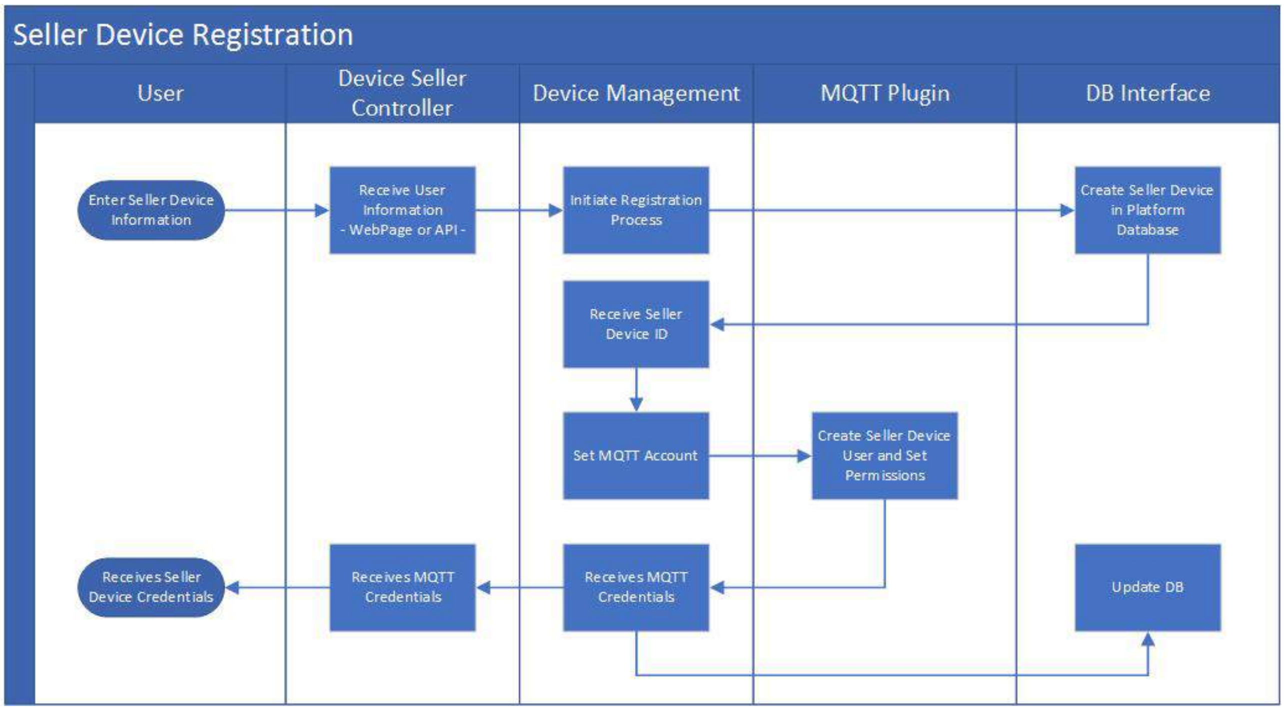

Seller Device Registration
User
Device Seller Controller
Device Management
MQTT Plugin
DB Interface
Enter Seller Device Information
Receive User Information - WebPage or API -
Initiate Registration Process
Create Seller Device in Platform Database
Receive Seller Device ID
Set MQTT Account
Create Seller Device User and Set Permissions
Receives Seller Device Credentials
Receives MQTT Credentials
Receives MQTT Credentials
Update DB

Figure 4.6 – Seller Device Registration

A User submits in the web page (or using the API) the Seller Device Name, the Description, and the Ethereum Account to receive the credits. The Device Management module initiates the process to register the Seller Device into the platform. With this information, the Device Management module creates the device at the database level. Once created, the device receives a unique identifier and, with this identifier, an MQTT account is set up at the RabbitMQ platform (MQTT APIs information in Appendix B). The MQTT user name starts with "prod_" and the MQTT Topic name to deliver the messages starts with "inData/prod_.", appending the Seller Device unique identifier to the names. Following, the User receives the MQTT credentials and topic name.

After the Seller Device is created in the platform, the User can associate one or more messages type the device can deliver. Figure 4.3 contains the process. First, the User requests the list of available message types that are categorized. The list is at the database level and is available in Appendix C. Once the User receives the list, he selects the message type that his Seller Device can deliver and set the price (in Gwei) he wants to receive per message sold. Later, the platform receives back the message type and price fixed by the User and save it at the database. Subsequently, the database level sends back the JSON message template the User should use when delivering the messages.

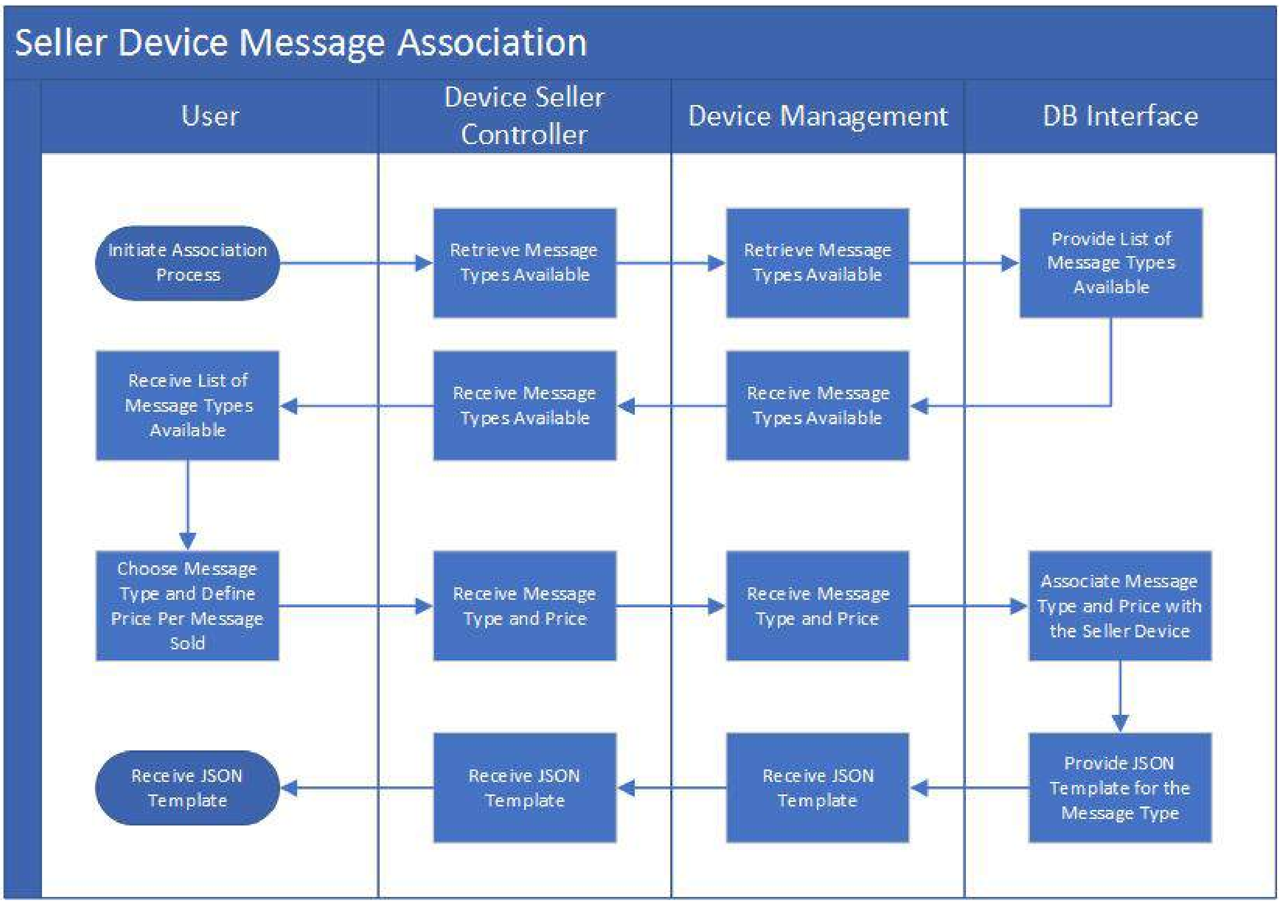

Seller Device Message Association
User
Device Seller Controller
Device Management
DB Interface
Initiate Association Process
Retrieve Message Types Available
Retrieve Message Types Available
Provide List of Message Types Available
Receive List of Message Types Available
Receive Message Types Available
Receive Message Types Available
Choose Message Type and Define Price Per Message Sold
Receive Message Type and Price
Receive Message Type and Price
Associate Message Type and Price with the Seller Device
Receive JSON Template
Receive JSON Template
Receive JSON Template
Provide JSON Template for the Message Type

Figure 4.7 – Seller Device Message Association

4.3 Data Purchase Management

When a User wants to buy some data, the Data Purchase Management module (Figure 4.1.K) handles the process which Figure 4.8 has the details.

The Purchase Order (PO) starts when the User submits the interest to buy some data. The Data Purchase Management module initiates the process after the Buyer provides a Name, a Description, and the Ethereum Account to receive any remain balance when the PO closes. With this information, it creates a PO ID at the database and, similar to what occurs with the Seller Device, it calls the MQTT API and creates a user and a topic specific for the PO. The topic will be used to receive the data. After creating the MQTT user, the platform deploys a Smart Contract in the Ethereum Blockchain. The Smart Contract code and how to interact with the Ethereum blockchain is available in Appendix D. Each PO has a unique Smart Contract associated. The Smart Contract address and MQTT information are stored at the database and shared with the User. The User now knows where to receive the data (MQTT) and the Ethereum address he should transfer amounts to if he wants to start receiving any message. You can imagine that the PO/Smart Contract

