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.

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/

Dedico este trabalho a meus pais e minha esposa Pauline.

Why use Electrum? Here are a dozen of the benefits of electrum wallets.
If you want to import a private key into a hardware wallet such as a ledger, you need to perform additional steps. First, you need to import the private key into a wallet such as Electrum before you can send coins to your ledger address.
Send transnsaction? (Number)? (address)??Send FM to an address.
Doing so protects your privacy because other Electrum nodes won't be able to see your address and balance. It also lets you verify on the mobile side that the in-account transaction is confirmed.
A new repo qtum-electrum-new was built to add qtum-related features to the latest code for Bitcoin electrum. The Electrum team has also been developing other features. Electrum Wallet users can view the full release notes here.
In data-supporting eth wallets, use the locked eth address to send a transfer transaction (any amount of money transferred) to any address, including your own address, and in the Data field with the transcoding corresponding to the ChainX account address (data generated in step 3), send the transaction to complete the experience.
Send transaction information from address A (the address of the wallet and server) to address B, which is any address of the attacker.
It's not hard to run your own Electrum server and point your wallet to just use it. This restores Electrum to the point where it has the same privacy and security attributes as the full node, where no one else can see the address or transaction that the wallet is interested in. Electrum then becomes an all-node wallet.
To use mnomets to transfer addresses from electrum wallets to web wallets, you need to set Electrum to be compatible with Qtum phone mnomets in the initial installation (and then use phone wallet mnomets to restore phone wallets on Electrum). This setting is screenshot of the Electrum configuration.
Next, Zhang San entered 10 in the wallet software, and entered Li Four's collection address (the bank account number of the collection), click to send, the wallet knew Zhang San to send 10 bitcoins. Then the wallet found the address A, address B, address C belonging to Zhang San, from address A to take 5 coins, from address B to take 3 coins, from address C to take 2 coins. But Zhang San only want to send 10 coins, address A in the last 3 coins, still stored in address A?
Legacy: Although many exchanges and wallets use CashAddr, some haven't yet adopted it (such as fire coins). If you encounter situations where the CashAddr address format is not supported, you

can choose the Legacy address format.
Send "RECEIVE" and you will receive a top-up address for the BCH. The address here is in the new format of BCH, and the preceding "bitcoincash:" is also part of the address. Unfortunately, wallets don't support this new format yet, so keep entering "LEGACY" here and you'll receive an old BCH wallet address.
Send <amount> <address> a specified number of VBKs to the specified wallet address. For example send 10 VH1gg LJk4pjA7mJbidaGxQt5K8Sinw.
Proto Recv-Q Send-Q Local Address Foreign Address State.
Bitcoin wallet Electrum official Twitter announced that the next version of Electrum will support Lightning online payments. Its lightning node implementation has been consolidated into the main branch of Electrum. Electrum also confirmed that the wallet will adopt a new implementation of in-house development written using Python.
In this demo, Electrum developer Chris Belcher shows how to set up and use an Electrum personal server.
The data shows that 40 bitcoins have been transferred to a legacy address, but are still unavailable, and 10 BTCs appear to have been transferred to different addresses.
Electrum is a popular software wallet that works by connecting to a dedicated server. These servers receive a hash of the Bitcoin address in the wallet and reply with transaction information. Electrum Wallet is fast and has few resources, but by default, it connects to these servers and can easily monitor users. In addition to Electrum, some other software uses public Electrum servers. By 2019, it is a faster and better alternative to BIP37.
State Recv-Q Send-Q Local Address: Port Peer Address: Port.
The Healthy Security Lab is concerned that Nearly 250 bitcoins have been stolen in a recent hacking attack on an Electrum wallet. This attack, confirmed by Electrum, involves creating a fake version of the wallet to trick users into providing password information. Electrum responded on Twitter that "this is an ongoing phishing attack on Electrum users and advised users to download wallet apps from the official website" and that The Healthy Security Lab advised users not to install an unknown source of Electrum wallets to avoid being tricked.
Like legacy addresses, Segwit addresses exist. For legacy addresses, P2PKH and P2SH represent "pay to public key hash" and "pay to script hash value" respectively. For Segwit addresses, P2WPKH and P2WSK represent "Paid Witness Pubkey Hash" and "Paid WitnessScript Hash" respectively. The method of making each address will be discussed in another article.

