AN EXAMINATION OF ETHEREUM PLATFORM IN THE IMPLEMENTATION OF UNIVERSAL BUSINESS LANGUAGE USE CASES

by

Ramkumar Velmurugan

Submitted in partial fulfilment of the
requirements for the degree of
Master of Applied Computer Science

at

Dalhousie University
Halifax, Nova Scotia
July 2016

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a project work entitled “An Examination of Ethereum platform in the implementation of Universal Business Language use cases” by Ramkumar Velmurugan in partial fulfilment of the requirements for the degree of Master of Applied Computer Science.

Dated: 9-Aug-2016

Supervisor: _____

Dr. Peter Bodorik

Reader: _____

Dr. Meng He

DALHOUSIE UNIVERSITY

DATE: _____

AUTHOR: Ramkumar Velmurugan

TITLE: An Examination of Ethereum platform in the implementation of Universal Business Language use cases

DEPARTMENT: Faculty of Computer Science

DEGREE: Master of Applied Computer Science

CONVOCATION: October 2016

_____

Signature of Author

Table of Contents

ABSTRACT

Ethereum is a platform that is based on the blockchain technology. The Ethereum system allows running specific scripts called smart contracts that can be programmed to perform the computational task to be executed in the blockchain. The running of specific scripts provides a framework that can be used to develop decentralized applications. In this work, we examine the suitability of using the Ethereum platform for implementation of the Universal Business Language use cases developed by OASIS, which is a nonprofit consortium that drives the development, convergence and adoption of open standards for the global information society. We selected a small collection of use cases, for implementation with Ethereum, while ensuring that our selection includes one of the most sophisticated/complex use cases. Along with the implemented use cases, we examine the performance of the Ethereum by analyzing the parameters that affect it. We also examine issues related to the scalability of the Ethereum platform and analyze the architecture. We review the solutions that are being proposed for overcoming the issues related to the scalability and performance of the Ethereum platform.

Acknowledgements

My Supervisor, Dr. Peter Bodorik, deserves the most praise for his invaluable guidance, friendship and suggestions. I would also like to thank Dr. Dawn Jutla for her helpful suggestions and support. Lastly, I thank all of the friends and acquaintances I have made at Dalhousie University. They have made this journey something to remember. This journey has taught me humility.

Chapter 1 - Introduction

1.1 Motivation and purpose

Ethereum is a platform that runs smart contracts that are built on top of the blockchain technology. The application that is built with the Ethereum runs in such a way that it does not have any possible downtime, fraud, censorship or third party interference ("Ethereum Project," n.d.). Ethereum runs on a custom built blockchain which is a global infrastructure and is powerful when compared to its predecessor, Bitcoin. Ethereum infrastructure enables the developers to create a custom decentralized application that can be built on top of the Ethereum platform ("Ethereum Project," n.d.).

Presently, on a traditional client-server architecture, application code is written and deployed on the server and a special client application is written to interact with the server. In the client-server model if a server goes down many users are affected. In Ethereum, anyone can set up a node and access the application that has been deployed in it, thus making the application that is deployed in the Ethereum completely decentralized ("Ethereum Project," n.d.).

Like Bitcoin, Ethereum cannot be controlled by anyone in the world. The project is completely open source and there is no sole owner for Ethereum. In a narrow sense, Ethereum provides a platform to develop and deploy decentralized applications. The central concept of the Ethereum is to provide a blockchain with a built in Turing-complete programming language allowing users to write smart contracts into the block chain ("Ethereum Project," n.d.). Turing complete programming language is defined as the programming language that has the capacity to simulate a single taped turing machine.

Ethereum as a platform does not provide any special features. It is up to the developers and the programmers to develop the smart contracts and applications that use the infrastructure of the Ethereum.

Ethereum is suited for applications that automate direct interaction between the peers or coordinate a group of actions in a group. In theory, Ethereum provides a simple set of instructions to be run automatically when a set of conditions are met. Apart from building decentralized application and deploying smart contracts, Ethereum provides an environment where trust, security, and performance are very important (Mougayar, W. 2015).

Ethereum currently has three crowdfunded projects that are successful in the market. *The DAO (Decentralized Autonomous Organization)* is a project that creates autonomous organization using the Ethereum platform. The project is currently funded with a hundred million dollars. *DAO* provides a new decentralized model for organizing both commercial and non-profit enterprises ("DAO (organization) - Wikipedia, the free encyclopedia," n.d.). *DigixDAO Crowdsale* provides a separate set of tokens called DGD, which is equivalent to the gold standards token of the Ethereum platform. The DigixDAO has raised more than five million dollars from the investors ("List of highest funded crowdfunding projects - Wikipedia, the free encyclopedia," n.d.). *Augur* is a decentralized market prediction software built using Ethereum. The software has collective intelligence to make predictions. Thus, using the block chain functionality generates better forecasts about the future events ("Decentralized Prediction Markets | Augur Project," n.d.).

The rest of this introductory chapter is dedicated to presenting the research question and a project outline.

1.2 Research Objective

In the world of business, several standards are set up for business documents. One such standard is the Universal Business Language (UBL), which are the standard business XML document supporting the digitization of the commercial and logistical processes for domestic and international supply chains, such as procurement, purchasing, transport and other supply chain management functions ("Ethereum Project," n.d.). In this project, we will convert real world use cases of the UBL to the Ethereum platform to test the suitability of the Ethereum for representation and implementation of UBL.

In the implementation of the business use cases, we are analyzing the Tendering, Catalogue, Quotation, Ordering, and Billing. Based on the use cases, specific smart contracts are written and deployed on the Ethereum platform. The Ethereum platform provides a simple set of instructions that can be carried out when a condition is met. Based on the developed smart contract, a decentralized application is developed with the help of standard web scripting languages.

Our project has two objectives:

1. Examine the suitability of deployment of business logic expressed in UBL in Ethereum. For the purpose we selected a small subset of the use case from reference, ensuring that our selected set has one of the most sophisticated and complex cases.
2. Explore the performance and scalability of the Ethereum platform along with the parameters that affect it.

1.3 **Outline**

This report is organized as follows. Chapter 2 provides background information about the decentralized application, the concept of mining, smart contracts, and the UBL. Chapter 3 explains the implementation of the selected UBL use cases, design patterns used in designing the smart contracts and the development framework used in the project. Chapter 4 examines the Ethereum performance and scalability. Chapter 5 describes the conclusion and the future work of Ethereum.

It should be noted that the smart contracts developed for this project were deployed and tested in the private test network. The application did not make any connection with the live block chain. In the project, personal or identifying data were not collected and thus no ethical issues arose during the work in the project.

Chapter 2 – Background

In this chapter, we will review background information related to Ethereum and t0 tegh Universal Business Language. Applications and services are developed day-to-day using web technologies in the normal client-server model to satisfy the business requirements. But building an application in a completely decentralized fashion is a very new approach for normal application to improve massively.

(Buterin, 2016) claimed that Ethereum provides an enormously powerful shared global infrastructure that can move value around and represent the ownership of the property and thus enable developers to create markets and store registries of debts or promises without a middle man or counterparty risk.

In the following, we provide relevant background information of Ethereum's key components that relate to our project.

2.1 Ethereum – High Level Description

Ethereum is a platform that can be used to develop and host decentralized applications. The platform can be easily setup just by installing the *geth* client in the machine. Once the *geth* client is setup the host machine will act as a node in the Ethereum network. The main task of the Ethereum node is to perform the mining operation and validate transactions that are happening in the network.

Ethereum leverages the blockchain technology to deploy and develop the decentralized application. "*A blockchain is a distributed computing architecture where every network node executes and records the same transactions, which are grouped into blocks"* ("Ethereum homestead documentation — Ethereum homestead 0.1 documentation," 2016).

Ethereum allows the developers to write specific scripts called the smart contract that live on the blockchain. Ethereum also follows an incentive-driven model, all the computations done in the blockchain needs to be paid by a cryptocurrency called Ether. Ethereum blockchain will record all the transactions that are taken place in the network. The Ethereum blockchain will store the recent state information and the transaction list in the blockchain. A transaction in Ethereum needs to be approved by all the nodes before getting added into the blockchain.

"Only one block can be added at a time, and every block contains a mathematical proof that verifies that it follows in sequence from the previous block. In this way, the blockchain's "distributed database" is kept in consensus across the whole network. Individual user interactions with the ledger (transactions) are secured by strong cryptography. Nodes that maintain and verify the network are incentivized by mathematically enforced economic incentives coded into the protocol." ("Ethereum homestead documentation — Ethereum homestead 0.1 documentation," 2016).

In a nutshell, Ethereum is a programmable blockchain that lets anyone to build and use decentralized application. Though the computation performed by the Ethereum is complex when compared to the normal computer the platform provides extreme levels of fault tolerance, ensures zero downtime and makes the data stored in the blockchain unchangeable and censorship-resistant.

The following subsections are designed in such a way that it will give a detailed background information about the concepts about the Ethereum.

2.2 Decentralized Application

A decentralized application looks very similar to that of the normal software application. The decentralized application provides a service that creates direct interaction between the end

users and the providers. An application is categorized as a decentralized application if it satisfies the following conditions.

