UNIVERSITY OF TARTU
Institute of Computer Science
MS(SE) Curriculum

Jazib Sawar

Blockchain-based business process execution on Hyperledger

Master's Thesis (30 ECTS)

Supervisor(s): Marlon Dumas
Orlenys López Pintado

Tartu 2022

Blockchain-based business process execution on Hyperledger

Abstract:

Collaboration of business processes among organisations has recently become increasingly vital in achieving corporate goals. To achieve this, enterprises must manage business processes that extend to multiple organisations. However, lack of trust among stakeholders is one of the significant obstacles to implementing and executing collaborative business processes. Caterpillar is an existing Business Process Management System (BPMS) prototype that utilises Blockchain's immutable behaviour to resolve this problem by executing business processes translated into smart contracts. However, Caterpillar integrates Ethereum, a public blockchain, to manage smart contracts, which has disadvantages (i.e., permissionless, low throughput, and data privacy). Considering the shortcomings of public blockchains, this thesis proposes a solution to integrate a permissioned blockchain platform into Caterpillar. The proposed solution utilises Hyperledger Fabric, which provides a membership-based blockchain network, resulting in fewer nodes than Ethereum. Hence, it enables organisations to perform faster transactions, achieve relatively more data privacy, and customise their network as per their requirements.

Keywords:

Business Processes, Business Process Management System, Blockchain Technology, Solidity smart contract, Hyperledger Fabric

CERCS: P170 - Computer science, numerical analysis, systems, control

Plokiahelapõhine äriprotsesside täitmine Hyperledgeris

Abstraktne:

Organisatsiooni äriprotsesside koostöö on viimasel ajal muutunud ettevõtte eesmärkide saavutamisel üha olulisemaks. Selle saavutamiseks peavad ettevõtted juhtima äriprotsesse, mis laienevad mitmele organisatsioonile. Kuid usalduse puudumine osapoolte vahel on üks olulisi takistusi koostööl põhinevate äriprotsesside rakendamisel ja läbiviimisel. Caterpillar on olemasolev äriprotsesside haldussüsteemi (BPMS) prototüüp, mis kasutab selle probleemi lahendamiseks Blockchaini muutumatut käitumist, täites nutikateks lepinguteks muudetud äriprotsesse. Caterpillar integreerib aga nutikate lepingute haldamiseks avaliku plokiahela Ethereumi, millel on puudused (nt lubadeta, madal läbilaskevõime ja andmete privaatsus). Arvestades avalike plokiahelate puudusi, pakub käesolev lõputöö lahenduse integreerida Caterpillarisse lubatud plokiahela platvorm. Kavandatud lahendus kasutab Hyperledger Fabricut, mis pakub liikmelisusel põhinevat plokiahelavõrku, mille tulemuseks on vähem node kui Ethereumis. Seega võimaldab see organisatsioonidel sooritada kiiremaid tehinguid, saavutada suurem andmete privaatsus ja kohandada oma võrku vastavalt nende vajadustele.

Võtmesõnad:

Äriprotsessid, Äriprotsesside juhtimissüsteem, Plokiahela tehnoloogia, Solidity nutikas leping, Hyperledger Fabric

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimis-teooria)

List of Figures

Figure 1: Blockchain: sequence of blocks with transactions [8]......12
Figure 2: Decentralized peer-to-peer blockchain network......13
Figure 3: Public vs Private blockchain networks [33]......15
Figure 4: The Architecture of Caterpillar [8]......22
Figure 5: Caterpillar's compilation process [8]......23
Figure 6: Thesis work architecture......24
Figure 7: Hyperledger Fabric transaction workflow[55]......28
Figure 8: Burrow EVM integration with Fabric [57]......29
Figure 9: Example Peer Chaincode command......32
Figure 10: Hyperledger Fabric adapter......33
Figure 11: Hyperledger Fabric First Network......39
Figure 12: Crypto Configuration......39
Figure 13: *script.sh* file......40
Figure 14: Run of docker containers......41
Figure 15: Organisation setup and channel joins......41
Figure 16: Running docker containers......42
Figure 17: EVM Chaincode mounting......42
Figure 18: SSH into peer-cli......43
Figure 19: Install-evm script......43
Figure 20: Utils script modifications......43
Figure 21: Install EVMCC on peers result......44
Figure 22: Running Fab3 proxy script......45
Figure 23: Fab3 service for User1......45
Figure 24: Unit tests output......55
Figure 25: Invoice Handling BPMN model......56
Figure 26: Role-based binding policy......57
Figure 27: Deployments integration tests output......58
Figure 28: Process Case Request Output......59
Figure 29: UNBOUND Role state......59
Figure 30: BOUND Role state......59
Figure 31: Nominate Example Request......60

List of Tables

Table 1: Caterpillar’s Runtime Registry REST API (extends from [8]) ... 35
Table 2: Caterpillar's Compilation-based REST API (extends from [8]) ... 35
Table 3: Caterpillar’s Interpreter-based REST API (extends from [8]) ... 36
Table 4: Caterpillar’s Dynamic Control REST API (extends from [8]) ... 36

List of Source Code

Source Code 1: Set Web3/Fab3 Provider ... 46
Source Code 2: Get Web3/Fab3 Provider ... 46
Source Code 3: Deploy Smart Contract Function ... 47
Source Code 4: Execute Smart Contract Function ... 48
Source Code 5: Call Smart Contract Function ... 49
Source Code 6: Get Event information from Hyperledger logs ... 50
Source Code 7: Get Transaction Info ... 51
Source Code 8: Validate Fab3 URL Middleware ... 52
Source Code 9: Example Middleware Usage ... 52
Source Code 10: Set Web3 Provider in Middleware ... 52
Source Code 11: Caterpillar Example Common Controller ... 53
Source Code 12: Caterpillar Handling Ethereum Adapter ... 53
Source Code 13: Caterpillar Handling Hyperledger Adapter ... 53