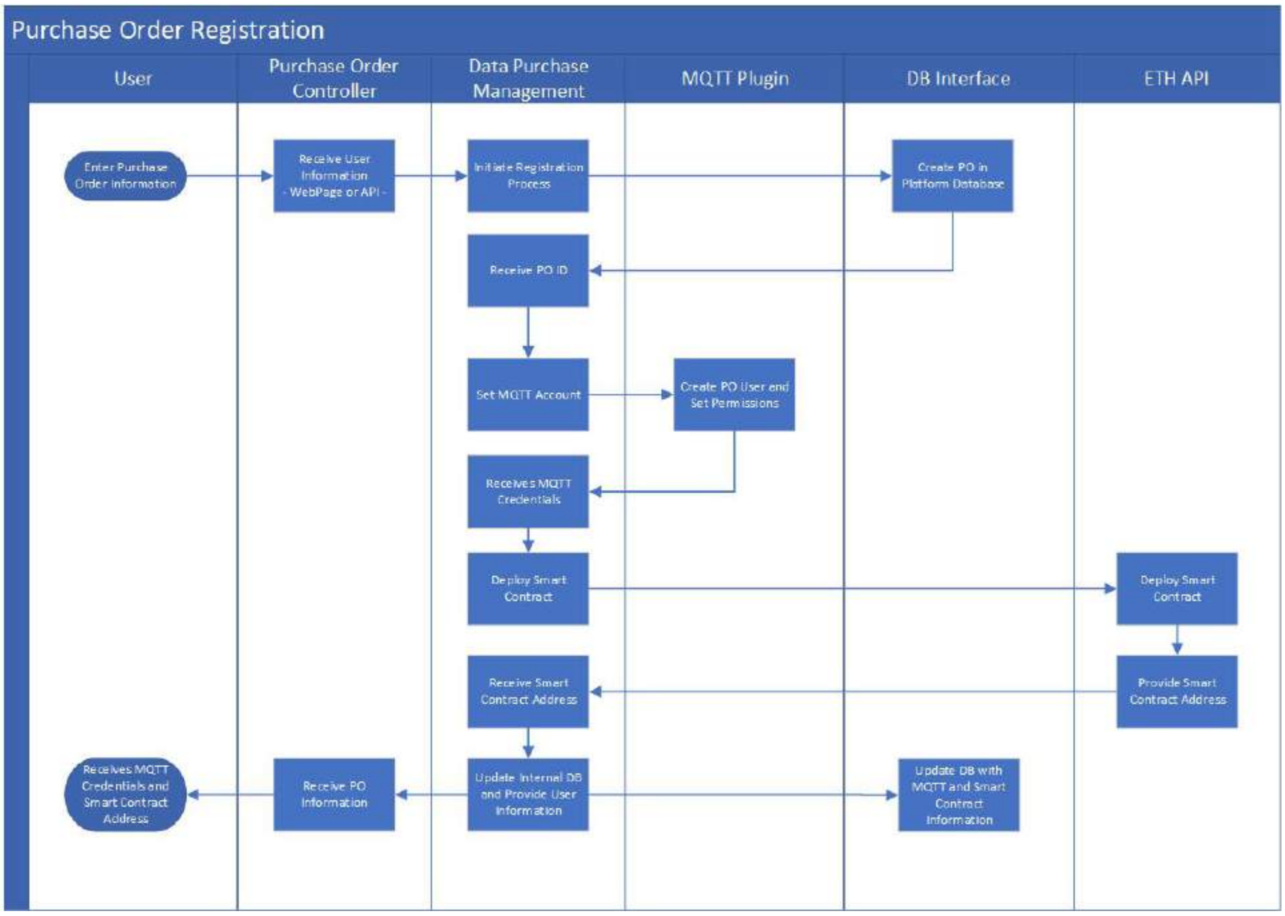

Purchase Order Registration
User
Purchase Order Controller
Data Purchase Management
MQTT Plugin
DB Interface
ETH API
Enter Purchase Order Information
Receive User Information - WebPage or API -
Initiate Registration Process
Create PO in Platform Database
Receive PO ID
Set MQTT Account
Create PO User and Set Permissions
Receives MQTT Credentials
Deploy Smart Contract
Deploy Smart Contract
Provide Smart Contract Address
Receive Smart Contract Address
Update Internal DB and Provide User Information
Update DB with MQTT and Smart Contract Information
Receive PO Information
Receives MQTT Credentials and Smart Contract Address

Figure 4.8 – Purchase Order Registration Process

is a pre-paid cellphone. Once created, the User should transfer funds using a Third Party Exchange (Figure 4.1.N) and, the platform will manage these funds.

After the PO is created, the user can associate different message types (Appendix C) and, for each message type, specify a sort of settings (section 4.3.1).

4.3.1 Define PO Message Type

After creating a PO, the User should define the type of messages that want to receive (and pay for). Figure 4.9 illustrate the process.

The Buyer receives from the platform the list of message types available and, after selecting the message type, another set of definitions are necessary. Figure 4.10 contains the decisions the Buyer should take.

The obvious question is to define how much the user wants to pay per message. After that, the user needs to define if want to receive messages for devices located in a specific region or not. If yes, when in the platform web site, can select the region in a Google Maps (using the API should provide GPS coordinates). When defining a region, it is

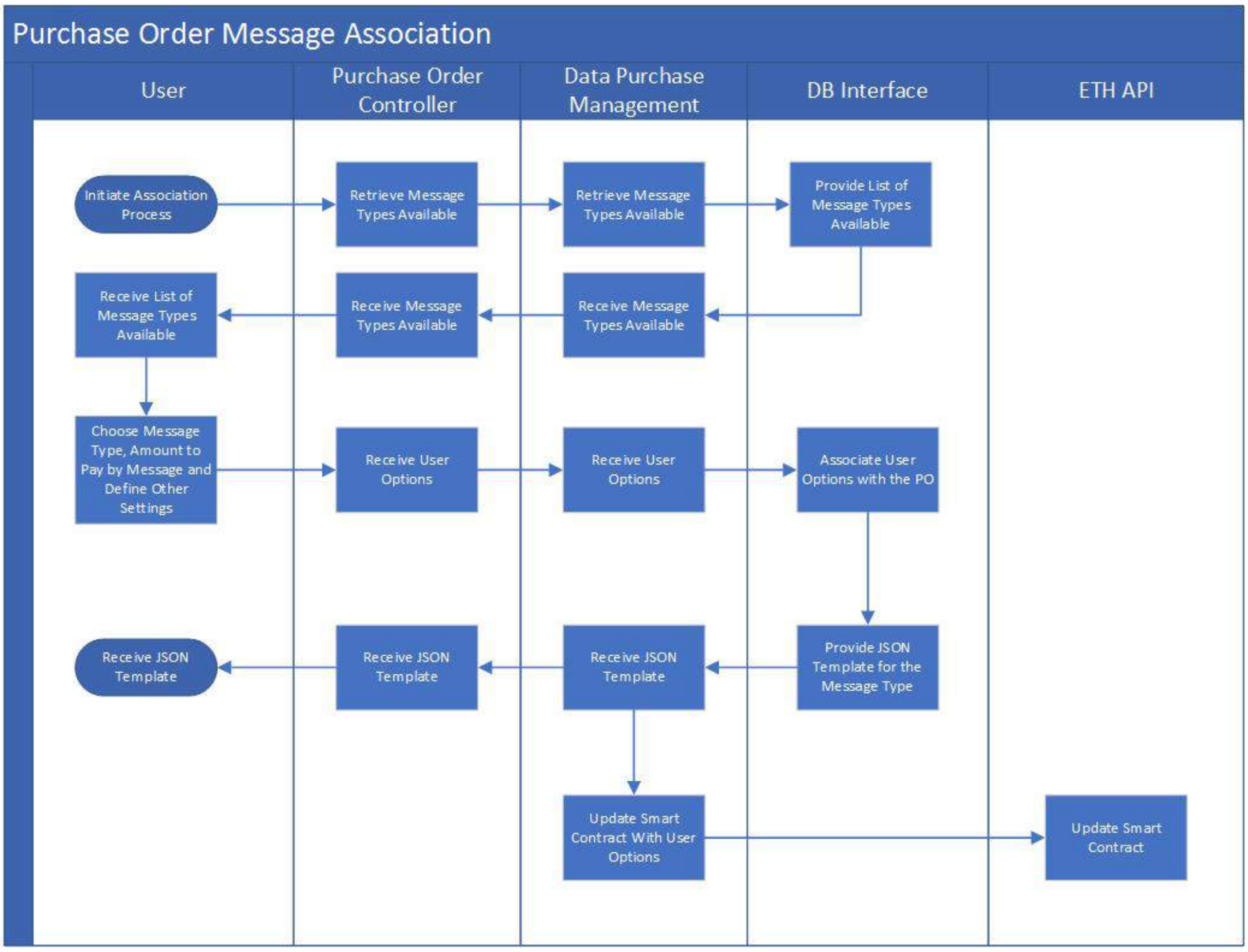

Purchase Order Message Association
User
Purchase Order Controller
Data Purchase Management
DB Interface
ETH API
Initiate Association Process
Retrieve Message Types Available
Retrieve Message Types Available
Provide List of Message Types Available
Receive List of Message Types Available
Receive Message Types Available
Receive Message Types Available
Choose Message Type, Amount to Pay by Message and Define Other Settings
Receive User Options
Receive User Options
Associate User Options with the PO
Receive JSON Template
Receive JSON Template
Receive JSON Template
Provide JSON Template for the Message Type
Update Smart Contract With User Options
Update Smart Contract

Figure 4.9 – Purchase Order Message Association

possible to set the option to receive data only for devices fixed in that location (not from mobile devices). Subsequent is possible to set the minimum time between each message delivered, like to receive one message per minute or one message every hour. Next can define an additional set of limits. It can limit the maximum amount that wants to spend in a specific time window (for example, not to spend more than 0.10 Ether in the one-hour period). Another limit is about unique Seller devices. The Buyer can choose to receive only one message per unique Seller during the time window and set the maximum number of different Sellers that want the messages.

After all these definitions, the Purchase Order Controller receives the data, and it goes to Device Management and the DB Interface. The message setting defined by the User is associated with the PO in the database and, the database provides back the JSON template in which the messages will be delivered to the Buyer. The Data Purchase Management module also updates the Smart Contract with the message settings just defined. The Smart Contract has a variable that stores the type of messages requested by the buyer.