1. *"The Application must be completely autonomous, open source and no third party should control the application.*
2. *The data and the records of the operation performed by the application should be stored in a public blockchain.*
3. *The application must implement a cryptographic token system.*
4. *The token generated by the system should follow a standard cryptographic algorithm, as a proof of concept (Johnston et al)".*

A decentralized application can be further classified into three broad categories. If a decentralized application has a dedicated blockchain, then it is categorized as the type 1. If the decentralized application has a protocol and has a token system that is necessary for their function, then it is categorized to be type 2. The type 3 decentralized application uses the type 2 protocol and uses a separate token system to handle all the functions. Based on the above types, Ethereum platform can be used in all the three types of decentralized application (Johnston et al.). The legal model of developing a decentralized application is to be under open source license so that it is open for innovation without the restriction of copyrights or patents.

2.3 Ether

Ethereum uses a separate cryptocurrency called Ether. The main use of Ether is to pay for the computation done in the blockchain (What is Ether. n.d..). Ether can be obtained by multiple ways. One of the best approaches for generating Ether is by mining. Once the Ethereum node is set up the node starts generating ether by automatic mining. The Ether can also be transferred from one account to another account or traded for real world goods. The sending and receiving of Ether

can be done easily with the help of the Ethereum wallet. Ethereum wallet provides a simple GUI that can be used to monitor the Ether in the account (Get an Ethereum Wallet. n.d.). Some Ethereum wallets implementations are Mist Ethereum, Ktyptokit Jaxx, Etherwall, MyEtherWaller and Cold Storage. *geth*, which is Ethereum node software, has a console that provides the *sendtransaction* command used to transfer ether between accounts.

Apart from the Ether to perform the computation, Ethereum has a utility value called *cryptofuel* (gas). In each transaction, the sender needs to pay a small transaction fee, which is the gas, to perform the computation. The cost of gas is determined dynamically and will depend on the volume and the complexity of the computation to be performed. The gas price is determined by Ethereum based on the network resources and utilization.

2.4 Mining Algorithm

Ethash is the planned proof of work algorithm for the Ethereum. The main purpose of the mining algorithm Ethash is to generate Ether and validate the transaction. The high-level description of the algorithm is as follows

1. *"A seed value is calculated for each block by scanning the header information of the block.*
2. *Based on the seed value a 16 MB pseudorandom number cache is generated.*
3. *Based on the cache value a 1 GB dataset is generated which is hashed and grows linearly with the time.*
4. *The Mining operation involves grabbing the random slices from the dataset and hashing them together.* (Ethereum/wiki. n.d.)*"*

The main activity of the miner is to read the dataset and hash it. Based on the successful hash operation performed by the miner the Ethereum platform will award Ether to the miner. The dataset is automatically updated for every 30000 blocks. The average time needed for the algorithm

to generate a new block will depend directly on the difficulty threshold. Hence, it is possible to control the time for finding the new block by manipulating the difficult threshold level. The Ethereum dictates the difficulty level of the algorithm. The difficulty level automatically gets adjusted that one block is generated every 15 seconds. Any node that is participating in the network will do the process of mining automatically. Each node has an attribute called hash rate, which will decide the speed of the mining (Ethereum/wiki. n.d.). The speed of mining operation is defined as the number of the hash operation performed by the node.

Ethash uses a directed acyclic graph for the proof of work algorithm. The directed acyclic graph (DAG) is generated for every 30000 blocks. The DAG contain the information about the block heights and the other node information in it. Every node needs to generate the DAG before starting the mining operation. Figure 1 shows the DAG generated by the *geth* client before the start of the mining operation. The DAG is designed in such a way as to provide a fast verifiability within the slow CPU environment. The CPU with higher processing power will have higher benefit in mining when compared to the normal CPU. The communication between the external mining application and the Ethereum platform will happen through a JSON-RPC API. The mining operation can be carried out in private node or in a public blockchain (Ethereum/wiki. n.d.).

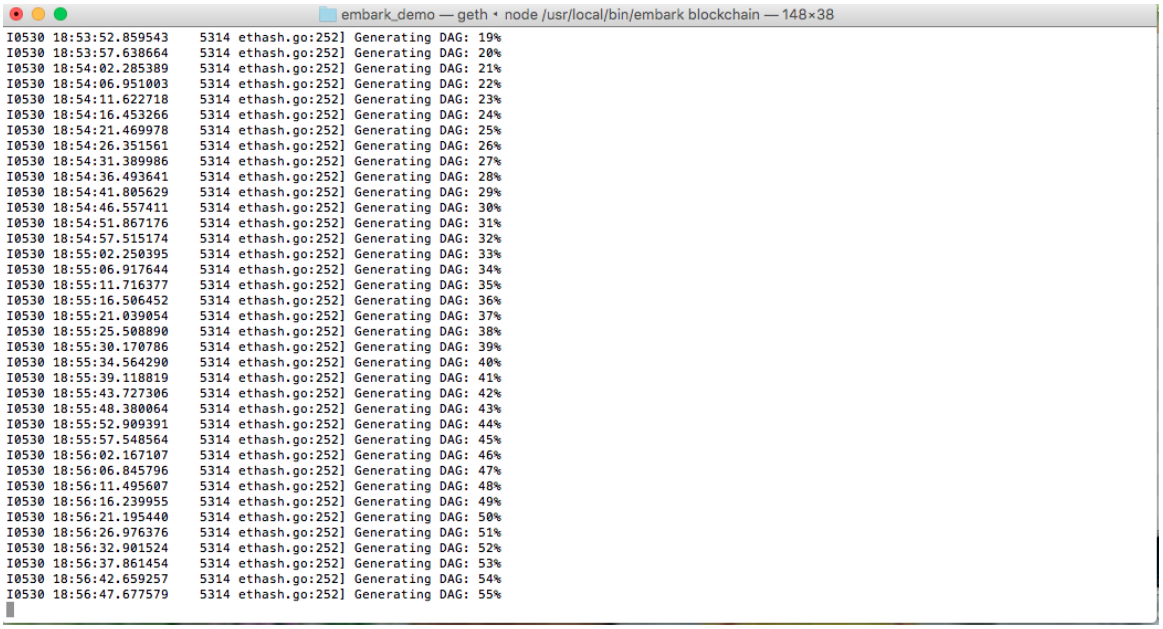

Figure 1 Generation of DAG before the start of the mining operation

2.5 Smart Contracts

The Contract is the collection of code and the data that resides at the specific address on the Ethereum blockchain. A contract can pass messages between two different contracts and can perform turing complete computation. The contract that lives on the block chain will have Ethereum specific byte code called as Ethereum Virtual machine (EVM) byte code.

The contract can be written with the help of high-level languages such as *solidity, serpent, LLL,* and *mutan*. In this project, we develop the smart contract with the help of the solidity. The main reason for selection of solidity is that it provides an online compiler and the development methodologies are well documented. The solidity provides features and syntax similar to that of the modern high-level programming languages like Java and C++. The solidity provides datatypes such *state variables, function, function modifiers, events, struct,* and *enum*. The solidity provides the features of inheritance which enable contracts to inherit functions from one contract to another. Table 1 overviews the features of solidity.

Table 1 Basic set of data types provided by solidity

Type	Description
State variable	A simple variable that is used to store permanent information inside the contract
Function	An executable unit of code written inside the smart contract that can be called from outside of the program
Function modifier	A function modifiers can be used to amend the semantics of the function.
Events	An event is the special type that is used to trigger a function when a condition is met.
Struct	Struct is custom defined datatype that is used to combine a multiple number of a variable into one single type.
Enum	An enum datatype holds a multiple number of values that can enclose a multiple number of a finite set of values.

The deployment of the smart contract on the Ethereum platform will be discussed in Chapter 3 in detail.

2.6 Ethereum Transaction

In Ethereum a transaction is nothing but a signed data package that is sent to the external account (Ethereum/wiki. n.d.). The transaction parameters from the receipt of the message, signature that identifies the sender, the amount of ether transferred from the sender to receiver, an optional data field, start gas and gas price. The start gas price will decide amount of computation needed to perform the transaction. The gas price will represent the current gas value of the Ethereum platform. A contract will have the capacity to send message to another contract that is live on the blockchain. In a nutshell, an interaction between the Ethereum accounts is called a transaction and the interaction between the contract is called as the message.

In Ethereum all the state information is stored in the account, which is stored on the blockchain. The user needs to create an account before deploying contract in Ethereum. Thus, each of the smart

contracts deployed in the Ethereum is linked to an account. An account is a 20-byte address and contains the information such as Ether balance and the contract code.

An Ethereum account will contain four fields

1. *Nonce* - A counter variable that makes sure the each transaction is processed once.
2. *Ether balance* - The amount of Ether available in the account.
3. *Contract code* – The contract information deployed by the account.
4. *Account Storage* – empty by default.

In Ethereum the contract can be accessed with the help of the account information and the contract code that is linked along with it.