Table of Contents

1 Introduction 8
1.1 Problem Statement 8
1.2 Thesis Contributions 9
1.3 Thesis Outline 10
2 Background 11
2.1 Business Processes 11
2.1.1 Collaborative Business Processes 11
2.1.2 Business Process Management System 11
2.2 Blockchain Technology 12
2.2.1 Bitcoin 13
2.2.2 Smart Contracts 13
2.2.3 Consensus Mechanisms 14
2.3 Types of Blockchain 15
2.3.1 Permissionless Blockchain 16
2.3.2 Permissioned Blockchain 16
2.4 Ethereum Blockchain 16
2.4.1 Ethereum Virtual Machine 17
2.4.2 Development Tools 17
2.5 Hyperledger Blockchain 18
2.5.1 Hyperledger Fabric 18
2.6 Caterpillar 18
3 Related Work 20
3.1 Blockchain-based Collaborative Business Process: Implementation and Execution 20
3.2 Solidity-based smart contract execution on Hyperledger 21
4 Architectural Design 22
4.1 Caterpillar Approach 22
4.2 Our Approach 24
4.3 On-Chain Runtime: Hyperledger Fabric 25
4.3.1 Hyperledger Fabric Architecture 26
4.3.2 Solidity Smart Contracts on Hyperledger Fabric 28
5 Solution: Hyperledger Adapter 31
5.1 Hyperledger Adapter 31
5.1.1 Fab3 Proxy Integration 32
5.1.2 Handling multiple Fab3 Instances 33
5.2 Hyperledger Adapter Integration 33

5.2.1 Extended REST API Overview 35
6 Implementation And Evaluation 38
6.1 Technologies Used 38
6.2 Hyperledger Fabric Network Setup 38
6.2.1 Running the network 40
6.3 Configure EVM Chaincode 42
6.4 Setup Fab3 Proxy 44
6.5 Hyperledger Adapter Implementation 45
6.5.1 Hyperledger Adapter Implementation 46
6.5.2 Fab3 Middleware Implementation 51
6.5.3 Hyperledger Adapter Integration 53
6.6 Evaluation 54
6.6.1 Evaluation Setup 54
6.6.2 Unit Testing 54
6.6.3 Integration Testing 55
6.7 Discussion 60
6.7.1 Limitations 60
7 Conclusion and Future work 62
7.1 Future Work 62
8 References 63
Appendix 67
I. Abbreviations 67
II. Repository 68
III. License 69

1 Introduction

Business Processes are a core pillar of organisations. They are defined as a collection of tasks or activities performed by stakeholders to achieve business goals. Business processes integrate systems, tasks, and data in a structured course to streamline organisational activities, providing customers with products and services [1]. While business processes typically focus on only one organisation, collaborative business processes span multiple organisations to cooperate with other businesses. The information systems that coordinate human interactions with system operations and enable organisations to dynamically execute and manage their business processes are called Business Process Management Systems (BPMSs) [2].

In the 21st century, consumer demand has grown drastically because of globalisation and advancement in information technology (IT), which put immense pressure on organisations to stay competitive in the market. Therefore, collaborative business processes between organisations are becoming increasingly important. However, lack of trust between organisations is a huge problem, which is causing a great hindrance in implementing and executing collaborative business processes. To resolve this issue, companies usually rely on third parties (i.e., governments, lawyers, etc.), which serve as mediators in case of conflicts [3].

Recently, Blockchain technology has emerged as a solution to the lack of trust problem, enabling mutually untrusted parties to collaborate in a decentralised manner without the need for third authorities. Bitcoin is claimed to be the first digital currency that used distributed ledger to solve payment issues among untrusting entities because the proof of payment is cryptography rather than trust in the central bank [4]. Nowadays, Blockchain is finding applications in diverse fields such as the internet of things (IoT), internet interaction systems, smart contracts, and security systems [5].

Blockchain technology also provides solutions for untrusted organisations to implement and execute collaborative business processes without the need for third-party mediators. [6]. Specifically, blockchain systems allow organisations to deploy their business processes as a program (called a smart contract) on an immutable distributive ledger, which also helps them execute functions and implement transactions on top of the ledger. For example, Ethereum is one such blockchain platform which enables stakeholders to deploy smart contracts with the help of Solidity (the language to write smart contracts on Ethereum) [7].

Caterpillar is an existing tool that has been taken as a starting point for this thesis. This tool is a BPMS prototype that utilizes the characteristics of Blockchain mentioned above, i.e., tamper-proof and decentralised ledger, smart contracts, etcetera. Caterpillar enables organisations to execute collaborative business processes on the Ethereum blockchain by generating a set of smart contracts in Ethereum's Solidity language from the business process model [8]. However, Ethereum is a public permissionless blockchain with many drawbacks [9]. Next, Section 1.1 will explain the disadvantages of the public blockchain and how the private blockchain is solving them.

1.1 Problem Statement

Caterpillar is a BPMS prototype that implements and executes business processes using smart contracts on the Ethereum network, a public and permissionless blockchain. While Ethereum-based BPMS possesses many benefits over traditional BPMS, considerable drawbacks exist.