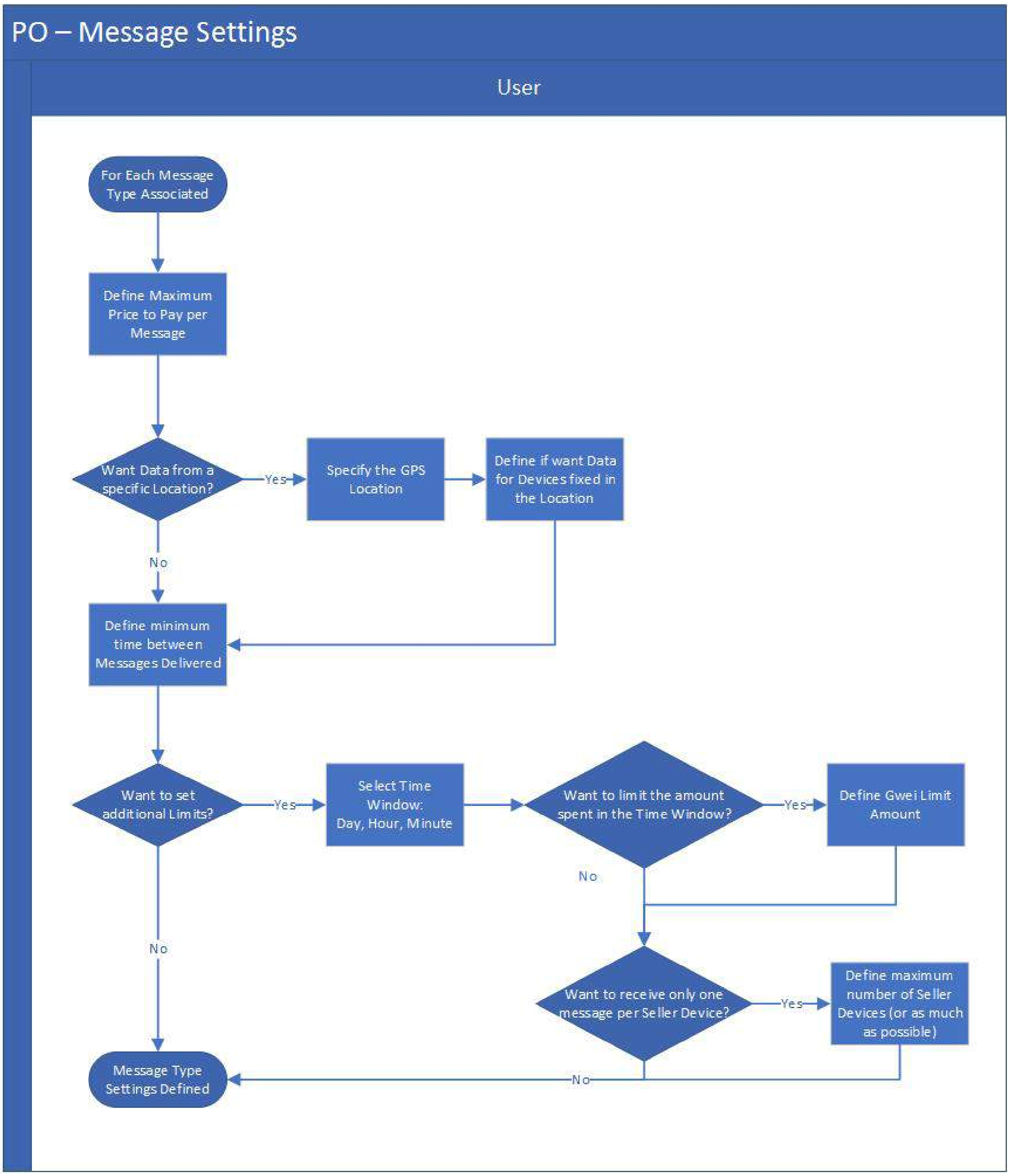

PO – Message Settings
User
For Each Message Type Associated
Define Maximum Price to Pay per Message
Want Data from a specific Location?
Yes
Specify the GPS Location
Define if want Data for Devices fixed in the Location
No
Define minimum time between Messages Delivered
Want to set additional Limits?
Yes
Select Time Window: Day, Hour, Minute
Want to limit the amount spent in the Time Window?
Yes
Define Gwei Limit Amount
No
No
Want to receive only one message per Seller Device?
Yes
Define maximum number of Seller Devices (or as much as possible)
No
Message Type Settings Defined

Figure 4.10 – Purchase Order Message Settings

4.3.2 Update Purchase Order Balance

The platform handles most of the transactions off-chain, which means it happens outside the Blockchain. The interaction with the Blockchain should be as minimal as possible to avoid delays in the process and to reduce the costs associated with the transfers.

For that reason, a background process runs every two hours and get from the database all active Purchase Orders. For each of the PO, that is represented by a Smart Contract in the Blockchain, the platform calls the ETH API and retrieves the balance. The process is illustrated in Figure 4.11. The amount updated in the database is used in the process responsible for delivering the messages (see section 4.5).

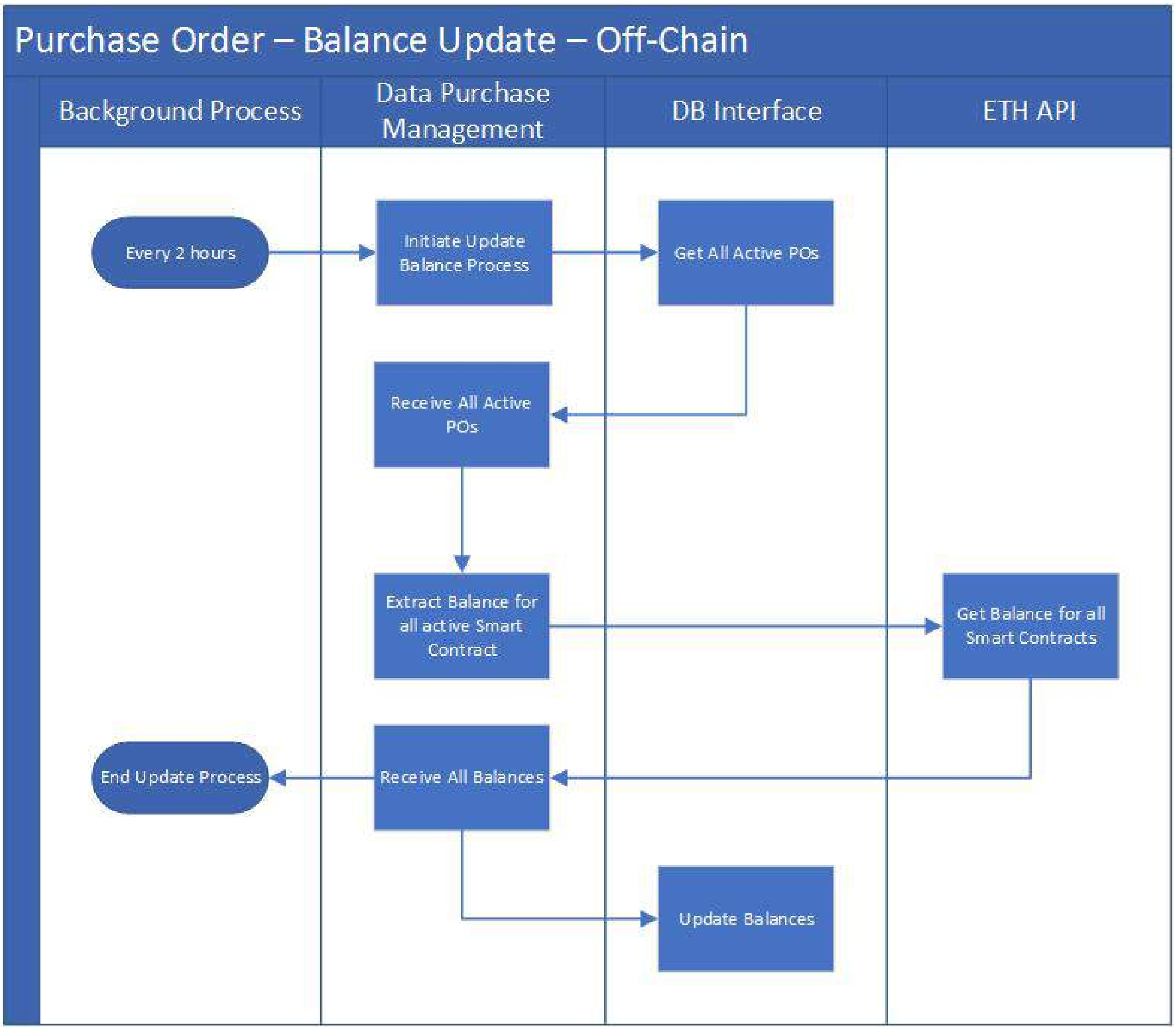


Figure 4.11 – Purchase Order Balance Update

4.4 Mapping Management

The platform assumes that many sellers with different message types and prices are available. Every 6 hours, a background process runs and creates a list of potential sellers for each active PO in the platform. Figure 4.12 shows the steps.

A background process starts the process, and the Mapping Management module retrieves from the database all active Purchase Orders and all Active Seller devices. After that, an interaction process initiates between each message type associated with a PO and each message type associated with the Seller device. The conditions below should match to have the Seller device associated with the PO:

- The message type should be the same;
- The message price defined by the Seller plus the platform commission (20% of the user's price) should be less or equal the maximum amount set by the Buyer;
- The device type (static or mobile) should be in sync with the Buyer request;
- The area the device can be is inside the area of the Buyer interest (if applicable)

If all match, an association is made and stored at the database. The platform executes this process for all active Sellers and POs.