2.7 Ethereum State Transition Function

In Ethereum all the transaction that happens between the accounts needs to validate before it gets committed to the block. The Ethereum state transition function can be defined as

APPLY (S, TX) -> S'

Where S is the initial state, TX is the transaction and the S' is the state change after the transaction. When a state has been changed from S to S' the following conditions need to satisfy to commit the transaction (Ethereum/wiki. n.d.).

1. "*The Transaction should be well formed. The signature should be valid, and the nonce matches the nonce in the sender's account. If the signature is not valid then return an error.*
2. *The transaction fee is calculated using the startgas and gasprice. If the sender does not have the proper funds for the transaction fees then return an error.*

   *Computation fees = startgas price*gasprice*

   When the sender does not have the sufficient funds in their account to pay for the computation the transaction is invalidated.

3. *Transfer of the transactional value is effected from the sender to the receiving account. If the transfer fails, then revert the transaction* (Ethereum/wiki. n.d.)*".*

When the transfer of the transaction value to the receiving account is completed, the sender pays the gas consumed to the miners. In Ethereum all transactions are validated and stored in the transaction list. If there is any illegal transaction, then the platform will revert the state back to its previous state and nullifies all the transaction that had happened later (Ethereum/wiki. n.d.). The nullified transaction is not added to the blockchain. The only difference between the Ethereum blockchain and the bitcoin blockchain is that the Ethereum blockchain will store the recent transactional state and the transaction list.

2.8 Block Validation

The Ethereum blockchain will contain the most recent state and the copy of the transaction list. Each block will contain the *block number* and the *difficulty threshold* value (Ethereum/wiki. n.d.). The basic block validation algorithm is as follows

1. *"Check if there is a previous block exist and it is a valid block.*
2. *The time stamp on the previous block should be lesser than the new block that is created.*
3. *The block number, difficulty threshold, transaction root, uncle root, and gas limit of the previous block are valid.*
4. *Check the proof of work algorithm is valid on the block.*
5. *Check the root of the tree is valid."*

If the above conditions are validated, then the block is appended into the chain. In Ethereum all the block information is stored in the tree format called as *patricia tries* (Ethereum/wiki. n.d.). A *patricia tree* is a modified version of the *merkel tree* which is used to store the blockchain information.

2.9 Merkel Patricia Tries

The data structure that is used to store the information in the Ethereum platform is called *merkel patricia trie* (Ethereum/wiki. n.d.). A *merkel patricia trie* is a modified version of a radix tree and introduces a couple of modification to boost the performance. In Ethereum the following modification is made to the *merkel patricia trie* to boost the performance.

1. All the node information is hashed and the reference to the node is done based on the hash value.
2. Several node types are introduced to store the information. The types of leaf nodes include empty node, standard leaf node and extension node.

Since the root node is the fingerprint of the entire data, a simple lookup can be implemented to retrieve the information from the tree. Thus making the *patricia trie* as the optimal data structure to store and retrieve the information in the blockchain.

2.10 Universal Business Language

The Universal Business Language (UBL) defines a royalty-free library of XML documents that supports the standard business process (McGrath, T., Holman, K., & Bosak, J. n.d.). UBL is owned by OASIS, a non-profit organization dedicated to the open development of public XML standards. The international and domestic supply chains such as procurement, purchasing, transport, and other supply chain management functions are described with the help of UBL.

UBL provides a language that allows the different business application to exchange information using a common format. The XML syntax provides an easy approach for the software developers to migrate from different business standard formats. The royalty free nature provided by UBL made the companies make profit easily. UBL provides an entry point for small and medium sized business to move into electronic format. One of the best aspects of the UBL is that

the design can be customized to meet the requirements of the individual organization. The UBL schemas are modular, reusable and extended from the standard XML.

Several European public procurement frameworks have accepted UBL and all the major European Union countries have implemented UBL in their government agencies. The European common framework had accepted the UBL for their transport and logistics domain. Beyond the scope of the supply chain, UBL can be configured to suit any industry or business. Since UBL is available with open access there is no registration or approvals are required. In this project we will implement business processes specified by UBL in Tendering, Catalogue, Quotation, Ordering and Billing. We transform the use cases into a smart contract and then deploy. The detailed description about the implementation is discussed in the chapter 3.

Chapter 3 - Software Implementation of UBL use cases

In this chapter, we will discuss the implementation steps that are involved in the development of the UBL uses cases. We will review the design pattern used in smart contracts, development framework and the methods involved in converting the smart contract into a complete decentralized application. The UBL use cases that are implemented in this project are Tendering, Catalogue, Quotation, Ordering and Billing operations. In the selected use case the Tender is the most complex use case. This is due to the reason that the tender had operations like submission of tender, processing the submitted tender and awarding the tender. The rest of the selected UBL use cases are simple to implement. This section will discuss the detailed implementation of the use cases.

3.1 Design Pattern

The *state design pattern* is a behavioural design pattern that implements a state machine and will act differently when there is a state change (Design Patterns and Refactoring. n.d.). The smart contract written in the solidity programming language can be designed in such a fashion that it can act as a state machine. The smart contract will behave differently at certain stages.

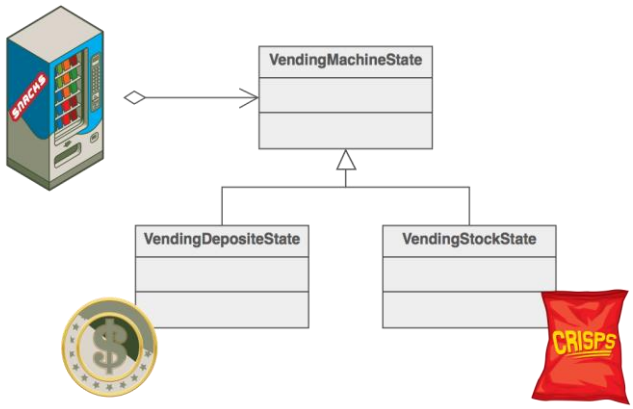

Figure 2 State design pattern (adapted from https://sourcemaking.com/design_patterns/state)

A simple function call is written inside the smart contract that is used to trigger a state change of a smart contract. Figure 2 is the simple example of the smart contract that follows the state design pattern. Figure 3 is a simple state diagram that represents a smart contract. The smart contract will act differently at different stages.

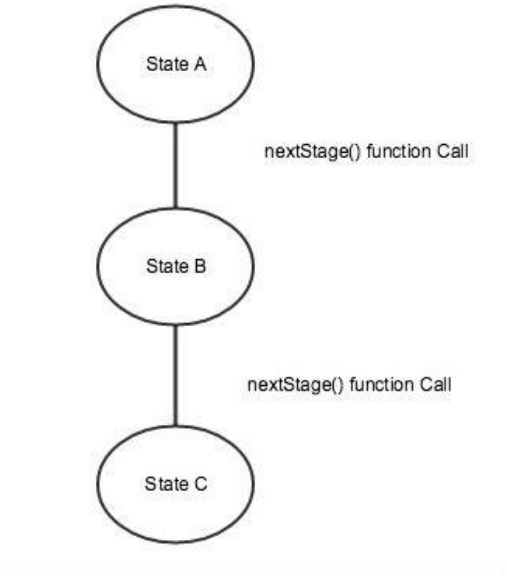

Figure 3 State design pattern

```
contract StateDesign {
    enum Stages {
        Stage1,
        Stage2,
        Stage3
    }
    Stages public stage = Stages.Stage1;
    modifier atStage(Stages _stage) {
        if (stage != _stage) throw;
        _
    }
    function nextStage() internal {
        stage = Stages(uint(stage) + 1);
    }
    function stage1()
    atStage(Stages.Stage1)
```

```
    {
    }
    function stage2()
    atStage(Stages.Stage2)
    {
    }
    function stage3()
    atStage(Stages.Stage3)
    {
    }
}
```

Figure 4 Sample code snippet for state design pattern

Figure 4 is a simple code snippet example of the state design pattern. The smart contract is written with the help of *solidity* language. In the above code, *enum* data type holds the list of different stages hardcoded inside it. The *enum* data type will be holding all the internal states that are needed to implement our UBL use cases. The above program is hard coded with three stages in the *enum* datatype. The function *nextStage* is used to move the smart contract stage. Each state represented in enum will have a separate function that will be called at a different stage of the smart contract.

The information that is provided by the UBL documents is stored inside the smart contracts as state variables. A simple get and set method is used to store and retrieve the information from the smart contract.

```
contract StoreNumber {
    uint storedData;
    function set(uint x) {
        storedData = x;
    }
    function get() constant returns (uint retVal) {
        return storedData;
    }
}
```

Figure 5 Sample code snippet to store single variable in the smart contract

Figure 5 represents a simple code snippet that can be used to store one single variable in the blockchain. The *storeData* is a state variable that can be used to store information in the blockchain. The set and get method can be used to access the information of the state variable from outside of the smart contract.

Once the contract is deployed in the blockchain any decentralized application can access the smart contract with the address location where the contract is deployed. The deployment of the smart contract can be done manually or can be assisted through frameworks such as *Embark* or *Truffle*. We describe the manual deployment and then deployment through *Embark* framework that was used in this project. For each one, we assume that the Ethereum framework has already been downloaded and installed and also that an Ethereum node has been created using the geth client.