Permissionless blockchain includes public blockchains where anyone can join the network as a node in a decentralised manner because there is no centralised authority managing the membership. This behaviour increases the network's security and makes it virtually impossible to corrupt the data because every transaction needs to be validated by all the nodes. However, this means organisations' transactions will be shared publicly, losing their data privacy. Usually, organisations don't want to share their business processes because they want to maintain their competitive advantage. Additionally, due to the size of the public network (i.e., Ethereum has approximate 6000 nodes[1]), transactions take longer to get validated because the network requires consensus among the nodes in the network [10, 11]. Proof of work is the most common consensus mechanism, which takes time and introduces some latency due to the time the nodes require to complete and verify the proof of work. So, with low throughput, operations can perform slowly and hamper organisations' business processes.

Contrary to public blockchains, private blockchains, often termed permissioned or consortium blockchains, have a mechanism where the central authority decides who can be a network member. Additionally, the users can only perform specific actions granted to them by the ledger administrators. This characteristic enables organisations to control their network and allow only agreed entities to join it. It also substantially increases data privacy because only known actors participate in the network. Moreover, these blockchains have fewer nodes, thus considered more efficient and provide faster transaction speed. However, permissioned blockchains lack decentralisation and can be tampered with and lose integrity if most of the organisations inside the consortium agree to it [10, 11].

Caterpillar has a Business Process Modelling Notation (BPMN) compiler that translates the business process model into Solidity smart contracts [8]. The goal of the thesis is to execute these smart contracts on a permissioned blockchain. There are many permissioned blockchain platforms (i.e., Quorum [12], Hyperledger Fabric [13], R3 Corda [14] etc.), but not all execute Solidity smart contracts. In this thesis, we will explore the capability of EVM Chaincode, which adds Solidity smart contracts support to Hyperledger Fabric [15].

1.2 Thesis Contributions

This thesis will extend the Caterpillar BPMS prototype by adding Hyperledger blockchain support to allow users to execute Solidity smart contracts generated from the BPMN model on permissioned blockchain. Below we describe how this thesis contributes to the Caterpillar project:

a) **Integrate EVM Chaincode with Hyperledger Fabric** – Hyperledger Fabric out of the box doesn't support Solidity-based smart contract execution. However, with the integration of EVM Chaincode, Hyperledger Fabric will be able to deploy and run Solidity smart contracts. This integration will enable users and organisations to deploy their Ethereum applications on the Hyperledger Fabric network and utilise the benefits of permissioned blockchain. Additionally, the Fab3 proxy service setup will provide support for JSON-RPC API, which will let us use Ethereum's Web3.js SDK to invoke blockchain operations on the Fabric network easily.

[1] https://www.ethernodes.org/

b) **Develop Hyperledger Adapter for Caterpillar** – Caterpillar has developed an Ethereum adapter, which creates a bridge between its core services and the Ethereum blockchain. Similarly, Hyperledger adapter development will provide an interface to invoke operations on the Hyperledger Fabric network. This adapter consists of deploying a smart contract, executing smart contract functions, monitoring the blockchain network, etc. These operations will let Caterpillar perform its BPMS tasks efficiently on the Hyperledger blockchain.

c) **Integration of developed Hyperledger Adapter** – The developed adapter for Hyperledger Fabric will support integration with existing Caterpillar engines such as Runtime Registry, Compilation Engine, Interpretation Engine, and Dynamic Access Control. For example, suppose a user wants to deploy a BPMN model. In that case, Caterpillar will be able to compile the provided BPMN model into Solidity smart contract and deploy it on the Hyperledger Fabric network using the developed adapter.

1.3 Thesis Outline

This thesis is structured as follows: Chapter 2 introduces business processes, BPMS and blockchain concepts and discusses different types of blockchains and some permissionless and permissioned blockchain platforms. This section also discusses Caterpillar’s ability to run business processes on the blockchain. Chapter 3 reviews the existing tools and methods related to this thesis. Chapter 4 discusses the architectural overview of Caterpillar, our approach, and the detailed architecture of the Hyperledger Fabric network. Next, Chapter 5 discusses our solution and how the Hyperledger adapter integrates with the Caterpillar components. Chapter 6 provides the implementation and evaluation of the proposed work. Finally, Chapter 7 concludes the thesis by summarizing the contribution and providing the future work directions.

2 Background

This chapter discusses relevant background on business processes, blockchain technology, smart contract, and types of blockchains.

2.1 Business Processes

Business processes are defined as a set of activities performed within the organisation to achieve some goals by providing the product or service to the customer. Each participant in the process is assigned some tasks to act on in a given period. It becomes the building block of the process as well [1]. The business process converts individual efforts into teamwork and helps an organisation attain specific aims. An organisation should work on the business process in a well-defined fashion because these are necessary for the organisation to identify the tasks they should perform to make the company more efficient. It will also help to streamline communication between workers of the organisation. Moreover, it will make the business organisation more focused on the goals. The business processes should be managed with care to get the desired results from the organisation [16].

2.1.1 Collaborative Business Processes

Globalisation and commercialisation in the 21st century have changed the working mechanism of different business organisations, and they are becoming dependent on business processes day by day. By Gartner Inc., Business Process Management (BPM) was ranked in 3rd position among the top 10 strategic technologies for 2008 [17].

This upgrading strategy has attracted different organisations to collaborate on executing various business processes together dynamically. Unlike an organisation's business process, a collaborative business process has developed harmony among various business organisations to gather knowledge and complementary competencies to attain more success in the business. Some prominent examples of collaborative business processes are enterprise information systems based on PAIS (Process-Aware Information Systems), medical business processes and e-commerce business processes [18].

Tough collaborative business processes are providing the organisations achieving their goals; there are some challenges, such as trust issues among cooperative parties [19]. For instance, when a partner organisation performs a task, which partners would have access to data of that particular task? Moreover, the business processes of different organisations should be adaptable to constantly changing business environments to remain competitive in the global market [17].