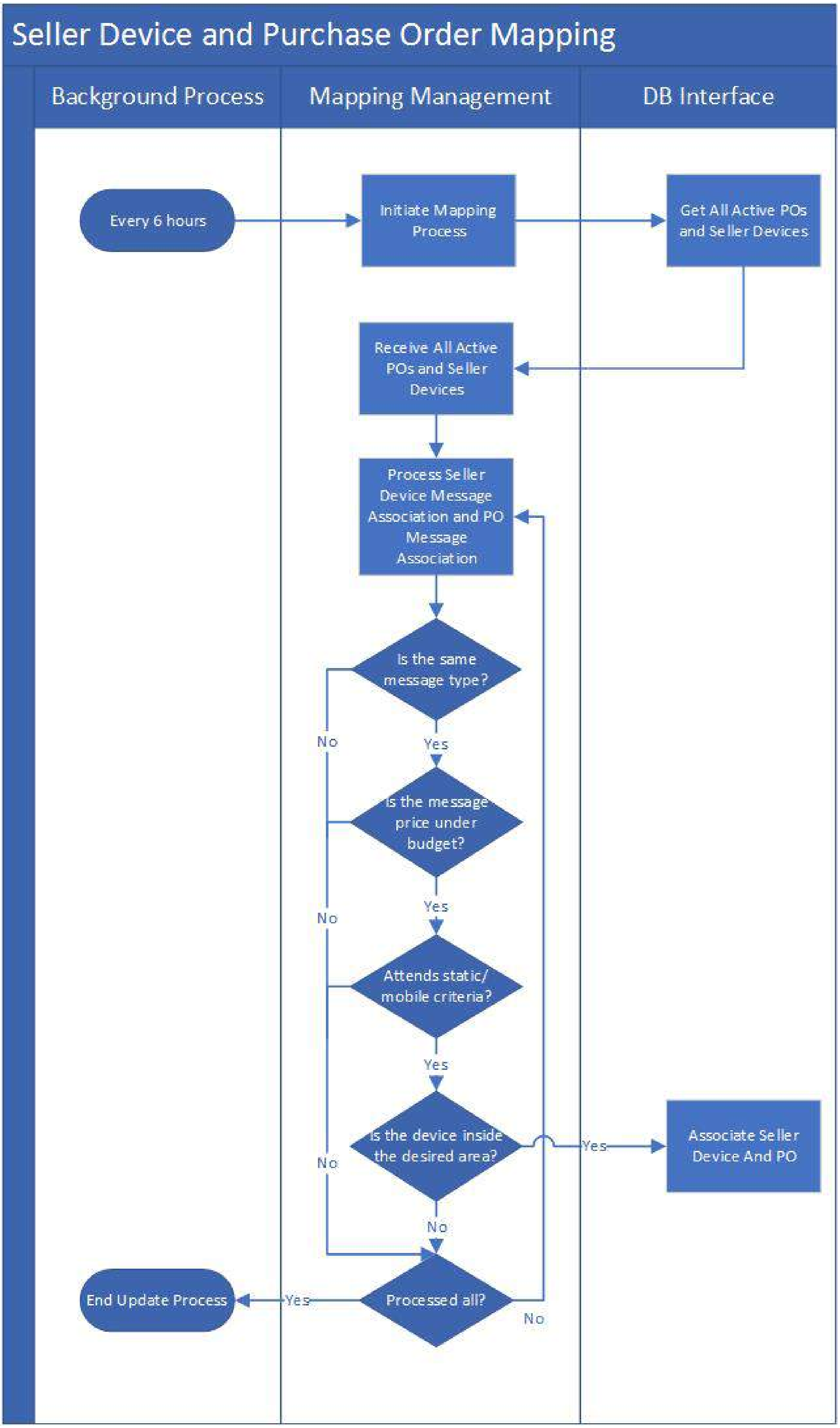

Seller Device and Purchase Order Mapping
Background Process
Mapping Management
DB Interface
Every 6 hours
Initiate Mapping Process
Get All Active POs and Seller Devices
Receive All Active POs and Seller Devices
Process Seller Device Message Association and PO Message Association
Is the same message type?
No
Yes
Is the message price under budget?
No
Yes
Attends static/ mobile criteria?
Yes
Is the device inside the desired area?
Yes
Associate Seller Device And PO
No
No
End Update Process
Yes
Processed all?
No

Figure 4.12 – Seller Device and Purchase Order Mapping

4.5 Process New Messages

Every minute a background process calls the Process New Messages module and initiates the activity. The first step is to update the Purchase Order limits (for the applicable ones). A Buyer can set, for example, the maximum amount he wants to spend in an hour or day. The Update Limits is responsible for zeroing the amounts once the period ends. After that, the module calls the MQTT Plugin and retrieves all messages that are in the MQTT server. The messages received are all saved at the database. An iteration process starts for every single Seller message that arrived. The system compares first if the message sent by the Seller is in the correct format. If yes, it checks if exist any mapping between the Seller and any PO. For the sellers that have mapping, additional actions are executed before delivering the message to the Buyer. The checking are:

- Confirm the Purchase Order is still active. The platforms map the POs with the Sellers every 6 hours, and there is a chance the PO was inactivated during this period;
- Confirm if the Purchase Order has enough balance;
- Confirm if the data is from the desired location. The Seller set the device area but, for some reason, may deliver from another location;
- Confirm all other limits are satisfied. The other limits are:
  - The minimum time between each message;
  - The maximum amount spent during the time window;
  - If the PO limited to receive data only from unique users during the time window and a message from the Seller was sent before;
  - If the PO limited the maximum number of Sellers in the time window and this number was already reached.

For the messages and POs that satisfy all these criteria, the platform updates the limits and create off-chain transactions at the database. The off-chain transactions are the records that indicate the amount of Gwei pending to be transferred from the Smart Contract to the Seller and the Platform. Subsequently, the platform calls the MQTT Plugin and delivers the message to the Buyer topic. The process is executed over and over again for each single Seller Device and PO mapping.

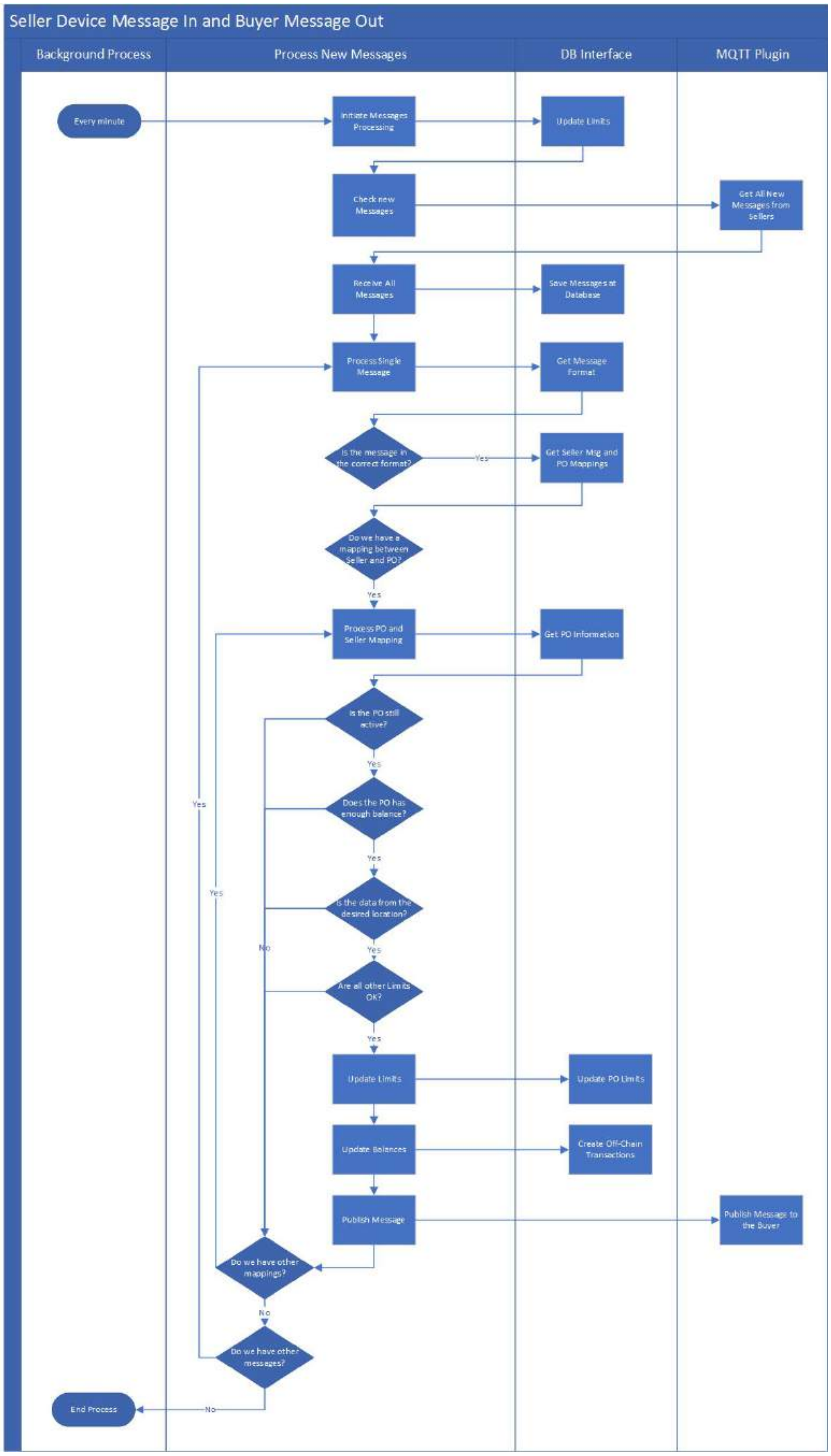

Seller Device Message In and Buyer Message Out
Background Process
Process New Messages
DB Interface
MQTT Plugin
Every minute
Initiate Messages Processing
Update Limits
Check new Messages
Get All New Messages from Sellers
Receive All Messages
Save Messages at Database
Process Single Message
Get Message Format
Is the message in the correct format?
Yes
Get Seller Msg and PO Mappings
Do we have a mapping between Seller and PO?
Yes
Process PO and Seller Mapping
Get PO Information
Is the PO still active?
Yes
Does the PO has enough balance?
Yes
Is the data from the desired location?
Yes
Are all other Limits OK?
Yes
No
Yes
Yes
Update Limits
Update PO Limits
Update Balances
Create Off-Chain Transactions
Publish Message
Publish Message to the Buyer
Do we have other mappings?
No
Do we have other messages?
End Process
No

Figure 4.13 – Process New Message

4.6 Ether Transfer Management

A background process (Figure 4.14) runs every 12 hours and receive from the database all pending transactions (the ones generated in section 4.5), and group by PO and Seller. The platform transfers the Ether from the PO (Smart Contract) to the Seller wallet only if the Seller has more than 0.032 Ethers (around U$10) pending to receive or has pending transactions (messages delivered but not paid yet) that happened more than six months ago. For these scenarios, the Ether Transfer Management module calls the ETH API, and all the pending amounts are transferred from the PO Smart Contract to the Seller and Platform wallet address. The ETH API provides back the transaction ID (a unique identifier in the Blockchain platform), and the transactions are marked as processed.