3.2 Manual Contract Deployment

Deploying the smart contract in the block chain is one of the important steps in developing the Ethereum platform. The smart contract can be deployed manually or automatically with the help of any framework. In this section, we will discuss the manual deployment.

Once the smart contract is created, perhaps using a simple text editor, it needs to be compiled using the online solidity compiler, which generates the byte code. The byte code that is generated is then loaded into the Ethereum platform. Currently, there is only one compiler available that is on *ethereum.github.io*. The compiler generates a load file that contains the byte code with pre-fix and post-fix javascript instruction.

Figure 6 Software Stack

The loader, invoked by the *geth* client command loadscript, is used to load the byte code on the Ethereum node on the blockchain at a particular address. The delay in contract deployment will take time depending on the processing power of the Ethereum node. Once the loadscript operation is successful the *geth* client will return the address of the contract deployment.

```
>loadScript("<local path to the .js>");
>Contract mined! address: 0xdaa24d02bad7e9d6a80106db164bad9399a0423e
```

Figure 7 smart contract with loadscript command

The contract that is deployed in the Ethereum node can be accessed with the help of Web3 API layer. Figure 6 represents the software stack used to develop a decentralized application, in which the Ethereum node will be acting as the base layer. The smart contract will be deployed on the top of the Ethereum node. Web3 API layer will act as an interface between the smart contract and the front end. The front end of the application can be designed with the help of modern web development frameworks (e.g *HTML5, Java,* and others).

3.3 Contract Development and Deployment with Frameworks

To simplify the development and deployment of smart contracts, frameworks was developed, such as Truffle and Embark. Embark is the framework that is used in this project to

develop and deploy the decentralized application in the Ethereum platform. The Embark framework takes care of the background activity that is needed for the contract deployment. Figure 8 represents the file structure of the Embark file structure.

```
app/
  |___ contracts/ #solidity or serpent contracts
  |___ html/
  |___ css/
  |___ js/
 config/
  |___ blockchain.yml #environments configuration
  |___ contracts.yml  #contracts configuration
  |___ server.yml     #server configuration
 spec/
  |___ contracts/ #contracts tests
```

Figure 8 Embark file structure

The contracts are written in the *contract* folder and are stored as files with *solc* extension. The front-end code is placed in the *html* and *css* directory. The Web3 API javascript binding between the front-end and the Ethereum node are placed in the *js* directory as simple javascript files. Embark framework will take care of the deployed contract and will make the use of the deployed contract when there is a need for the contract in the application.

The *config* directory will contain *yml* files that are used configure the setting of the project. The three major yml files are *blockchain.yml*, *contracts.yml* and *server.yml*. The *blockchain.yml* file is used to specify the configuration of the Ethereum node. The *contracts.yml* file will have the specification of the names of the contract, gas limit and the gas price needed for the contract to be deployed in the project. The *contracts.yml* file will be automatically edited by the Embark framework when there is a change made in the contract file. The *server.yml* file will contain the host address and the port number where the project is deployed. Apart from the configuration

embark is quite flexible and allows custom file configuration which can be used to deploy any application written in embark to be deployed in the server.

The deployed application can be accessed with the help of the internet thus making the decentralized application completely available to everyone. A user accessing the decentralized application does not need to have a *geth* client. In this project, we use the embark framework to develop the UBL use cases and test its performance.

3.4 Overview of UBL use cases of the Project

Each UBL document, an XML file, may be used in the different business processes and for the different purposes. There are many UBL documents defined that are used by the various use cases. One document may be used for many use cases, and a use case may use many documents. We only implement documents that are required for the selected use cases. Our selected use cases are listed in the subsequent sections. The Tender use case is the most complex and its business logic includes the process such as submission of the tender and awarding of the tender. The rest of the use cases are simple.

3.4.1 Tendering:

Tender is the process in which an organization, such as government or financial entity, invites bids for a project to be completed within a finite set of the deadline (Tender Definition | Investopedia. 2003). Figure 11 explains the list of steps/stages that are carried out in the tender use case. The process of Tendering can be subdivided into contract information preparation, contract information notification, the invitation of tender, submission of qualification information, submission of tenders and awarding of tenders (Universal Business Language Version 2.1. n.d.).

In our project, we implement submission of tenders and the awarding of tenders. In the submission of tenders, the tenderer needs to formally submit a tender document to the contracting

authority. Once the submission is completed the contracting authority will respond back with a receipt. Figure 9 shows the submission of tenders.

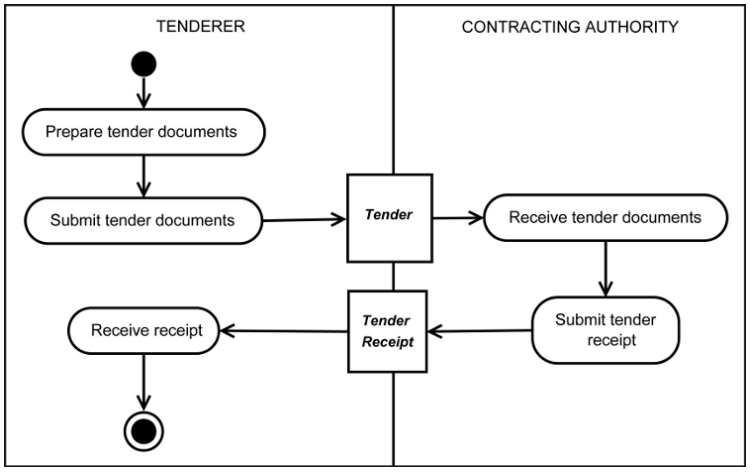

Figure 9 Submission of Tender

(Adapted from http://docs.oasis-open.org/ubl/os-UBL-2.1/art/UBL-2.1-Tender-SubmissionOfTenders.png)

The Tender document contains information such as *UBL Version Identifier, Customization Identifier, Profile Identifier, Profile Execution Identifier, Identifier, Indicator, UUID, Tender Type Code, Contract Folder Identifier, Issue Date, Issue Time, Contract Name, Note, Period, Document Reference, Signature, Party, Document Reference, Party, Contracting Party, Customer Party and Tendered Project*. The contracting authority will issue the tender receipt to the tenderer. The important variable to the tender receipt would be the issue date and the issue time.

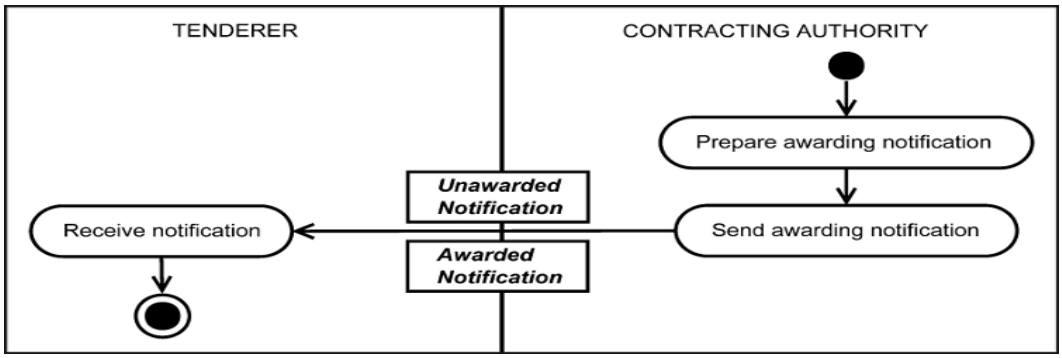

Figure 10 Tender notification

(Adapted from http://docs.oasis-open.org/ubl/os-UBL-2.1/art/UBL-2.1-Tender-AwardNotification.png)

The contracting authority notifies all the tenderers about their tender status. The outcome of the tender is success or failure. The successful tenderer gets the awarded notification and the failure to win the tender would get the un-awarded notification. Both the awarded and un-awarded notification will have the same set of variables. The only difference would be that the awarded notification document will contain the *signature* variable added along with it.