Business processes should be flexible to work in dynamic scenarios to permit modifications in real-time. For example, some collaborative processes need dynamic binding and re-binding in the logistics field. Notably, in a buyer-supplier-carrier process, the supplier can appoint a carrier, and sometimes a buyer selects the carrier. In some cases, the supplier has a right to alter the carrier even after an initial selection, e.g., the carrier is not able to grab the parcel on time. In a nutshell, we can say that the duty for a particular task can change across different scenarios [8].

2.1.2 Business Process Management System

Business process management consists of three components: principles, methods, and tools that are used to analyse, execute, and monitor different business processes of the organisations [1]. Especially, BPM leads stakeholders on how they can coordinate and collaborate to attain the organisation's goals during the life cycle of the whole process that

is comprised of identification, discovery, analysis, redesign, implementation, execution, monitoring, and adaptation. Organisations excerpt high-level data and relations related to a specific problem under the given scope of a process during the identification stage. The next stage is the discovery phase, where data analysts perform functions to elaborate the process in graphical representations termed “as-in” model. The analysis phase comprises identification issues to improve the process for further steps [8].

As a result, the "as-in" model is transformed into a "to-be" model that may be implemented and processed in the following redesign step. The “to-be” model is transformed into software during the implementation step to execute the process. After the implementation phase, the creation and handling of the specific process are involved in the execution step, which is called process cases. The process execution is performed by BPMSs, which also helps in the monitoring [8]. BPMSs were not efficient due to dependency upon third parties to execute them. However, BPM can be performed using decentralised technology, named blockchain, that bypasses the need of any third parties to run the business process on a secure and efficient medium.

2.2 Blockchain Technology

Blockchain is a tamper-proof, immutable, and decentralised ledger [8]. It can be accessed anywhere globally because the ledger is distributed to all the peer nodes without trust issues. The ledger consists of a specific sequence of blocks linked together, and it contains an orderly set of transactions, as represented in Figure 1. Each block has a particular hash value that connects a block to its previous block. If someone wants to change or remove a transaction, the only process is to recreate the entire chain. Additionally, some peer nodes are miners, which are accountable for validating and grouping the transactions processed by the users at the end of the chain. As there is no central authority, a distributed decentralised mechanism is the only way for miners to reach a consensus [20] [21].

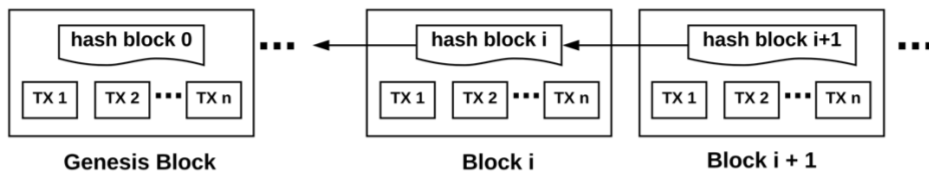

Figure 1: Blockchain: sequence of blocks with transactions [8]

All the clients use a concrete network of the blockchain (peer-to-peer network, as shown in Figure 2) for submitting the transaction and reading the data from it. All the submitted transactions are assembled in the form of blocks, and then these blocks are broadcasted across the network at the end of the blockchain. A creator must form the transaction adequately and sign it properly so that it can be validated on the network. The transactions are not affected by trust issues among nodes because these are cryptographically signed, validated, and broadcast widely across the network. A consensus mechanism makes it secure and tamper-proof without considering mutual trust among the actors [22].

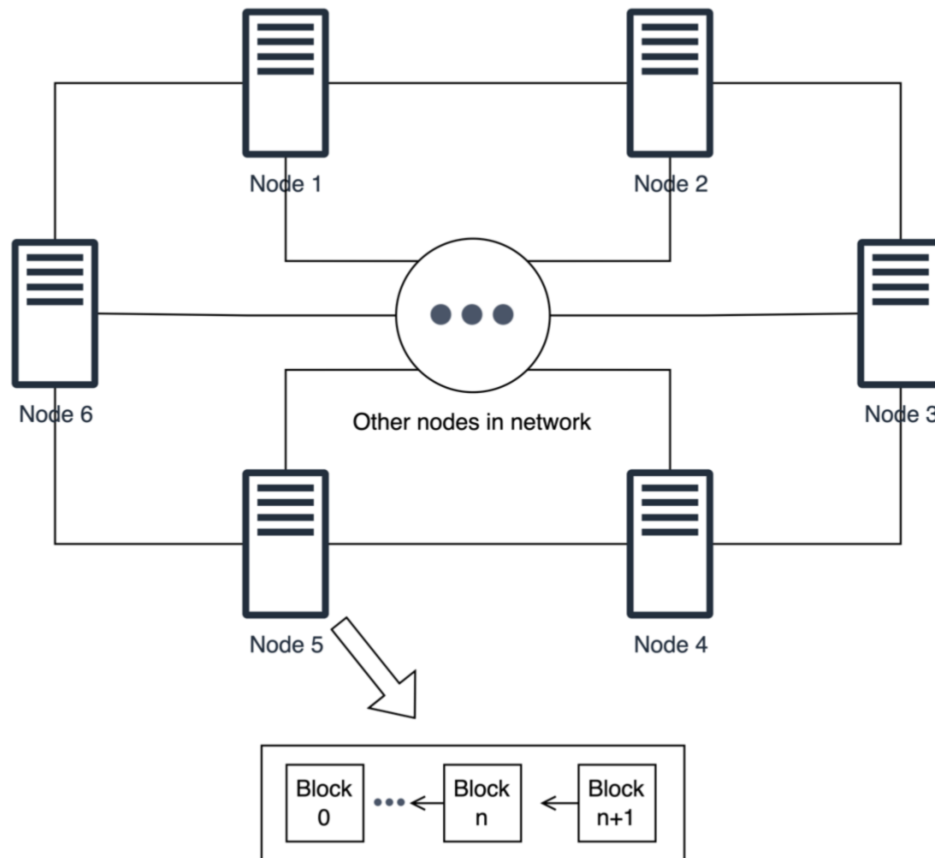

Figure 2: Decentralized peer-to-peer blockchain network

One of the typical applications of blockchain is a cryptocurrency, a digital asset, such as bitcoin. However, the blockchain is not limited to only cryptocurrencies. Blockchain is finding applications in diverse fields such as the internet of things (IoT), internet interaction systems, smart contracts, and security systems. The blockchain can be a good storage and verification technology for legal documents, and it has been named “World Wide Ledger” as well [5].