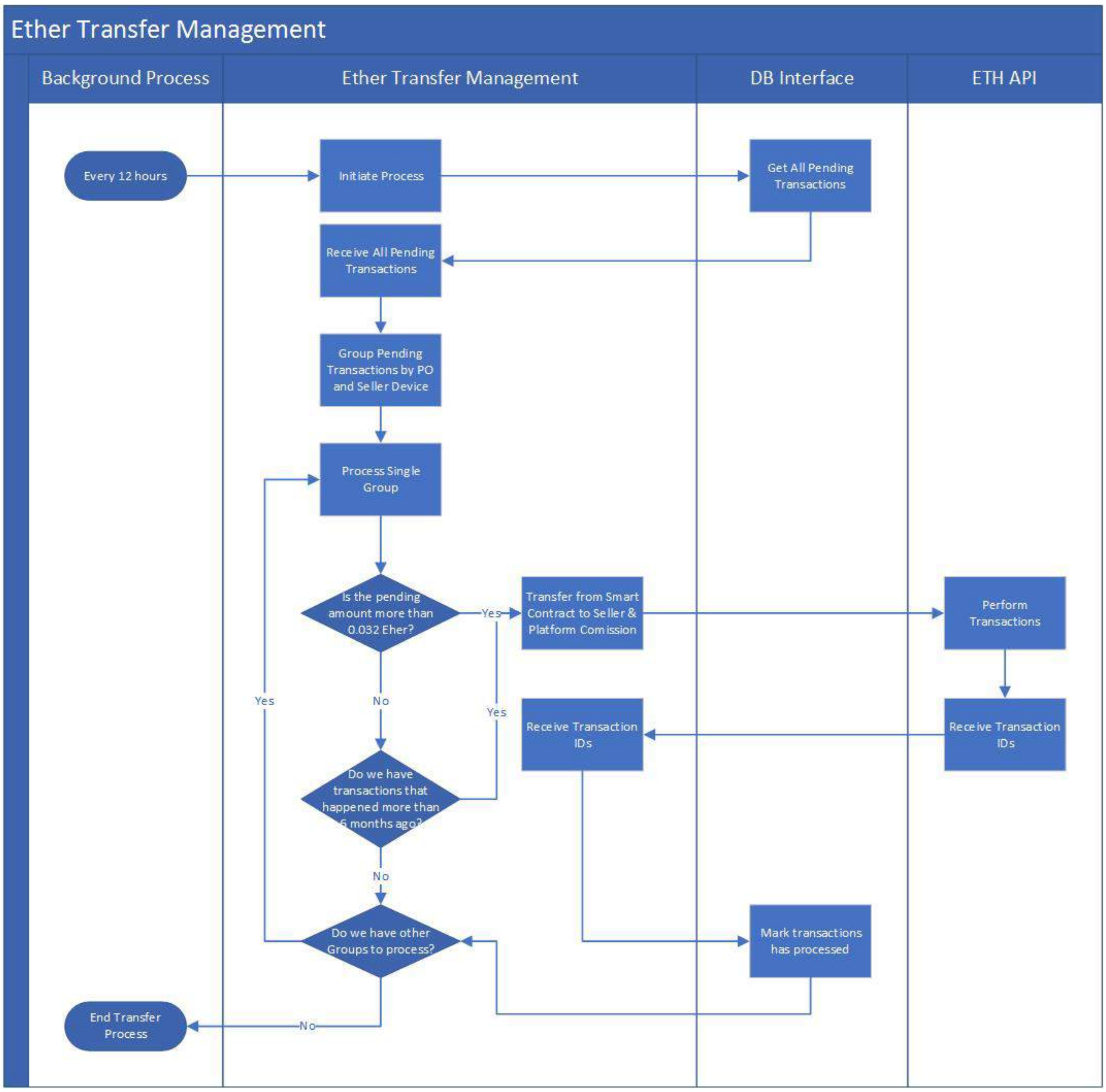

Ether Transfer Management
Background Process
Ether Transfer Management
DB Interface
ETH API
Every 12 hours
Initiate Process
Get All Pending Transactions
Receive All Pending Transactions
Group Pending Transactions by PO and Seller Device
Process Single Group
Is the pending amount more than 0.032 Eher?
Yes
Transfer from Smart Contract to Seller & Platform Comission
Perform Transactions
No
Yes
Yes
Receive Transaction IDs
Receive Transaction IDs
Do we have transactions that happened more than 6 months ago?
No
Do we have other Groups to process?
Mark transactions has processed
End Transfer Process
No

Figure 4.14 – Ether Transfer Management

5. EVALUATION

This chapter presents the evaluation tests performed in the data marketplace platform. The main goal of the tests is to evaluate the average time it takes between the seller sent, and the buyer receives the message (Figure 5.1), and how long the platform takes to create the mapping between seller and buyer.

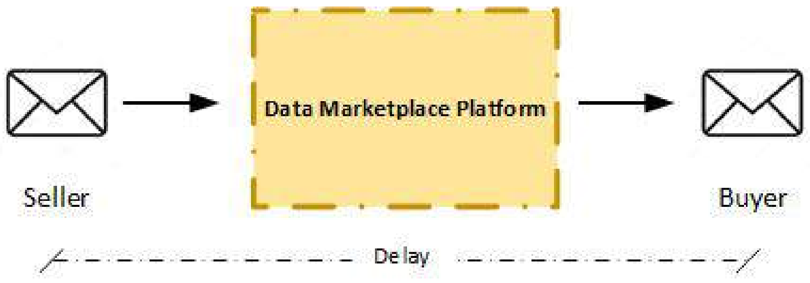


Figure 5.1 – Delay

5.1 TEST ENVIRONMENT

The platform was deployed in the Microsoft Azure cloud. The database (Microsoft SQL Server version 12.0.2000.8) and the Web Site run as Azure Services. Two other virtual machines are in place.

Virtual Machine A

Purpose: hosts the RabbitMQ (version 3.7) and the Background services (writing in C# language) responsible for processing the data in, delivering the data out, transfer the Ether, create the mapping between sellers and vendor.

Configuration: Windows Server 2016, 4CPUs and 16Gb of memory

Virtual Machine B

Purpose: host the Ethereum private blockchain (Go Ethereum version 1.8.27).

Configuration: Windows Server 2016, 2CPUs and 8Gb of memory

5.2 TEST SCENARIO PREPARATION

A total of 11000 different users were created in the data marketplace system using C# code and the platforms APIs. For each user, an Ethereum account was created in the private blockchain. A total of 10000 users were set as Sellers and 1000 as Buyers. For each buyer, a Purchase Order was created, and the respective Smart Contract deployed.

Every seller publishes a People Location data, and all the buyers are interested. The maximum price per message, the location area of interest are set, but no other additional limit is in use.

5.3 TEST METHODOLOGY

A CSV file (Figure 5.2.a) is created with the seller MQTT credentials and message content to be delivered. A python script (Figure 5.2.b) loads the CSV file and publish each message to the platform MQTT inbound topic (Figure 5.2.c). Another python script (Figure 5.2.d) saves the data received in the MQTT inbound into the Microsoft SQL database (Figure 5.2.e). A C# background service (Figure 5.2.f) monitors the messages saved into the Microsoft SQL database and process each one. It performs all necessary validations, and for the messages with a designed buyer, it publishes into the MQTT topic (Figure 5.2.g). The buyer side has a python script (Figure 5.2.h) that receives the message and saves it into another SQL database (Figure 5.2.i). The test represents the end-to-end process, starting in the data generation by the seller until the buyer consumes it.

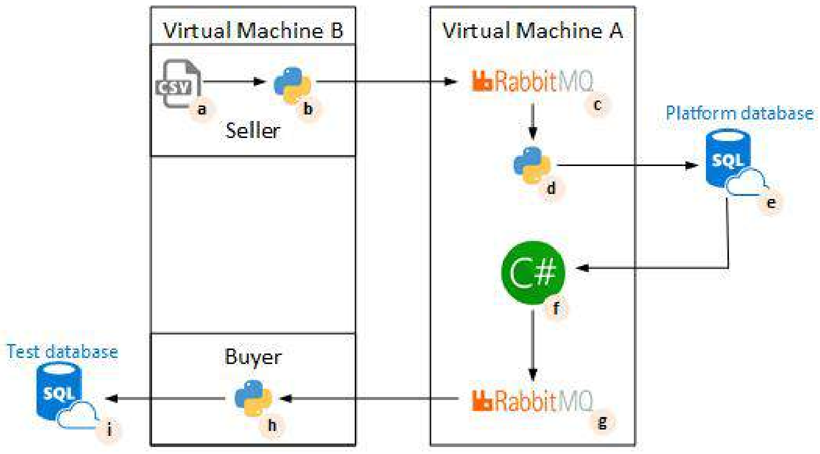


Figure 5.2 – Test Workflow

5.4 SCENARIO 1 - IMPACT OF THE NUMBER OF BUYERS

The data information sent by a seller can be sold to multiple buyers.

The impact of memory, and CPU consumption in the Platform server (Virtual Machine A) and the delay the message takes between the seller generates, and the time the buyer receives it are evaluated. Also, the time the background process responsible for generating the list of potential buyers for a specific seller takes longer when the number of buyers increase. The test started with one buyer and was increased to 10, 100, 200, 300, 400, 500, and 1000. For each variation, the same test was performed five times.

5.4.1 RESULTS

The background process responsible for generating the list of potential buyers for a specific seller takes 1 second in all scenarios.

The CPU and memory consumption for all the affected services remain stable during all the execution (Table 5.1).

Table 5.1 – Scenario 1 - CPU and Memory Utilization

Service	**Memory**	**CPU**
RabbitMQ	100MB	15%
Pythong Script IN	5MB	1%
C# Background Service	25MB	1%

The time it takes between the seller generates the message and the last buyer to receive it increased proportionally (Figure 5.3).

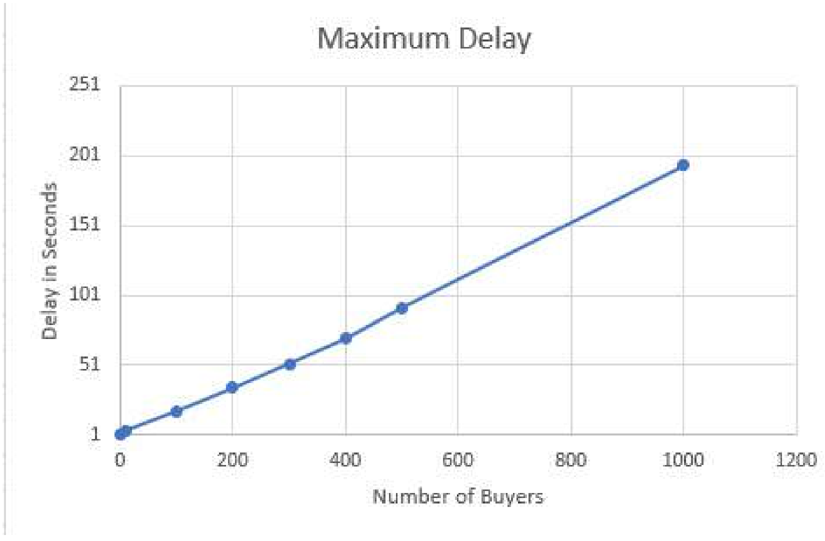


Figure 5.3 – Delay in Scenario 1

Figure 5.3 shows that when we had ten buyers that should receive the message, the 10th buyer received the message at maximum 4 second from the time the seller sent the message. In the worst-case scenario, the 1000th buyer received the message 194 seconds after the time the seller sent the message to the platform. This delay occurs because the process responsible for processing and delivering the message is all sequential. The current code generates the list of potential buyers and performs the additional checks and message delivery sequentially, one buyer after the other. This piece can be parallelized, and each buyer processed individually and concurrently since one process does not impact the other.

5.5 SCENARIO 2 - IMPACT OF THE NUMBER OF SELLERS

A buyer can be interested in receiving the message from multiple devices. In this scenario, during the test, a message from a different seller was sent each 0.15 seconds. There is only one buyer, and the number of sellers were increased, starting at 1, 10, 100, 200, 300, 400, 500 and 1000. For each variation, the same test was performed five times.

5.5.1 RESULTS

The background process responsible for generating the list of potential buyers for a specific seller takes maximum of 6 seconds.