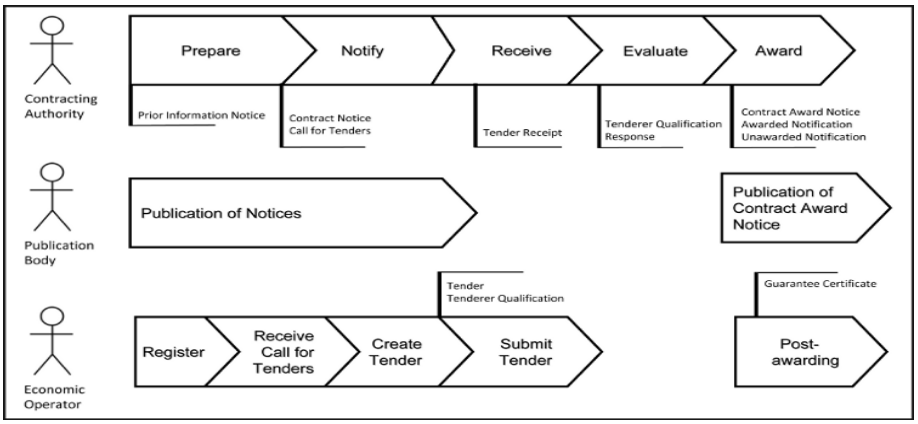

Figure 11 Overview of Tender use case

(Adapted from http://docs.oasis-open.org/ubl/os-UBL-2.1/art/UBL-2.1-Tender-TenderingProcess.png)

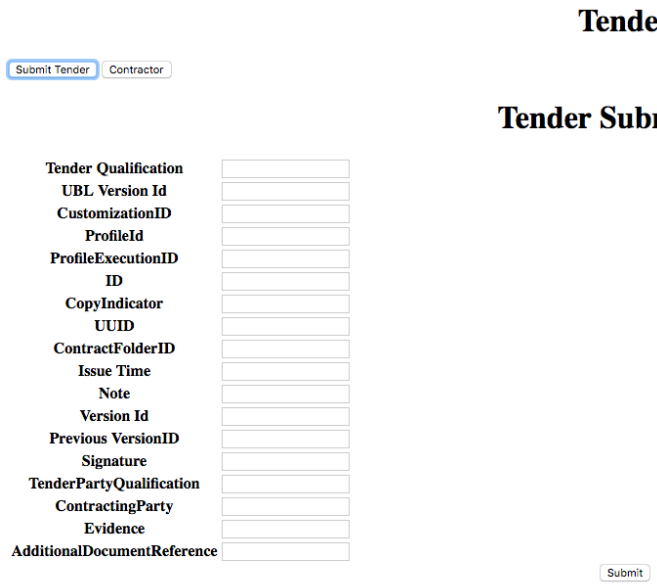

Figure 12 Tender Submission form

The Tender uses case implementation is divided into two functionalities. The first functionality is for the tenderer to submit the tender information to the contractor. The second functionality enables the contractor to retrieve the tender submission and announce the winner of the tender. Figure 12 shows the form that is used to submit the tender information to the contractor.

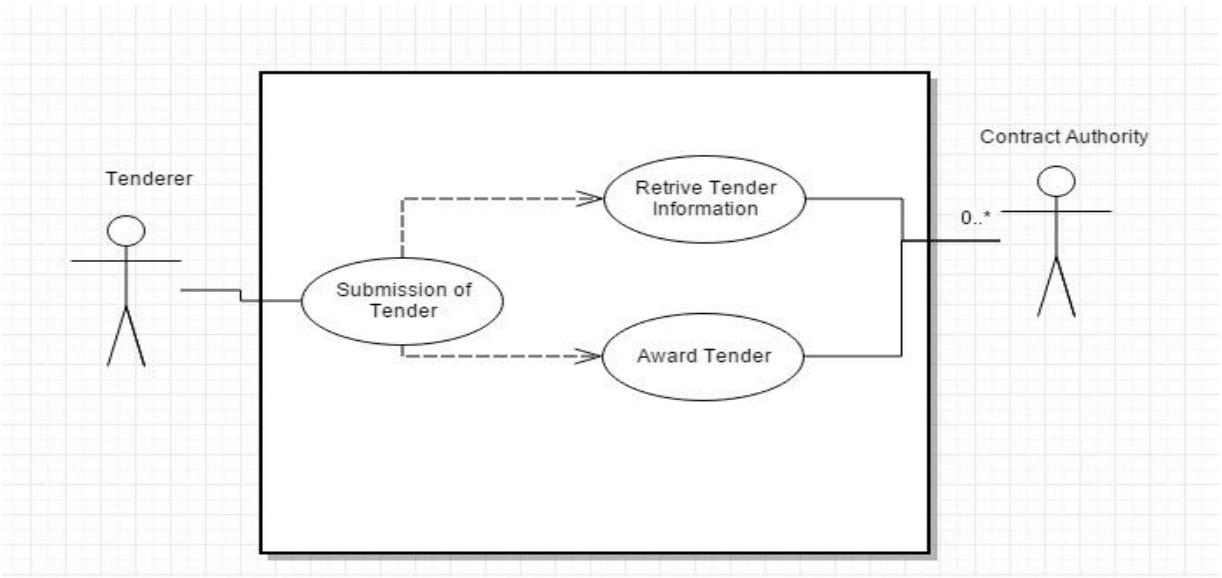

Figure 13: UML diagram of implemented tender use case

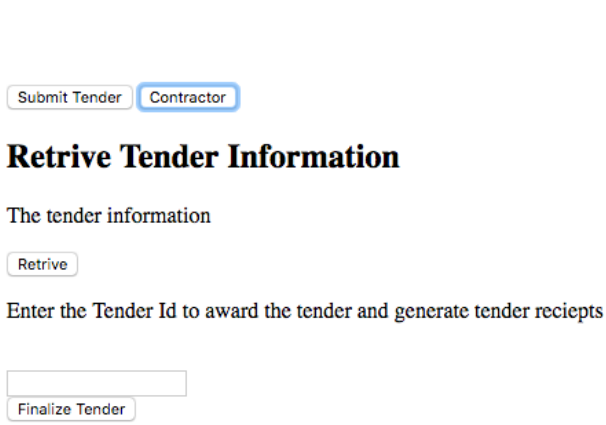

Figure 13 Contractor Portal

The contractor can retrieve the specific tender information, which has been submitted, using the tender ID. The contractor can analyze the submitted tender information and award the tender. Figure 13 is the contractor portal that is used to retrieve and award the tender information.

3.4.2 Catalog:

A catalog is used to store all the transaction records carried out over a period of time. In the contract, the catalog will be storing all the transactional information that are carried out by other business process.

The business processes that are linked to the catalog are catalog request, application response, catalog item specification, catalog pricing update and catalog deletion. In our project, the catalog is implemented as a simple logging tool that will record all the information that are made by the other contracts. The information that is stored in the catalog is the information that is carried out by the other business use cases.

In this project, the catalog is implemented as a logger that will record all the transaction that are taking place in the network.

3.4.3 Quotation:

The quotation is the standard business process whose purpose is to invite suppliers to bid on a specific product or service. The information about the product or service is written as a smart contract and deployed on the Ethereum platform.

A simple call to the contract is used to fetch the information from the contract. A timed transition function is written inside the quotation which will make the quotation expire after a certain period of time. The customer party can request for the quotation when it is available in the block chain. When the time expires the quotation will be automatically destroyed from the block chain.

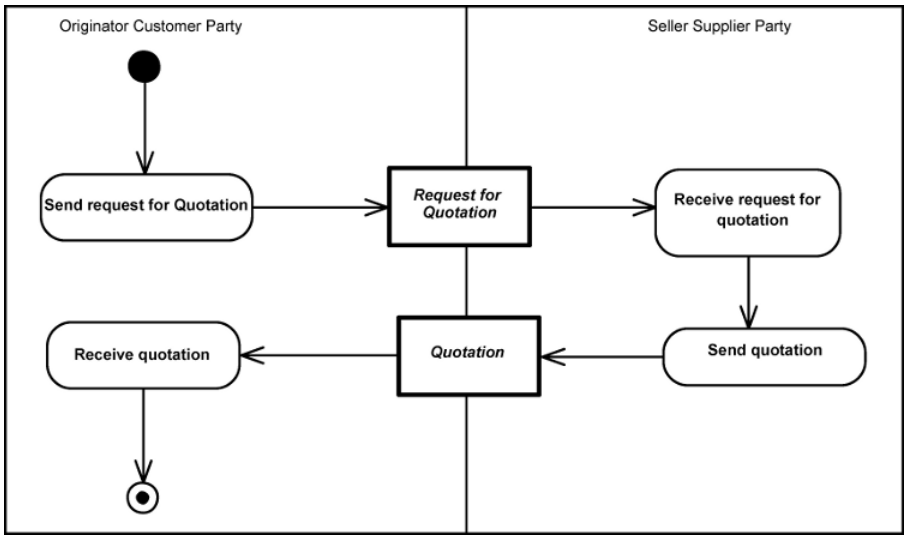

Figure 14 Quotation

(Adapted from http://docs.oasis-open.org/ubl/os-UBL-2.1/art/UBL-2.0-SourcingBuyerInitiatedProcess.png)

The quotation use case contains only one document associated with it which is the *request for quotation* (Universal Business Language Version 2.1. n.d.). The customer party can request a quotation for a product or a service to the seller.