2.2.1 Bitcoin

Bitcoin is claimed as the first digital cryptocurrency established in 2008-09 when Satoshi Nakamoto published the whitepaper “Bitcoin: A peer-to-peer Electronic Cash System” [4]. It was the first decentralised currency that used distributed ledger to store the data of transactions on the blockchain. Bitcoin was one of the solutions which could solve payment issues between untrusting parties because the proof of payment was cryptography rather than trust in the central bank. One of the significant limitations of bitcoin is scalability, as the largest size of one block is restricted to 1MB, and one block is being mined every ten minutes. Bitcoin cannot be used for high-frequency trading due to its slow transaction speed, which is mere seven transactions per second. With time, the size of the block will increase, which will further slow down the throughput. However, alternative off-chain solutions like lighting networks are available, which could solve the bitcoin scalability problems and ensure instant transactions [23].

2.2.2 Smart Contracts

A smart contract is a set of coded protocols that runs on a blockchain. It executes only when the information saved in the contract is verified. Smart contracts work on an “if-then” model

[24]. For example, if a specific condition is satisfied, then the information stored on the smart contract will be executed automatically. These smart contracts carry information that can be accessed and verified publicly.

The whole process of the smart contract can be divided into four distinct phases: Creation, deployment, execution, and completion. At first, all the organisations involved in the contract negotiations discuss the contract's rights, clauses, and prohibitions. When they agree upon some rules, with the help of lawyers, a traditional contract is written in natural language (i.e., English). Then a team of software developers will convert this natural language into a script using a programming language such as Solidity. This process is iterative and requires many rounds of discussions among collaborative parties, lawyers and software developers. [25].

The smart contracts are deployed to the top of the blockchain after validation. Smart contracts cannot be modified once deployed due to the immutable nature of blockchain. A new smart contract has to be deployed when an amendment is required on a contract. Smart contracts can be accessed by the parties once deployed, and the cryptocurrency needed for the smart contract is frozen in their wallets. Identification of the parties involved in the smart contract is processed through their digital wallets [26].

After deployment, the next step is execution, and then contractual clauses are evaluated. Once the contractual provisions are validated, the processes conditioned with them are executed automatically. Logical connections connect the information on blocks, so once information is validated, it will run the further information and consequently, miners of the blockchain will validate this information. Once a transaction is completed, the state of the block is updated, and it is stored on the blockchain [27].

The last step of the smart contract life cycle is the completion of the smart contract. During this step, the states of all the involved organisations are updated to the new forms, and digital assets are transferred from one organisation to another; then, all the digital wallets of the organisations are now unlocked. With the completion of this step, one life cycle of the smart contract is successful [28].

2.2.3 Consensus Mechanisms

"A consensus mechanism is a fault-tolerant mechanism used in computer and blockchain systems to achieve the necessary agreement on a single data value or a single state of the network among distributed processes or multi-agent systems, such as with cryptocurrencies" [29].

There are two main consensus mechanisms used in the blockchains are discussed below:

a) Proof-of-Work

The proof-of-work consensus mechanism is used by public blockchains such as Bitcoin and Ethereum [30]. In this, the addition of a new block is known as mining, where a mighty computational power is required for the computationally difficult cryptographic puzzle. Every node tries to solve the puzzle before any other node so that it will add a new block to the chain and will get a financial reward in return. Each new block has a unique hash value of the previous block. If someone tries to alter the data, it will require powerful computing servers and high computational costs to preserve all the cryptographic hashes and their recreation concerning the altered block. So, it looks virtually impossible to tamper with such a concrete network. Miners perform highly challenging puzzles that require a lot of

computational energy and power consumption, which are drawbacks of proof-of-work consensus.

b) Proof-of-Stake

Proof-of-stake is an energy-saving consensus mechanism, and some famous blockchains, such as Ethereum, are trying to adopt it in place of proof-of-work consensus [31]. In this mechanism, minors execute a process that selects a minor with more stake in the currency. This consensus encourages the scheme that miners with more stakes will have less probability of tampering with the data. This process can make blockchain more vulnerable because computing cost during the mining process is almost zero [11].

2.3 Types of Blockchain

Blockchains are now widely used in different areas of life, from cryptocurrencies to other decentralised solutions. Some blockchains are available publicly, and some blockchains, such as blockchain networks for business management within collaborative organisations, are so-called permissioned where these ledgers cannot be accessed publicly [32]. We can divide all the blockchains into three categories:

- **Public blockchain** where anyone is allowed to join the blockchain network. Additionally, everyone has permission to read and write the transaction. Public blockchains are also referred to as permissionless blockchains.
- **Consortium blockchain** where a set of people is allowed to join the blockchain, for instance, multiple collaborative organisations, but data of the transactions can be public or private depending upon the blockchain network.
- **Private blockchains** where a single party plays the role of central authority and decides the nodes which can join the network. Data of the transactions remain private as well [32].