The CPU and memory consumption for all the affected services are the same from section 5.4.1.

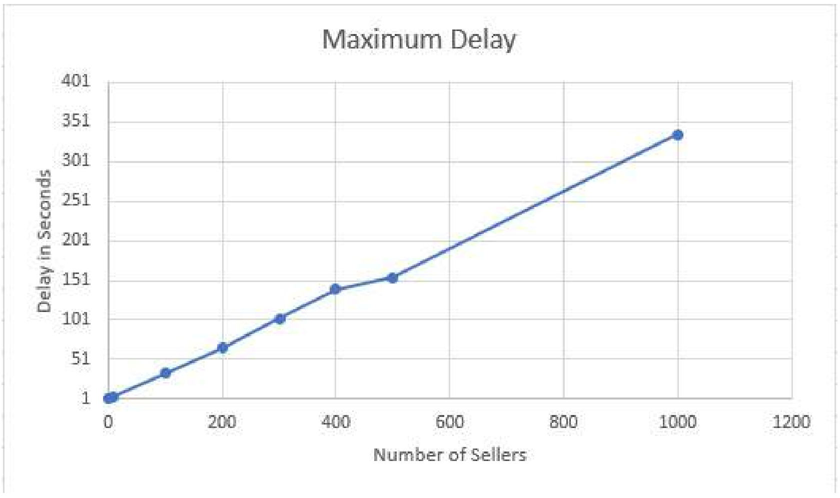


Figure 5.4 – Delay in Scenario 2

Figure 5.4 shows that when we had ten sellers delivering the message with 0.15 seconds interval, the buyer received the message from the 10th seller with a maximum delay of 4 seconds from the time the seller sent the message. When was increased the number to 1000 sellers sending a message with a 0.15 seconds interval, the maximum delay increases to 335 seconds. Table 5.2 shows the minimal, maximum, and average delay in seconds for the tests.

As mentioned in section 5.4.1, the delay occurs because the process responsible for processing and delivering the message is all sequential. We should at least separate

Table 5.2 – Scenario 2 - Message Delay in Seconds

Number of Sellers	**Minimum Delay**	**Average Delay**	**Maximum Delay**
1	1	1	1
10	1	3	4
100	1	19	33
200	1	35	65
300	1	53	102
400	1	72	140
500	1	80	155
1000	1	165	335

the two process and run in parallel — one process generating the messages and the other one consuming and publishing it.

6. CONCLUSION AND FUTURE WORK

In this chapter the conclusions for this work and the future directions are presented.

6.1 CONCLUSIONS

In a world with the estimate of having 20 billion IoT devices in 2020 [22] with a potential data marketplace of $3.6 trillion [1], it is understandable the emerging of research in this area, including a few startups ([20], [12], [55]).

This work has a purpose of showing a data marketplace architecture that contains a data matching mechanism that allows buyers to find the sellers automatically, integrating a payment and contract management mechanism, together with a data broker, keeping an eye on the system performance.

After analyzing and documenting the features of existing projects and papers related to our research, this data marketplace architecture was created.

The proposed architecture contains a critical item that only one paper [26] refers: we need to decide the seller on behalf of the buyers and, this aspect is in the core of our architecture model. Imagine a system with millions of connected devices, all looking to sell his data and other millions of devices wanting to buy these data. How to make the choose an option? A parallel in how Uber works today is possible. When calling an Uber, there is no option to choose the driver. The Uber platform chooses him. In Uber's case, it is only necessary to inform the origin and destination, and it can choose the driver. In a data marketplace, what are the necessary inputs from the buyers to select the best sellers? This work provided a few insights into this space.

A proof of concept covering all characteristics of the marketplace was developed and tested. The proposed architecture was integrated with MQTT for the data broker and with Ethereum blockchain for the contract management and money transfer mechanism, making it possible and end-to-end test in a controlled environment.

As a result, this work has achieved its proposal. A proposal architecture was established and validated. The performance was also validated in the proof of concept.

6.2 FUTURE WORK

The work presented here contains the foundation for a data marketplace focused on IoT devices, but it is not a complete one. It is necessary to explore some areas.

The current architecture assumed that the messages delivered by the sellers are genuine and authentic. It can be true if only certified sellers are allowed to join the marketplace but, doing that, it will limit the potential number of sellers. If in the other way, we allow anyone to be a seller, it is necessary to add mechanisms for the buyers to reject messages from "fake" sellers, devices that sent incorrect or wrong data only for the money.

As mentioned in section 5.4.1, it is necessary to recode the piece responsible for processing and delivering the messages. It is not feasible a considerable delay between the information generation and the time the consumer receives it. In a scenario where we look for some historical data for a particular analysis, this can be acceptable but, in some other scenario, where, for example, one particular device is looking to receive the data from a neighbored device to make some decision, this is not tolerable.

Only a few APIs are currently available for the devices and POs management. It is necessary to create new ones, responsible for editing the current devices/POs and ones responsible for querying the type of data available in the marketplace with the respective prices. The web interface should provide the minimum, maximum, and type of available messages. With the prices information, the user can decide how much in Ether to transfer to the Smart Contract.

In the current architecture, the pricing model is based on the number of messages sold. However, there are others [45]. Buyers may want to pay a fixed amount and receive all messages from a specific time, like one day. In this model, how to reward the sellers? Also, if a specific buyer wants a minimal number of messages during a specified interval, how to enforce SLA with the sellers?

When dealing with people's data, some regulatory compliances should be followed, depending on the source of the data. There are, for example, specific rules for Europe (EU GPDR) [18], Brazil (General Law on the Protection of Personal Data (LGPD)) [10] and so many other regions. In a global data marketplace platform, laws for these different regions should also be taken into consideration for future work.

REFERENCES

[1] Accenture. “Value of data: The dawn of the data marketplace”. Source: https://www.accenture.com/us-en/insights/high-tech/dawn-of-data-marketplace, Jul 2019.

[2] Ali, M. S.; Vecchio, M.; Pincheira, M.; Dolui, K.; Antonelli, F.; Rehmani, M. H. “Applications of blockchains in the internet of things: A comprehensive survey”, *IEEE Communications Surveys & Tutorials*, vol. 21–2, Dec 2018, pp. 1676–1717.

[3] Armstrong, A. A.; Durfee, E. H. “Mixing and memory: Emergent cooperation in an information marketplace”. In: Proceedings of the International Conference on Multi Agent Systems, 1998, pp. 34–41.

[4] Balint, F.-B.; Truong, H.-L. “On supporting contract-aware iot dataspace services”. In: Proceedings of the IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2017, pp. 117–124.

[5] Bates, J. “Thingalytics: Smart Big Data Analytics for the Internet of Things”. Software AG, 2015, 175p.

[6] Benet, J. “Ipfs-content addressed, versioned, p2p file system”, *arXiv preprint*, vol. 1407.3561, Jul 2014, pp. 1–11.

[7] Buterin, V. “Ethereum white paper, 2014”. Source: https://github.com/ethereum/wiki/wiki/White-Paper, Jan 2019.

[8] Cao, T.-D.; Pham, T.-V.; Vu, Q.-H.; Truong, H.-L.; Le, D.-H.; Dustdar, S. “Marsa: A marketplace for realtime human sensing data”, *ACM Transactions on Internet Technology*, vol. 16–3, Aug 2016, pp. 16–21.

[9] Consultants, H. “What are the differences between public, private and permissioned blockchain network?” Source: https://www.hashcashconsultants.com/media/differences-between-public-private-and-permissioned-blockchain-network/, Jan 2019.

[10] da República, P. “Lei geral de proteção de dados pessoais”. Source: http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm, Jul 2019.

[11] Dajie. “Dajie”. Source: https://dajie.eu, Jan 2019.

[12] DAO, D. “Databroker dao”. Source: https://databrokerdao.com, Jan 2019.

[13] Datapace. “Datapace”. Source: https://datapace.io/datapace_whitepaper.pdf, Jan 2019.

[14] Datum. “Datum”. Source: https://datum.org/, Jan 2019.

[15] de la Vega, F.; Soriano, J.; Jimenez, M.; Lizcano, D. “A peer-to-peer architecture for distributed data monetization in fog computing scenarios”, *Wireless Communications and Mobile Computing*, vol. 2018, Sep 2018, pp. 15.

[16] Di Pierro, M. “What is the blockchain?”, *Computing in Science & Engineering*, vol. 19–5, Sep 2017, pp. 92–95.

[17] Economist, T. “The world’s most valuable resource is no longer oil, but data”, *The Economist*, vol. 423, May 2017, pp. 7.

[18] EUR-Lex. “General data protection regulation”. Source: https://eur-lex.europa.eu/eli/reg/2016/679/oj, Jul 2019.

[19] Filament. “Filament”. Source: https://filament.com, Jan 2019.

[20] Foundation, I. “Iota”. Source: https://data.iota.org/, Jan 2019.

[21] Haenmi, R. “White paper v15”. Source: https://datum.org/assets/Datum-WhitePaper.pdf, Jan 2019.

[22] Hung, M. “Leading the iot”. Source: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf, Jan 2019.

[23] IBM. “10 key marketing trends for 2017 and ideas for exceeding customer expectations”. Source: https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-\papers-and-reports-wrl12345usen-20170719.pdf, Oct 2018.

[24] Lastovetska, A. “Blockchain architecture basics: Components, structure, benefits creation”. Source: https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture/, Jan 2019.

[25] Missier, P.; Bajoudah, S.; Capossele, A.; Gaglione, A.; Nati, M. “Mind my value: a decentralized infrastructure for fair and trusted iot data trading”. In: Proceedings of the International Conference on the Internet of Things, 2017, pp. 1–15.

[26] Misura, K.; Zagar, M. “Internet of things cloud mediator platform”. In: Proceedings of the International Convention on Information and Communication Technology, Electronics and Microelectronics, 2014, pp. 1052–1056.

[27] Mišura, K.; Žagar, M. “Data marketplace for internet of things”. In: Proceedings of the International Conference on Smart Systems and Technologies, 2016, pp. 255–260.

[28] Mohanty, S. P.; Choppali, U.; Kougianos, E. “Everything you wanted to know about smart cities: The internet of things is the backbone”, *IEEE Consumer Electronics Magazine*, vol. 5–3, Jul 2016, pp. 60–70.

[29] MQTT. “Mqtt faq”. Source: http://mqtt.org/faq/, Jul 2019.

[30] Noyen, K.; Volland, D.; Wörner, D.; Fleisch, E. “When money learns to fly: Towards sensing as a service applications using bitcoin”, *arXiv preprint*, vol. 1409.5841, 2014, pp. 1–6.

[31] of Things, A. “Array of things”. Source: https://arrayofthings.github.io, Oct 2018.