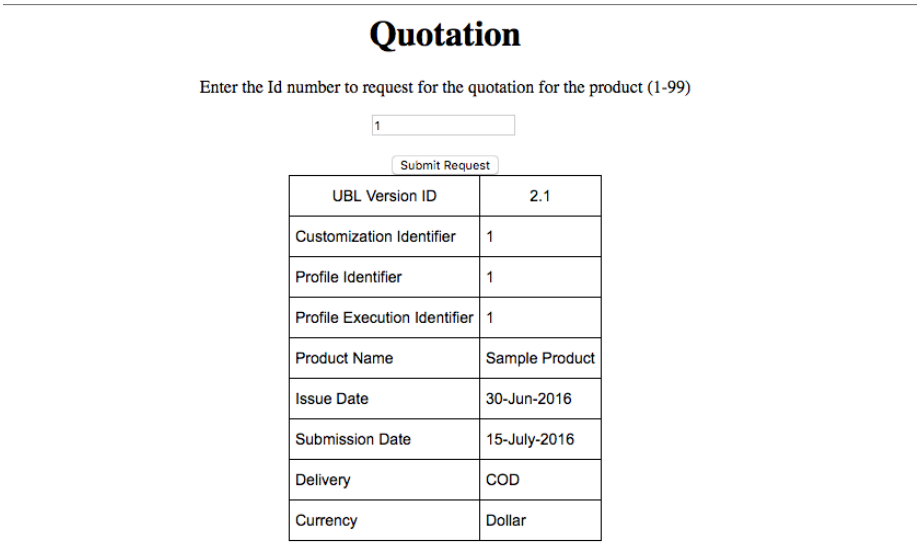

Figure 15 Quotation Submission form

The list of variable associated with the quotation are *UBL Version Identifier, Customization Identifier, Profile Identifier, Profile Execution Identifier, Identifier, Indicator, UUID, Issue Date, Issue Time, Submission Due Date, Note, Currency Code, Line Count, Period, Document Reference, Signature, Customer Party, Supplier Party, Delivery, Delivery Terms, Country, Contract and Request For Quotation Line*

In the implementation of the quotation, the customer party and the seller party will have a different screen. The seller party can create information about the product or service. Based on the product or service that has been created by the seller the customer can request for a quotation and can view the details of the product or service. The quotation is extended to the ordering and billing and will be discussed in the subsequent sections.

3.4.4 Ordering:

The ordering is the collaboration that creates a contractual obligation between the seller and the buyer. Ordering in the UBL can be subdivided into order request, order response, order change and order cancellation. In this project, we implement in the steps carried out in the order

request. In the order request, the buyer party will be placing an order to the seller party. When the order is received, the seller party can perform any operation on the order request.

The order request will have more than fifty different variables associated with it. The following are the variables that are associated with the order request are BuyerCustomerParty, SellerSupplierParty, TaxTotal, AnticipatedMonetaryTotal and OrderLine (Universal Business Language Version 2.1. n.d.).

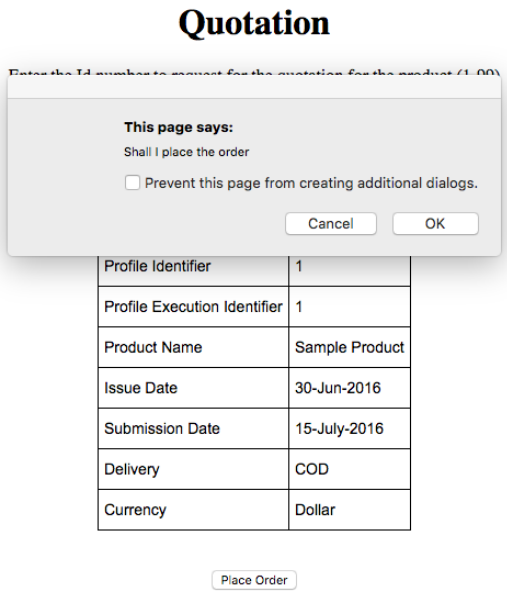

Figure 16 Order Submission

Once the order has been placed to the seller. The seller can accept or reject an order. The response of the order can be expressed using the order response document. In this project, we will be implementing the order request. The seller responds back to the buyer with the order response.

In this project ordering use case is implemented along with the quotation. Once the buyer requests the quotation from the seller, the buyer will be given an option to order the product or service.

3.4.5 Billing:

Billing is the process in which a request is made for the payment of the goods or service that has been ordered or received. In the Billing use case, we will do the implementation of the statement process that is carried out between the buyer and the seller (Universal Business Language Version 2.1. n.d.). The documents that are linked to the billing are Invoice, credit note, debit note, and application response.

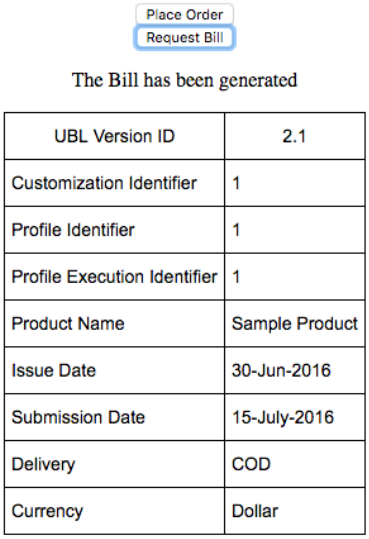

Figure 17 Request Billing

The process of billing can be subdivided into traditional billing and self-billing. Each billing operation can be done separately using the credit or debit note. The billing operation, when carried out using the credit and the debit are completely different. The UBL document specifies a separate operation under billing called as a *remainder*. The remainder operation is carried out when the buyer party has pending payments. In our project, we implement in the statement operation

which is used to get the status of the billing information. Figure 12 represents the UBL billing procedure.

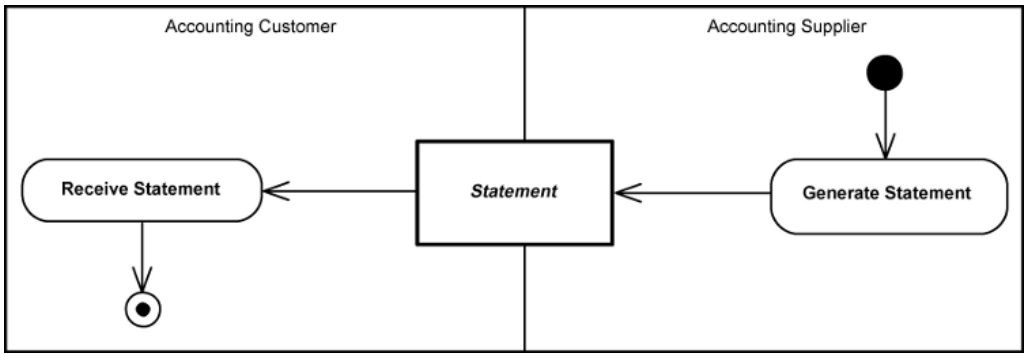

Figure 18 UBL Billing Statement processing

(Adapted from http://docs.oasis-open.org/ubl/os-UBL-2.1/art/UBL-2.0-ReportStateofAccountsProcess.png)

In the billing use case is an extended version of the ordering. Once the order request is completed, the bill is automatically generated to the customer.

3.5 Summary

In this project, we implemented the business use cases related to Tender, Quotation, Ordering, Billing and Catalog. A simple prototype is developed for the each business use case. The implemented prototype can be used to examine the suitability of deployment of business logic expressed in UBL in Ethereum. Our selected subset of the use case from reference ensured that our selected set had one of the most sophisticated and complex cases. In our selected use cases the Tender is the most complex use cases to be implemented. Thus we conclude that Ethereum is a suitable platform for the implementation of UBL use cases.

Chapter 4 - Performance and Scalability

In this chapter, we will review the performance and scalability of the Ethereum as a platform. Ethereum suffers from the flaw that every transaction needs to be processed by every node in the network. This affects the performance and the scalability aspect of the Ethereum platform. This chapter is designed in such a way that we will examine the factors that affect the performance and scalability of the Ethereum.

4.1 Performance Parameters

An Ethereum node that is participating in the mining operation is expected to generate some amount of *ether*. The generated ether is used to run computation tasks or validated transactions that are happening in the blockchain. The two important parameters that affect the performance of the Ethereum are the hash rate used in the mining algorithm and the difficulty threshold value used in the proof of work algorithm (Ethereum/wiki. n.d.). In this subsection, we will review the parameters that affect the performance of the Ethereum platform.

4.1.1 Hashrate

When a node participates in the network and performs the mining operation the node will automatically validate the transactions in the blocks that are newly created. The ether generated by the mining operation that is done in the node is controlled by the parameter called as *hash rate*. The *hash rate* is the number of *nonces* tried per second normalized by the *hash rate* of the network. The *hash rate* will determine the number of transactions that need to be validated by the node. The average *hash rate* increases as the number of nodes participating in the network increases.

4.1.2 Difficulty Threshold

In Ethereum miners are responsible for checking blocks for validity and then forwarding them onto the blockchain. Miners use the Etash algorithm for that purpose. The difficulty

threshold value will decide the amount of time needed for the new block to get finalized. The difficulty threshold value is controlled by the Ethereum platform and, the current Ethereum platform generates a new block for every 12 seconds (What is Ether. n.d.). Once the new block is generated the newly created block is validated by the nodes participating in the network.

4.1.3 Hardware

The mining operation performed in the node will directly depend on the hardware. The mining operation can be done in both CPU and GPU. The *Ethash* algorithm is designed in such a way that it can even run on a slow CPU environment. The number of the hash operation performed by the Ethereum will directly depend on the memory bandwidth of the hardware. The memory bandwidth is defined as the rate at which the data can be stored in the hardware memory. The hardware with higher memory bandwidth will have a higher advantage compared to hardware with slower memory bandwidth. The performance of the GPU has by default more advantage when compared to the node which is run on the CPU. The mining operation is started only after generating the DAG. When the node does not have the capacity to generate the DAG the mining operation will not start. Thus hardware disk space is also another parameter that affects the performance of the Ethereum.