As discussed above, the blockchain network is mainly divided between public and private blockchain networks. Figure 3 shows an overview of the main differences between the two types of networks.

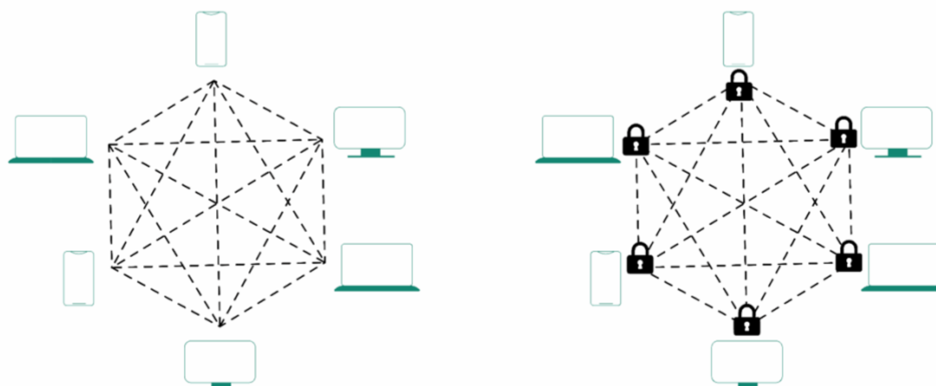

Figure 3: Public vs Private blockchain networks [33]

2.3.1 Permissionless Blockchain

Permissionless blockchain includes public blockchains where any peer/node can join the network in a decentralised manner because there is no central authority managing the membership [8]. Additionally, every peer can proceed with a transaction, and the transaction requires approval from every node of the network, making these blockchains tamper-proof. Alteration of data on these blockchains remains nearly impossible. Characteristics like data integrity, immutability, transparency, and public verifiability make public blockchains an excellent choice for untrusted actors to perform operations in a decentralised environment. In contrast, these blockchains possess some performance drawbacks, such as the limited number of transactions per second and high latency. Bitcoin [4] and Ethereum [34] are prominent examples of public blockchains.

2.3.2 Permissioned Blockchain

Consortium and private blockchains are also known as "permissioned" because a central authority determines memberships and permissions for users to join and perform actions within the network [8]. The users can only perform specific actions granted to them by the ledger administrators and are required to identify themselves through certificates or other digital means. These blockchains have fewer nodes, thus considered more efficient and provide higher throughput than permissionless blockchains. Additionally, permissioned blockchains offer more privacy and reduce redundancy due to lesser data replication. However, these blockchains may be subject to trust issues and data loss in the event of disagreement, given that they lack decentralisation. For instance, the network can be compromised and lose integrity if the majority of the organisations within the consortium network agree to it [10, 11]. The most relevant permissioned blockchain platforms are Hyperledger Fabric [13], Quorum [12], and R3 Corda [14].

2.4 Ethereum Blockchain

Ethereum blockchain is a decentralised technology used to run different smart contracts using cryptocurrency. Ethereum smart contracts are written in Solidity programming language deployed on Ethereum virtual machine (EVM) [7]. EVM is a virtual computer that is enabled to execute smart contracts and add blocks to the Ethereum blockchain. A smart contract is usually written in Solidity, which is converted into bytecode by a compiler. EVM then executes this bytecode, and this execution requires computational power depending upon the size of the smart contract [35]. The computational cost to perform the transactions and execute the smart contracts on the Ethereum blockchain is termed gas, and it is an essential factor of the Ethereum blockchain. The amount of gas required to execute the business process correlates to the smart contract's bytecode size [36].

Ethereum possesses two types of accounts protected by different types of keys. Externally owned accounts (EOA) are protected by private keys, and Contract accounts are protected by the contract code. There are four key components of each Ethereum account; nonce, ether balance, contract code hash, and storage root [37].

a) **Nonce** represents the number of transactions generated or the number of contracts created from a specific address. It is the guarantee of transactions because a transaction can proceed only once.
b) **Ether balance** is the amount of Wei present in an account (1 Ether=1018 Wei) used during the transaction [37]. Wei is considered a minor fraction of Ether.
c) **Contract code hash** is the specific hash of EVM related to the particular account.

d) **Storage** can be explained as a 256-bits hash that belongs to the root node of a tree that signifies the account's content [38].

2.4.1 Ethereum Virtual Machine

Ethereum virtual machine (EVM) is a virtual computer that is executed by the Ethereum protocol in a distributed manner [39]. So, the EVM is the component of the system that distributes messages based on events. EVM supports stack-based computing power whenever a small change occurs in the system. Each EVM word requires a space of 256 bits, and smart contracts are allocated space on the system known as "storage". The memory is mapped in a 256 bits-to-256-bits manner. This persistent memory of a smart contract is private so that everyone cannot access it. The EVM uses stack memory that can compensate 1024 words in it, and the top 16 words can be accessed during execution. Each smart contract writes data in a log fashion accessible to external applications. Every peer in the Ethereum network develops communication with the other with the help of the Ethereum wire protocol. Caterpillar uses the Ethereum blockchain for the execution of business process management. Ethereum blockchain consists of many blocks that contain information, and this process is performed using Ethereum Virtual Machine (EVM) [8].

Many programming languages are used to convert natural language into a script. After that, many compilers are present that can convert this script into bytecode for the EVM [8]. Solidity is one of the languages that can perform this function, and it is widely used. In Solidity, the procedures are written in Java-like script. So, a contract can be defined in the form of the contract's state and operations to query and manipulate the properties. It is a high-level Turning-complete programming language which is typed statically and supports inheritance and polymorphism. Contract codes written in Solidity are comprised of functions and variables which are used for reading and modifying these, likewise in traditional imperative programming [7].