[32] Oraclize. “Oraclize quick start”. Source: http://docs.oraclize.it/#ethereum-quick-start, Jan 2019.

[33] Özyilmaz, K. R.; Doğan, M.; Yurdakul, A. “Idmob: Iot data marketplace on blockchain”. In: Proceedings of the Crypto Valley Conference on Blockchain Technology, 2018, pp. 11–19.

[34] Panarello, A.; Tapas, N.; Merlino, G.; Longo, F.; Puliafito, A. “Blockchain and iot integration: A systematic survey”, *Sensors*, vol. 18–8, Aug 2018, pp. 2575.

[35] Perera, C.; Liu, C. H.; Jayawardena, S.; Chen, M. “A survey on internet of things from industrial market perspective”, *IEEE Access*, vol. 2, Jan 2015, pp. 1660–1679.

[36] Perera, C.; Wakenshaw, S. Y.; Baarslag, T.; Haddadi, H.; Bandara, A. K.; Mortier, R.; Crabtree, A.; Ng, I. C.; McAuley, D.; Crowcroft, J. “Valorising the iot databox: Creating value for everyone”, *Transactions on Emerging Telecommunications Technologies*, vol. 28–1, Jan 2017, pp. e3125.

[37] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. “Context aware computing for the internet of things: A survey”, *IEEE Communications Surveys & Tutorials*, vol. 16–1, Apr 2014, pp. 414–454.

[38] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. “Sensing as a service model for smart cities supported by internet of things”, *Transactions on Emerging Telecommunications Technologies*, vol. 25–1, Jan 2014, pp. 81–93.

[39] Ramachandran, G. S.; Radhakrishnan, R.; Krishnamachari, B. “Towards a decentralized data marketplace for smart cities”. In: Proceedings of the International Smart Cities Conference, 2018, pp. 1–8.

[40] Reyna, A.; Martín, C.; Chen, J.; Soler, E.; Díaz, M. “On blockchain and its integration with iot. challenges and opportunities”, *Future Generation Computer Systems*, vol. 88, Nov 2018, pp. 173–190.

[41] Sasaki, Y.; Yokotani, T.; Mukai, H. “Comparison with assured transfer of information mechanisms in mqtt”. In: Proceedings of the International Japan-Africa Conference on Electronics, Communications and Computations, 2018, pp. 95–98.

[42] Schomm, F.; Stahl, F.; Vossen, G. “Marketplaces for data: an initial survey”, *ACM Special Interest Group on Management of Data Record*, vol. 42–1, Mar 2013, pp. 15–26.

[43] Seth, S. “Public, private, permissioned blockchains compared”. Source: https://www.investopedia.com/news/public-private-permissioned-blockchains-compared/, Apr 2019.

[44] Slock. “Slock”. Source: https://slock.it, Jan 2019.

[45] Stahl, F.; Schomm, F.; Vossen, G. “The data marketplace survey revisited”, Working papers, ERCIS-European Research Center for Information Systems, 2014, 19p.

[46] Stahl, F.; Schomm, F.; Vossen, G.; Vomfell, L. “A classification framework for data marketplaces”, *Vietnam Journal of Computer Science*, vol. 3–3, Mar 2016, pp. 137–143.

[47] Suliman, A.; Husain, Z.; Abououf, M.; Alblooshi, M.; Salah, K. “Monetization of iot data using smart contracts”, *IET Networks*, vol. 8–1, Jan 2019, pp. 32–37.

[48] Sundmaeker, H.; Guillemin, P.; Friess, P.; Woelfflé, S. “Vision and challenges for realising the internet of things”, *Cluster of European Research Projects on the Internet of Things, European Commision*, vol. 3–3, Mar 2010, pp. 34–36.

[49] Team, T. H. “Quality of service 0,1 2 - mqtt essentials: Part 6”. Source: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/, Jul 2019.

[50] Tennant, L. “Improving the anonymity of the iota cryptocurrency”, Oct 2017.

[51] Terbine. “Terbine”. Source: https://terbine.com/, Jan 2019.

[52] Weeve. “Product highlights”. Source: https://weeve.network/cache/assets/fo0rvac1gv8v/35yrjNec7mK64gaaicyw2e/ac89a1e284e156f006c362a39e100527/Weeve_Product_Highlights.pdf, Jan 2019.

[53] Weeve. “Weeve”. Source: https://weeve.network/, Jan 2019.

[54] Wibson. “Wibson”. Source: https://wibson.org/wp-content/uploads/2018/10/Wibson-Technical-Paper-v1.1.pdf, Jan 2019.

[55] XBR. “Xbr”. Source: https://xbr.network/, Jan 2019.

[56] Yaga, D.; Mell, P.; Roby, N.; Scarfone, K. “Blockchain technology overview”, *arXiv preprint*, vol. 1906.11078, Jun 2019, pp. 1–68.

[57] Zhang, F.; Cecchetti, E.; Croman, K.; Juels, A.; Shi, E. “Town crier: An authenticated data feed for smart contracts”. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 270–282.

[58] Zuiderwijk, A.; Loukis, E.; Alexopoulos, C.; Janssen, M.; Jeffery, K. “Elements for the development of an open data marketplace”. In: Proceedings of the Conference for E-Democracy and Open Government, 2014, pp. 309–322.

APPENDIX A – API

APIs are in place, and users (or machines) can manage their needs using scripts.

A.1 Data Marketplace User Registration

As a first step, a user should register himself in the platform.

API Url: /api/User/CreateUser

Action: POST

Input Format: JSON

Input fields: FirstName, LastName, Email, Password, ConfirmPassword

Output: User Email if the registration works

Input Message Example:

```
{
   "FirstName": "First Name API",
   "LastName": "Last Name API",
   "Email":␣"UserAPI@Useremail.com",
   "Password": "PasswordAPI123",
   "ConfirmPassword": "PasswordAPI123"
}
```

A.1.1 User Token Generation

JSON Web Tokens [1] are in use for API authentication. Before calling the APIs responsible for managing the Seller Devices and Purchase Orders, the user should generate a token and use it to authenticate in the other services.

API Url: /api2/GenerateToken

Action: POST

Input Format: JSON

Input fields: UserID, Password

Response: JSON with access token

Input Message Example:

[1] https://jwt.io/introduction

```
{
    "UserID": "UserAPI@Useremail.com",
    "Password": "PasswordAPI123"
}
```

Response Message Example:

```
{
     "authenticated": true,
     "created": "2019-06-17 17:12:31",
     "expiration": "2019-06-17 17:21:51",
     "accessToken": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1
        bmlxdWVfbmFtZSI6WyJVc2V
     yQVBJQFVzZXJlbWFpbC5jb20iLCJVc2VyQVBJQFVzZXJlbWFpbC5jb20
        iXSwianRpIjoiZDc2Mzc4ZDM
     wN2Y0NDdhMmJiYjk4M2ZmYjkwNTQwZTYiLCJuYmYiOjE1NjA3OTE1NTEsImV4cCI6MTU
        2MDc5MjExMSw
     iaWF0IjoxNTYwNzkxNTUxLCJpc3MiOiIwMTIzNDU2Nzg5
        QUJDREVGIiwiYXVkIjoiMDEyMzQ1Njc4OUF
     CQ0RFRiJ9.YWKkZSS3hKlCtnIxZETNaCt8_Ftu9utjNqWJ_ViJpECaYmap5ClX9V91
        gpR6NGCMM9XUeK
     1jE8sIlIcdDLPulv-jt04pINn2x4SEgDnbk8jgbs21fgaufL9aVReDBCIwXS_7
        AKVJVuqTpKMZWpRgoc
     1IuXEAqm9ANGoSqVPYaBiUNAuGtV7HBlDkOHxyM2OgUrBNeSiICOGEYU9SATVi2o1k77
        dHwnkQbrkRr
     k7r4xATB59b8DV6uAFb8OOZtSsnhf4a4ep3-8i0Qqz9KgNhEDUhSGxi-B2zWOp0jjecQ
        8_sioe-2JaXw
     y-G5r4nBuE75citJV_5gjnM3MYVPhCvfQ",
     "message": "OK"
}
```

A.2 Seller Device Registration

API Url: /api2/Devices/Create

Authentication: Required. JSON Token

Action: POST

Input Format: JSON

Input fields: Name, Description, Enabled, MACAddress, ETHAccount, IsFixed, GPSLocation

Response: JSON message with the device information, including MQTT Queue, user name and password

Input Message Example:

```
{
    "Name": "Temperature Sensor - My Room",
    "Description": "This is my temperature sensor installed in my room",
    "Enabled": 1,
    "MACAddress": "00-0D-3A-1C-F7-A9",
    "IxFixed" : 1,
    "ETHAccount": "0x96eea2090f2b4c30f6b438dd8272ca8c27bc0948",
    "GPSLocation": "(-29.82113,-51.20797)"
}
```

Output Message Example:

```
{
    "DeviceID": 1026,
    "UserID": 1204,
    "Name": "Temperature Sensor - My Room",
    "Description": "This is my temperature sensor installed in my room",
    "Enabled": 1,
    "MACAddress": "00-0D-3A-1C-F7-A9",
    "ETHAccount": "0x96eea2090f2b4c30f6b438dd8272ca8c27bc0948",
    "PriceTypeID": 0,
    "MQTTUser": "prod_1001026",
    "MQTTPwd": "c477fd3-",
    "MQTTQueue": "/inData/prod_1001026/",
    "IsFixed": 1,
    "GPSLocation": "(-29.82113,-51.20797)"
}
```

A.2.1 Seller Device Message Association

API Url: /api2/Devices/AddMessageMap

Authentication: Required. JSON Token

Action: POST

Input Format: JSON

Input fields: MsgID, DeviceID, PriceAmountUser, MsgFrequencyAmount, FrequencyTypeID

Response: JSON message with template to be used when delivering the data

Input Message Example:

```
{
    "MsgID": 1,
    "DeviceID": 1026,
    "PriceAmountUser": 10000,
    "MsgFrequencyAmount": 1,
    "FrequencyTypeID": 4
}
```

Output Message Example:

```
{
    "Location" : "",
    "Type" : "Msg",
    "Date" : "",
    "MsgID" : 1,
    "Value": ""
}
```

A.3 Purchase Order Registration

API Url: /api2/Requests/Create

Authentication: Required. JSON Token

Action: POST

Input Format: JSON

Input fields: Name, Description, Enabled, ETHAccount

Response: JSON message with

Input Message Example:

```
{
    "Name": "Buy Temperature Data",
    "Description": "Request to buy temperature data for multiple
       locations",
    "ETHAccount" : "0x39848570802da1e37f67856c78f9040d38188e9e",
    "Enabled" : 1
}
```