Crypto Junction (Cryptojunctioncom, 2016) conducted a detailed study on the analysis of different GPUs in running the Ethash algorithm. Their study clearly showed that the mining algorithm Ethash had a different performance rate based on the hardware the algorithm was tested. This clearly showed that the hardware is an important parameter that affected the performance of the Ethereum. Crypto compare (Cryptocomparecom, 2016) has provided a simple guide to select the hardware to perform the mining operation.

4.2 Transactions Per Second

In Ethereum all the transactions that are carried out in the network need to be validated before it gets finalized. (Etherchainorg, 2016) Ether chain reports all the statistical information that is happening in the live Ethereum blockchain. The current *Transactions Per Second* (TPS) lies between the 0.03 and 0.81.

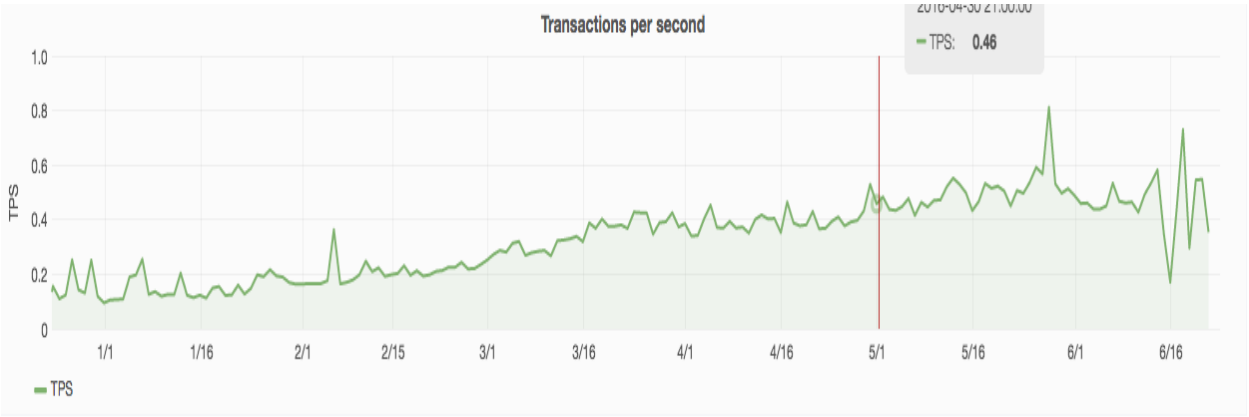

Figure 19 Transactions per second reported by Ethereum IO

Since the transactions are not validated concurrently delay time of the transaction getting approved will increase as the number of nodes increases. If the delay time increases, then the application that is developed on top of Ethereum will have more inconvenience (Ethereumorg, 2014).

"Transactions comes in from different peers, in no particular order, since there is no centrally managed queue. In addition, at average intervals of between 12 seconds (Ethereum) and 10 minutes (bitcoin), a new block comes in, confirming a set of transactions in a specific order. A node will probably have seen most of a block's transactions already, but some may be new. Either way, the order of the transactions in the block is unlikely to reflect the order in which they arrived individually. And since the order of transactions might affect the outcome, this means transactions cannot be processed until their order in the blockchain is confirmed."

In Ethereum the transaction is not validated concurrently by the node this due to the reason that the each transaction depend on each other (Multichaincom, 2016). A set of transactions can

be processed concurrently if the transactions are atomic and do not depend on each other. In Ethereum transaction comes from different peers in no particular order and does not have a centralized queue system to manage the order of transactions. This causes lots of problems because if each node processes the transaction in a different order, then the end result will be different for each transaction.

In Ethereum a transaction is only valid if it is committed into the block and added to the blockchain. On average, the time taken for the Ethereum to create a new block and confirm a set of transactions is 12 seconds (Ethereumorg, 2014). The transaction cannot be processed until their order in the blockchain is confirmed. The current implementation of the Ethereum does not process the unconfirmed transaction. If the node wants to process a new transaction, then the node needs to reference the newly created block and make the update to its current transaction. Thus making the concurrent implementation in the Ethereum an impossible task to implement. In a nutshell, an individual node needs to wait for 12 seconds before seeing the result of its own transaction. As (Multichaincom, 2016) Vitalik Buterin stated that the key problem of the Ethereum and the decentralized application is the latency.

Proof of Stake is an alternative protocol that has been proposed to replace the proof of work algorithm. The proof of work algorithm is based on the mining method carried out by the nodes and the proof of stake algorithm is based on the amount of stake (*ether*) that miners hold on the network. Proof of Stake does not require complex computation to be carried out. The consensus is achieved by placing bets and thus by removing the heavy computation involved in the proof of work, the barrier to the consensus is lowered so that a new block can be formed very easily. Ethereum had proposed to implement the proof of stake algorithm in their *casper* release.

The proof of stake algorithm on Ethereum has the following working principles (Ethereumorg, 2015)

1) When the new block has been created the validators in the network will place a small security deposit behind the block. When the largest percentage of the security deposit is placed on a block, then the block becomes a valid block.
2) The concept of placing the security deposit behind the block extends to the concept of betting. A block gets approved based on the number of bets placed by the validators in the network. If a block gets 2/3rd of bets placed by the validators then the block gets approved and added to the chain.
3) Once a block gets approved there is another voting process is taken place to decide the new block is added to the network or not. The voting process will involve 2/3rd of the validators need to approve the block to be added.

Proof of Stake algorithm has reduced the huge amount of computation power since the nodes that have the stake in the network will take care of the validation process. The implementation of the Proof of Stake algorithm is one of the best solutions that can be provided for the scalability and performance issues faced by the Ethereum.

4.3 Scalability

In Ethereum scalability is the most important factor that needs to be addressed. The current blockchain size of the Ethereum is over 16 GB size (Cryptomining-blogcom, 2016). The node will require the entire chain to participate in the mining operation. When the number of applications and the users interacting the blockchain increases, then the size of the blockchain will increase. The following list of problems arises when the chain size is very large

1. When the block chain size becomes very large (e.g., 100 TB), very few nodes will have the capacity to store such a large file size. This means that very few nodes will have the capacity to mine.

2. If the block chain size is very large, the time taken for the mining algorithm to generate the DAG will take a very long time. Hence this will affect the initialization of mining operations in the node.

The following are some of the proposed methods to deal with the scalability issues related to the Ethereum.

Ethereum white paper (Ethereum/wiki. n.d.) proposed that creating a lower bound on the number of mining nodes will solve the problem of scalability. If the number of mining nodes in the network is decreased, then the validation time taken for the nodes will also decrease. Hence allowing a fewer number of nodes to perform the mining operation will provide a naïve solution to solve the scalability issue. Though this solution contradicts the previous assumption that all the node participating in the network need to be a mining node. But reducing the number of nodes performing the mining operation can reduce the transaction approval time and improve the performance.

Vitalik Buterin proposed an alternative technique for achieving scalability in Ethereum (Buterin, 2015). The scalability can be achieved by splitting the transaction into small portions of the block chain state. If the transaction collection is valid then the group of transactions is added to the block. The proposed method validated transaction in groups rather than the validating the transaction individually.

Scalability of Ethereum is currently in its nascent stage, So we need to wait for a decision from the Ethereum foundation to make changes in the future versions of the Ethereum to solve the scalability issues.

4.4 Data Structure

The data structure used to store and retrieve the information is also a factor that affects the performance and scalability of the Ethereum platform. In Ethereum a *merkel patricia tries* are used to store and retrieve the information from the chain. The merkel patricia tries is a modified version of the radix tree data structure.

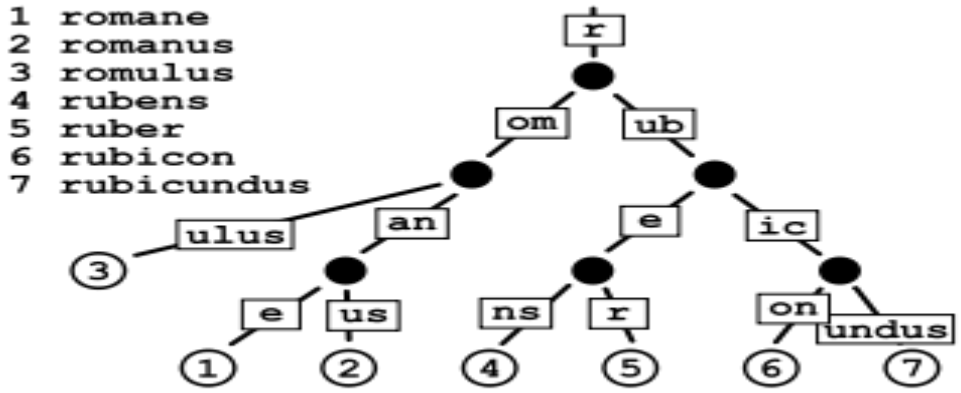

Figure 20: Radix tree (Adapted from http://www.birdland.co.jp/wordpress/wp-content/uploads/2015/07/320px-Patricia_trie.svg_.png)

Though the radix tree is efficient in storing and retrieving the string information. The Merkel Patricia tries make the following changes to store the blockchain information. All the information stored in the node is hashed and the information is referenced by the hash value. *Merkel Patricia tries* adds different types of nodes such as empty node, extension node, and standard leaf node. The idea of a different node type is to optimize search and retrieve operation carried out in the tries. Figure 20 is a simple example of the radix tree and information stored in it. The simple lookup function will traverse the tree and produce the result. Hence the same concept of the radix tree is used in the *Patricia tries* efficiency of the insert, lookup, and the delete operation

can be done in O (log N) time (Ethereum, 2016). Although the Ethereum project is open source, the source code for the tree manipulation has not been released and thus we are unable to confirm whether the statement, that the tree operations are O (log N), is for the average or for the worst case – and we were not able to confirm this in the literature.