2.4.2 Development Tools

As Ethereum is the most popular platform for creating and managing smart contracts and dApps, the developer experience is vital for massive adoption. Some of the famous developer tools are briefly discussed below:

- **Ganache** is a blockchain for Ethereum and Corda dApps development. Developers can use it for the entire developmental cycle, such as developing, deploying, and testing the dApps in a secure and deterministic environment [40].
- **Truffle** is the best developmental tool if testing dApps is the primary purpose. Its main focus is the testing of smart contracts. JavaScript is used for writing the test and can communicate with both test and main networks [41].
- **Remix** tool is a valuable developer tool because it is an all-in-one browser-based tool for testing, deploying, and managing smart contracts [42].
- **Web3.js** is a JavaScript-based API that can be used to develop clients that connect with Ethereum. Developers can use Web3.js libraries to perform different actions, such as sending Ether from one account to another. It also functions to create smart contracts and read and write the data from smart contracts [43].

2.5 Hyperledger Blockchain

Hyperledger is a framework developed under the Linux Foundation. It is an enterprise framework that ensures secured, immutable, and faster transactions on the blockchain [44]. Hyperledger is not a blockchain itself but provides a framework for different pre-existing networks to run on the blockchain. Unlike other blockchains, it does not require any cryptocurrency or computing powers. An authority regulates all the transactions in the Hyperledger, and only the network peers can read and write the transactions. In this way, the data stored on Hyperledger remain secure and protected for collaborative organisations.

Though Hyperledger is a different blockchain from Ethereum, still, it supports the smart contracts of Ethereum [15]. This network was designed for enterprises and business process management where collaborative parties do not want to share their data with everyone on the blockchain [14]. Though the execution network requires all the nodes to validate the transaction, peers always need permission from a central authority to get into the network. This permissioned blockchain is a great value addition to the blockchain, where many business models can be executed without the dependency upon the third parties. Many collaborative business processes are performed between untrusting parties. They can also use this Hyperledger network with an immutable, secure, and faster blockchain network [45]. Hyperledger is working on many projects to introduce new technical ways to make these things more applicable. These projects are Hyperledger Sawtooth, Hyperledger Iroha, Hyperledger Burrow, Hyperledger Fabric, Hyperledger Composer, Hyperledger Explorer, and Hyperledger Indy [44, 46]. In our thesis, we focus on how we can execute Ethereum-based smart contracts on Hyperledger Fabric.

2.5.1 Hyperledger Fabric

Hyperledger Fabric is one of the most popular network among all Hyperledger projects [44 , 46], and our executions of smart contracts of Ethereum are supported on Hyperledger Fabric. It is very efficient and can perform 3000 transactions per second [47], whereas Ethereum can only perform between 1-20 transactions per second [48].

Hyperledger Fabric is specifically designed for enterprises and thus used by many business organisations for their private collaborative processes. Additionally, participants are unable to perform their actions until given authorisation by the central authority managing the blockchain network. Hyperledger Fabric is flexible and secure at the same time, as compared to other Hyperledger projects [44]. Besides, in recent years, IBM technical ambassadors developed the EVM Chaincode that adds the capabilities to deploy and execute bytecode of Solidity smart contract over the Hyperledger Fabric network [15].

2.6 Caterpillar

Caterpillar is software for business process management that enables the collaborative parties to perform transactions on the blockchain. This software is available in the public domain, but peers need approval from the cooperative parties to read and write on the blockchain. This software has brought the process-centric mechanism (process discovery, implementation, and maintenance) to the blockchain. It works on compliance by design model so that a peer cannot perform a transaction if it is not abiding by the collaborative process model [19].

Caterpillar captures the input in the form of a business process model and notation (BPMN), which will generate different smart contracts in the form of a set that will apprehend the underlying behaviour [8]. At first, the smart contracts scripted in Solidity programming

language are compiled, and then these can be run on any permissioned (i.e. Hyperledger) or permissionless (i.e. Ethereum) blockchain networks. However, in its current form, Caterpillar only supports the execution of smart contracts on Ethereum, a permissionless blockchain.

In this thesis work, we focussed on Hyperledger (specifically Hyperledger Fabric) to deploy and execute smart contracts compiled by Caterpillar for permissioned blockchain cases. For that reason, we decided to exploit the capabilities of EVM Chaincode to implement the thesis work solution.

3 Related Work

This Chapter discusses the existing studies and tools related to the domain of this thesis.

3.1 Blockchain-based Collaborative Business Process: Implementation and Execution

This section analyses the existing solutions for implementing and executing collaborative business processes on blockchain technology. Further, this section will also briefly examine why Caterpillar's approach is different.

Weber et al. [49] used the intrinsic characteristics of blockchain to address the trust issue in collaborative business processes. They implemented a solution to manage business processes using the Ethereum blockchain platform as an underline decentralised technology. In their approach, the authors propose to compile the BPMN model into a Solidity smart contract, which records the exchange of messages among parties correlated with the ordering relationship of the BPMN model. Moreover, they also presented a technique to connect the Off-chain environment with the business process execution on the blockchain using triggers and interfaces.

In another approach, Prybila et al. [50] proposed using the Bitcoin blockchain to monitor the execution of business processing using specialised tokens. Similar to the method proposed by Weber et al. [49], the authors assumed that the collaborative business execution among participants is done by the exchange of messages following the relation of the BPMN model. Thus, blockchain is only used to record the exchange message to check the status of business processes.

In another study, Garcia-Banuelos et al. [51] suggest an approach to translating the BPMN model into optimised Solidity smart contracts. While Weber et al. [49] and Prybila et al. [50] assume that interaction of business processes between parties occurs using an exchange of messages, the authors of this study consider that the parties use blockchain to collaborate and preserve the current process's state and identify the succeeding tasks or events. This work, however, does not entirely cover the BPMN process paradigm and supports only tasks and necessary gateways (AND and XOR gateways).