we power

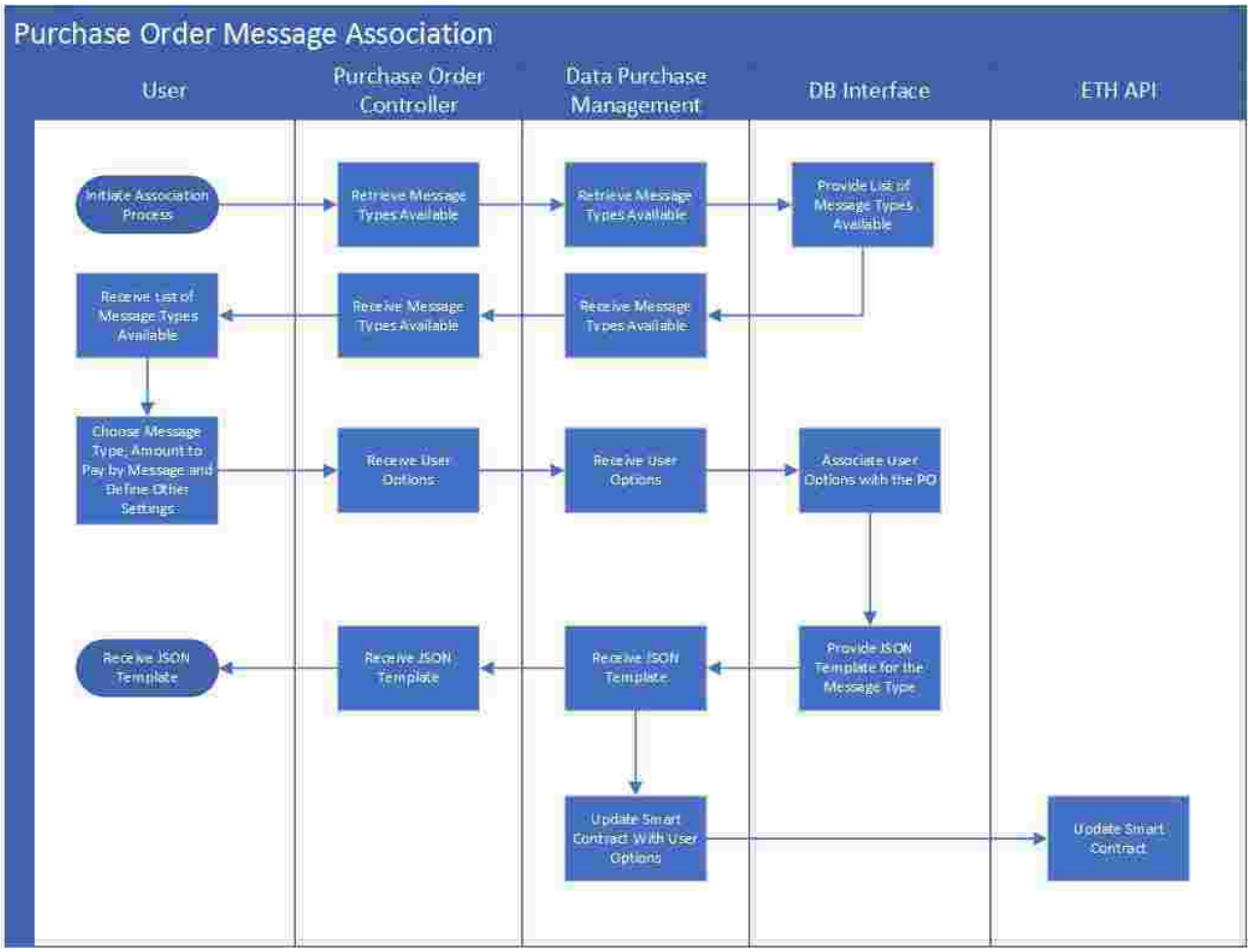

Figure 4.9 – Purchase Order Message Association

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.

CONTENTS

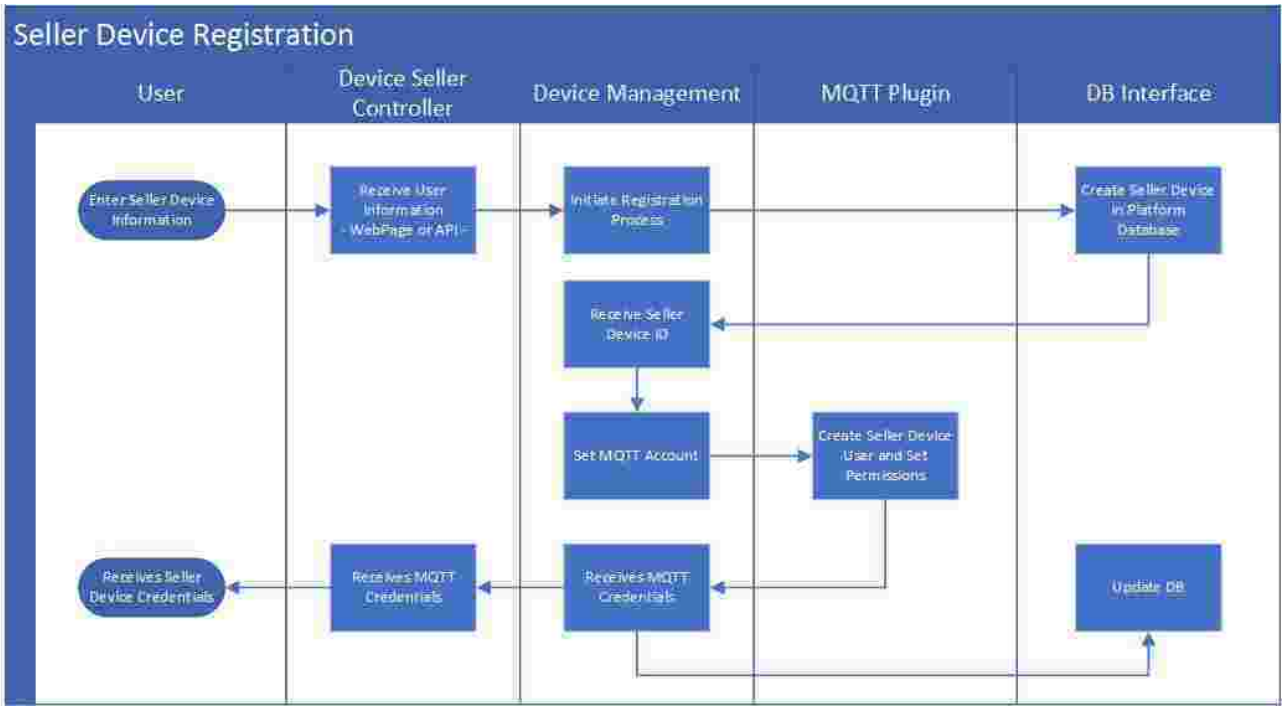

Figure 4.6 – Seller Device Registration

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.