4.5 Future Direction

Though Ethereum foundation has the complete control of the changes that needs to be made in the Ethereum protocol, the Ethereum yellow paper has proposed the following changes to the Ethereum blockchain (wood, G. 2016).

1. *"The modification in the validation algorithm provides a solution to the scalability of the Ethereum platform. The nodes participating in the mining operation will not be forced to store all the state.*
2. *Replacing Proof of Work algorithm with the Proof of Stake in the future releases of the Ethereum will have a huge impact on the performance and the scalability.*
3. *The nodes would be given an age and the validation should be done by the nodes after it attains a certain particular age.*
4. *When a node wants to participate in the mining operation. The node does not need the entire chain to participate in the mining.*
5. *A compressed archive of the entire chain is maintained so that If a node wants to access the chain can access the archive and retrieve the information from it.*
6. *A lightweight chain can be introduced to perform the computation and establish the nodes.*
7. *A priority set is created for the nodes so that certain node will have the higher verification rate when compared to the other nodes that participating in the mining operation.*

8. *Blockchain compression can be used to remove the nodes that have not sent or received any transaction for a considerable period of time."*

The changes that we propose for the Ethereum platform in the future releases are as follows:

1) A standard programming language needs to be adopted for the Ethereum platform. Currently, Ethereum supports multiple programming languages for the development of the smart contract. Thus, by adopting one language for the development will encourage the developers to think in a unified way.
2) Ethereum foundation needs to work on better development tools to attract more developers into developing the decentralized application. The Ethereum foundation provides *Ethereum Integrated Development Environment*, which is not easy to use to develop Decentralized applications.
3) Ethereum needs to support special data types for date and floating point values.
4) Ethereum platform needs to provide basic library functions for string and file handling.

Though the current Ethereum blockchain has performance and scalability, issues related to it. But in the future with continuous improvements, the issues can be tackled

Chapter 5 - Conclusion

Ethereum protocol was originally developed as an upgraded version of the existing cryptocurrencies. Beyond the transfer of value, Ethereum provides a framework to build the decentralized application. Ethereum in the future will provide a decentralized computation environment where the developer can set up independent Ethereum nodes to develop and deploy the decentralized application. Ethereum is currently an open-ended project, which means that based on the requirements of applications, new features will be added to the platform. Current financial and nonfinancial organization are exploring the Ethereum capacity to implement their products and services.

In this project, we explored whether UBL use cases can be ported to the Ethereum platform. The UBL use cases were developed by OASIS (OASIS, 2016), which is a nonprofit consortium that drives the development, convergence and adoption of open standards for the global information society. As the OASIS standard includes 73 use cases, we selected a small subset, while ensuring that it contained one of the most complex cases, for implementation using Ethereum. We created a template to be followed for use cases in Ethereum and then used it to implement the selected use cases- thus providing a proof of concept that the OASIS standard UBL use cases can be implemented in Ethereum. Along with the implementation of the UBL use cases, we reviewed the performance and scalability of the Ethereum platform.

As the current version of the Ethereum platform has issues related to performance and scalability, the future Ethereum releases need to address them and fix the drawbacks. If the Ethereum continuously evolves as time progresses, then it is likely that it will be the future of building the decentralized application. If Ethereum gets more amount of funding and support from the large business corporations then it is going to be a force to reckon with.

5.2 Future work

Currently, Ethereum is completely open-ended and the technology is not stable. There are lot of modifications that need to be done to overcome the limitation of performance and scalability of the platform. The changes in the future versions of the Ethereum platform are done by the Ethereum foundation. Ethereum is currently in its nascent stage and we need to wait for the future releases of the platform.

Reference

Bitcoinit. (2016). *Bitcoinit*. Retrieved 27 June, 2016, from https://en.bitcoin.it/wiki/Why_a_GPU_mines_faster_than_a_CPU

Bitcoinminingcom. (2016). *Bitcoinminingcom*. Retrieved 27 June, 2016, from https://www.bitcoinmining.com/ethereum-mining/

Buterin, V. (2016). *Ethereum Homestead*. Retrieved 4 June, 2016, from www.ethereum.org

Buterin, V. (2015). *Notes on Scalable Blockchain Protocols*. Retrieved 2 July, 2016, from https://github.com/vbuterin/scalability_paper/blob/master/scalability.pdf

Cryptomining-blogcom. (2016). *Cryptomining-blogcom*. Retrieved 27 June, 2016, from http://cryptomining-blog.com/tag/ethereum-blockchain-size/

Cryptojunctioncom. (2016). *CryptoJunction*. Retrieved 2 July, 2016, from https://cryptojunction.com/ethereum-mining-hardware-comparison/

Cryptocomparecom. (2016). *CryptoCompare*. Retrieved 2 July, 2016, from https://www.cryptocompare.com/mining/guides/how-to-choose-a-gpu

Decentralized Prediction Markets | Augur Project. (n.d.). Retrieved from https://www.augur.net/

Design Patterns and Refactoring. (n.d.). Retrieved June 27, 2016, from https://sourcemaking.com/design_patterns/state

Ethereum. (2016). Ethereum/wiki. Retrieved August 11, 2016, from https://github.com/ethereum/wiki/wiki/Patricia-Tree

Ethereum Project. (n.d.). Retrieved from https://www.ethereum.org/

Etash. (n.d.). *Ethereum/wiki*. Retrieved June 15, 2016, from https://github.com/ethereum/wiki/wiki/Ethash

Ethereumorg. (2014). *Ethereum Blog*. Retrieved 2 July, 2016, from https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/

Ethereumorg. (2015). *Ethereum Blog*. Retrieved 27 July, 2016, from

https://blog.ethereum.org/2015/12/28/understanding-serenity-part-2-casper/

Etherchainorg. (2016). *Etherchainorg*. Retrieved 2 July, 2016, from https://etherchain.org/

Ethereum homestead documentation — Ethereum homestead 0.1 documentation. (2016). Retrieved July 14, 2016, from http://www.ethdocs.org/en/latest/

Get an Ethereum Wallet. (n.d.). Retrieved June 15, 2016, from https://www.weusecoins.com/ethereum-wallets/

Get an Ethereum Wallet. (n.d.). Retrieved June 16, 2016, from https://www.weusecoins.com/ethereum-wallets/

Johnston, D., Yilmaz, S. O., Kandah, J., Bentenitis, N., Hashemi, F., Gross, R., Mason, S. (n.d.). DavidJohnstonCEO/DecentralizedApplications. Retrieved June 15, 2016, from https://github.com/DavidJohnstonCEO/DecentralizedApplications

List of highest funded crowdfunding projects - Wikipedia, the free encyclopedia. (n.d.). Retrieved June 6, 2016, from https://en.wikipedia.org/wiki/List_of_highest_funded_crowdfunding_projects

Multichaincom. (2016). *Multichaincom*. Retrieved 2 July, 2016, from http://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/

Mougayar, W. (2015). *The Business Imperative Behind the Ethereum Vision - Ethereum Blog*. Retrieved June 05, 2016, from https://blog.ethereum.org/2015/05/24/the-business-imperative-behind-the-ethereum-vision/

McGrath, T., Holman, K., & Bosak, J. (n.d.). *OASIS Universal Business Language (UBL) TC*. Retrieved June 16, 2016, from https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl

OASIS (2016). *Committee categorie*s. Retrived July 28,2016 from https://www.oasis-open.org/

The DAO (organization) - Wikipedia, the free encyclopedia. (n.d.). Retrieved June 6, 2016, from https://en.wikipedia.org/wiki/The_DAO (organization)

Tender Definition | Investopedia. (2003). Retrieved June 27, 2016, from http://www.investopedia.com/terms/t/tender.asp

Universal Business Language Version 2.1. (n.d.). Retrieved June 27, 2016, from http://docs.oasis-open.org/ubl/UBL-2.1.html

What is Ether. (n.d.). Retrieved June 15, 2016, from https://www.ethereum.org/ether

Wood, G. (2016). *Ethereum: A secure decentralized generalized transaction ledger homestead*. Retrieved 27 June, 2016, from http://gavwood.com/paper.pdf