Output Message Example:

```
{
    "RequestID": 109,
    "UserID": 1204,
    "Name": "Buy Temperature Data",
    "Description": "Request to buy temperature data for multiple
       locations",
    "Enabled": 1,
    "ETHAccount": "0x39848570802da1e37f67856c78f9040d38188e9e",
    "MQTTUser": "cons_100109",
    "MQTTPwd": "6ab55a7-",
    "MQTTQueue": "/outData/cons_100109/",
    "CloseUser": null,
    "CloseDate": null,
    "ETHContract": "0x394217adf3a7629c812e1fd8a70818ae2b3a1889",
    "TransactionHashClosed": 0
}
```

A.3.1 Purchase Order Message Association

API Url: /api2/Requests/AddMessageMap

Authentication: Required. JSON Token

Action: POST

Input Format: JSON

Input fields: MsgID, RequestID, AmountUnit, UniqueUsers, OnlyFixedDevices, MinimumTimeBetweenMessages, ExpectedDistinctUsers, FrequencyTypeID, MaximumAmount, GPSLocation

Response: JSON message with template to be used when consuming the data

Input Message Example:

```
{
    "MsgID":1,
    "RequestID":109,
    "AmountUnit": 15000,
    "UniqueUsers": 1,
    "OnlyFixedDevices": 0,
    "MinimumTimeBetweenMessages": 0,
    "ExpectedDistinctUsers": 0,
    "FrequencyTypeID": 6,
    "MaximumAmount": 0,
    "GPSLocation": "(-30.09479,-51.19595), (-30.06608,-51.09909), (-30.01
       955,-51.08505), (-30.00155,-51.17401), (-29.99833,-51.23348), (-3
       0.03733,-51.26)"
}
```

Output Message Example:

```
{
    "Location" : "",
    "Type" : "Msg",
    "Date" : "",
    "MsgID" : 1,
    "Value": ""
}
```

APPENDIX B – MQTT

The current platform uses RabbitMQ 3.7 [1] for the message integration. This tool has APIs for managing users, and messages, available for .Net platform [2].

B.1 Create a new MQTT User

Both Seller Devices and POs need user setup in the MQTT environment for sending and receiving messages.

The system calls the RabbitMQ API (http://localhost:15672/api/users/new_user), replacing the new_user by the desired username, including a JSON message with the new user password and an optional tag. We use the tag to identify if the user is Buyer (Consumer) or a Seller (Producer).

Example of JSON message:

```
{
    "password":"90cd2bd-",
    "tags":"Producer"
}
```

B.2 Assign MQTT Topic Permission

A user can only publish or consume from his topic. For this, after creating the user, the platform adjust some permissions, executing the API (http://localhost:15672/api/topic-permissions/%2F/new_user), passing a JSON message.

For a Seller Device, the JSON message responsible for setting the permission is:

```
{
    "exchange":"amq.topic",
    "write":"new_user.*",
    "read":""
}
```

For a Purchase Order (Buyer), the JSON message is:

[1] https://www.rabbitmq.com
[2] https://www.rabbitmq.com/dotnet-api-guide.html

```
{
    "exchange":"amq.topic",
    "write":"",
    "read":"new_user.*"
}
```

B.3 Consuming and Publishing MQTT Messages

The SAEA [3] and the Eclipse Paho MQTT Python [4] are in use for the MQTT messages interactions.

[3] https://github.com/yswenli/SAEA
[4] https://pypi.org/project/paho-mqtt/

APPENDIX C – MESSAGE TYPES

The platform has cataloged a different set of data types for selling. Buyers and Sellers should follow the message template when delivering or consuming a message. The format is available for the user in the Web Layer (section 4.1).

The Basic Message template is:

```
{
    "Location": "",
    "Type": "msg",
    "Date": "",
    "MsgID": X,
    "Value":""
}
```

Fields Definition:

- Location: The device latitude and longitude position;
- Type: Fixed value of "msg";
- Date: The date the information applies in ISO 8601 [1] format;
- MsgID: Integer number related to the message type;
- Value: Amount measured for the specific data type.

C.1 Application Usage - Facebook

Provides the number of seconds a person used the Facebook application during a specific time window.

The MsgID has a value of 7.

The Location is the coordinates the person is when delivering the information.

Example.: a person arrived PUC-RS Building 32 on June 28th, 2019 at 07 PM BRT, and between June 27th, 2019 01 AM BRT and June 28th, 2019 7 PM BRT, used Facebook for 2 hours.

```
{
    "Location": "(-30.061183,-51.173791)",
    "Type": "msg",
```

[1]https://www.iso.org/iso-8601-date-and-time-format.html

```
    "Date": "2019-06-28T19:00:00-03:00",
    "MsgID": 7,
    "StartDate": "2019-06-27T01:00:00-03:00",
    "EndDate":   "2019-06-28T19:00:00-03:00",
    "Value":"7200"
}
```

C.2 Application Usage - WhatsApp

Provides the number of seconds a person used the WhatsApp application during a specific time window.

The MsgID has a value of 10.

The Location is the coordinates the person is when delivering the information.

Example.: a person arrived PUC-RS Building 32 on June 28th, 2019 at 07 PM BRT, and between June 27th, 2019 01 AM BRT and June 28th, 2019 7 PM BRT, used WhatsApp for 3 hours.

```
{
    "Location": "(-30.061183,-51.173791)",
    "Type": "msg",
    "Date": "2019-06-28T19:00:00-03:00",
    "MsgID": 10,
    "StartDate": "2019-06-27T01:00:00-03:00",
    "EndDate":   "2019-06-28T19:00:00-03:00",
    "Value":"10800"
}
```

C.3 Lumen

It provides the amount in Lumen is emitted by a light source.

The MsgID has a value of 12.

The Value has two decimal precision.

Example.: device located at Arena do Grêmio stadium emits 1600lm (100 W) in June 29th, 2019. The message, in this situation, is:

```
{
    "Location": "(-29.973592,-51.194894)",
    "Type": "msg",
    "Date": "2019-06-29T13:00:00Z",
    "MsgID": 12,
    "Value":"1600.00"
}
```

C.4 People Location

Indicates that a person was in a specific location at that time.

The MsgID has a value of 5.

Example.: a person arrived PUC-RS Building 32 on June 28th, 2019 at 07 PM. The message, in this situation, is:

```
{
    "Location": "(-30.061183,-51.173791)",
    "Type": "msg",
    "Date": "2019-06-28T19:00:00-03:00",
    "MsgID": 5
}
```

C.5 Temperature

For delivering and consuming the temperature in Celcius.

The MsgID has a value of 1.

The Value has two decimal precision.

Example.: device located at Arena do Grêmio stadium measured 25.30ºC in June 29th, 2019. The message, in this situation, is:

```
{
    "Location": "(-29.973592,-51.194894)",
    "Type": "msg",
    "Date": "2019-06-29T13:00:00Z",
    "MsgID": 1,
    "Value":"25.30"
}
```

C.6 Voltage

For delivering and consuming the voltage in Volts.

The MsgID has a value of 3.

The Value has two decimal precision.

Example.: device located at Arena do Grêmio stadium measured 113.45V in June 29th, 2019. The message, in this situation, is:

```
{
    "Location": "(-29.973592,-51.194894)",
    "Type": "msg",
    "Date": "2019-06-29T13:00:00Z",
    "MsgID": 3,
    "Value":"113.45"
}
```

APPENDIX D – ETHEREUM

Ethereum has the main and the test network but, for this validation, a private network is in use to avoid costs associated with transactions (Smart Contract deploy, Ether transfer).

The private network [1] is running in a dedicated Azure Virtual Machine with 2 CPUs and 8Gb of memory. The machine has an Ethereum node operating with Go Ethereum [2] version 1.8.27.

The custom Genis file is:

```json
{
  "nonce": "0x0000000000000055",
  "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "difficulty": "0x0400",
  "gasLimit": "0x8000000",
  "timestamp": "0x0",
  "extraData": "",
  "coinbase": "0x0000000000000000000000000000000000000000",
  "alloc": {},
  "config": {
    "chainId": 100,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "eip160Block": 0,
    "byzantiumBlock": 0
  }
}
```

The Ethereum node runs with the command:

```
geth --identity nodeSOL --nodiscover --networkid 13 --port 60303 --lightkdf --cache 16  --maxpeers 10 --rpc
--rpcapi personal,web3,eth --rpccorsdomain "*" --datadir "C:\ETH\data-private3"
--etherbase "0x141f9017737cd8f6ecf7161c152262db102390d7" --minerthreads 1  --mine
```

[1]https://github.com/ethereum/go-ethereum/wiki/Private-network
[2]https://geth.ethereum.org/

D.1 Data Marketplace Platform and Ethereum Integration

The DMP modules are in C# language, and the integration between the modules and the Ethereum is through Nethereum library [3].

D.1.1 Smart Contract

The Ethereum Smart Contract is written in Solidity language. The current code is:

```
pragma solidity ^0.5.0;
contract DataRequest {

address owner = msg.sender;
address buyerETHAddress;
string strDataTypeRequested;
bool contractStatus;

function deposit() payable public {
}

function setContractInfo(address _buyerETH, string memory  _strDataType)
    public onlyOwner {
    buyerETHAddress = _buyerETH;
    strDataTypeRequested = _strDataType;
}

function transfer(address payable to, uint256 amount) public onlyOwner {
         to.transfer(amount);
}

function getBalance() public view returns (uint256) {
        return address(this).balance;
}

function getStatus() public view returns(bool){
    return contractStatus;
}

function setStatus(bool _newStatus) public OwnerORBuyer
{
        contractStatus = _newStatus;
}
```

[3]https://nethereum.com/

```

    modifier onlyOwner {
        require(
            msg.sender == owner
        );
        _;
    }

     modifier OwnerORBuyer {
        require(
            msg.sender == owner || msg.sender == buyerETHAddress
        );
        _;
    }
}
```

There is an extension [4] for Visual Studio code that generates C# class (with Nethereum support) from a Solidity code.

In our case, to deploy the Smart Contract, we need to use:

```
var account = new Account(privateKey);      //DMP primary account
var web3 = new Web3(account, url);         //Ethereum Private Node

DataRequestDeployment dep = new DataRequestDeployment();
DataRequestService.DeployContractAndWaitForReceiptAsync(web3, dep);
```

Also, to transfer the amount from the contract to a specific address (like a seller), the main code responsible is:

```
DataRequestService dServices = new DataRequestService(web3, strContract); //
    Contract Address
dServices.TransferRequestAndWaitForReceiptAsync(strReceiverAccount, strAmount *
    1000000000); //Receiver Address and convert Gwei to Wei
```

[4] https://github.com/juanfranblanco/vscode-solidity

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br