Pourheidari et al. [52] presented a case study investigating the applicability of implementing and executing business processes on the permissioned blockchain among untrusted parties. They used the Hyperledger Fabric blockchain platform to determine the benefits of permissioned blockchain by implementing a business process of Order Processing.

Based on the above-discussed approaches, apart from [51], all the existing solutions use blockchain to record the exchange of messages for executing collaborative business processes. This means that organisations still carry out most of their business processes on BPMSs, and they utilise blockchain technology only to record the exchange of messages to manage process interactions. However, [51] suggests a solution for the above problem, but it only focuses on tasks and necessary gateways (AND and XOR gateways) of the BPMN model. Meanwhile, Caterpillar, a BPMS prototype, provides broad support for BPMN elements, including hierarchical process models (i.e. subprocesses). Moreover, Caterpillar facilitates the implementation and execution of collaborative business processes on Ethereum, with all significant BPMS components hosted on the blockchain [8]. In this thesis, we extend the Caterpillar BPMS to integrate permissioned blockchain to address the problems of the public blockchain (i.e., low throughput, privacy problems, lack of control over the network, etcetera).

3.2 Solidity-based smart contract execution on Hyperledger

Caterpillar is a BPMS that translates the BPMN model to Solidity smart contracts to implement and execute collaborative business processes on the Ethereum blockchain. Solidity is an exclusive language for implementing smart contracts on the Ethereum blockchain. This section explores existing approaches for executing Solidity smart contracts on Hyperledger.

Stefano Franzoni [33] created a blockchain-based solution for supply chain management for a fashion company. For this purpose, the author used cross-chain interaction between public and permissioned blockchains. While Hyperleder Fabric's consortium network handles the internal supply chain processes (i.e., orders, production, etcetera.) among vendors, the end-user sale process is dealt with by Ethereum public network. Solidity smart contacts handle the interoperability between both networks. The author used EVM Chaincode to execute smart solidity contracts on Hyperledger Fabric.

Swathi et al. [53] proposed a solution to resolve the issue of interoperability between permissioned and permissionless blockchains. They discussed the possibility of integrating permissioned Hyperledger Fabric with EVM Chaincode to provide the interaction with the Ethereum public network. Moreover, the authors of this study proposed that Hyperledger Fabric's functionality to support Solidity can provide an easier way to handle the interaction between different EVM compatible blockchains.

We observe that, existing approaches mainly focus on solving the interoperability problem between Hyperledger Fabric and Ethereum. Moreover, they indicate the possibility of executing Solidity smart contracts on Hyperledger. However, this thesis is an extension of the Caterpillar BPMS prototype; thus, the approaches mentioned above are outside our scope.

4 Architectural Design

This thesis chapter will discuss the current Caterpillar approach and compares it with our modified approach. Moreover, this chapter also explores the Hyperledger Fabric blockchain and where it falls in our approach.

4.1 Caterpillar Approach

Caterpillar is a BPMS prototype that currently runs on top of the Ethereum blockchain and compiles process models specified in BPMN into smart contracts that translate the underline behaviours. To explain the working of Caterpillar in detail, its architecture will be discussed, which is organised into three layers, as shown in Figure 4.

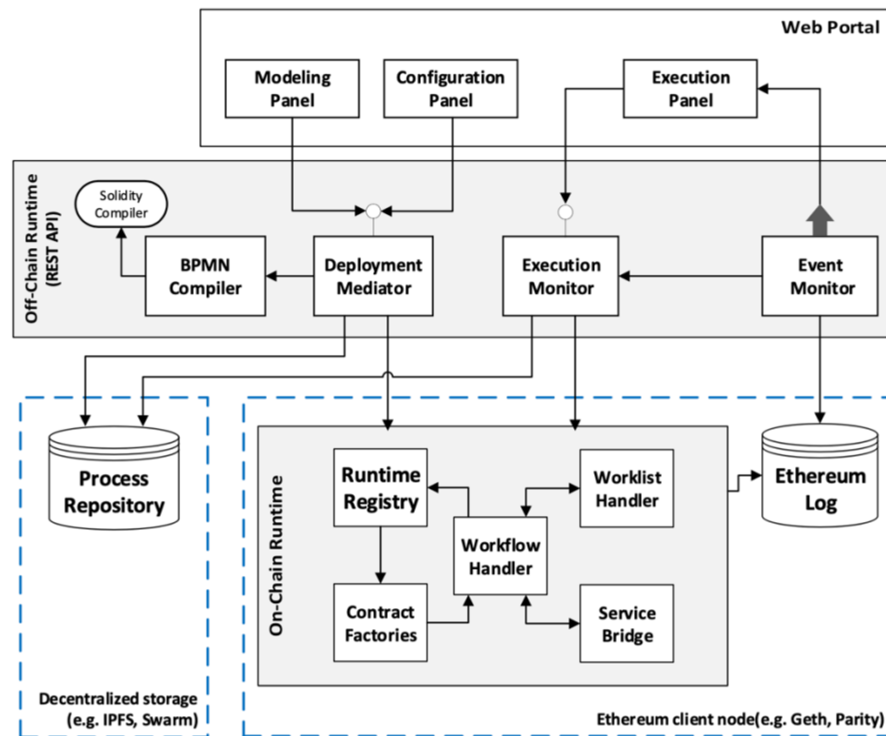

Figure 4: The Architecture of Caterpillar [8]

The bottom layer, which is termed "On-chain Runtime", is responsible for smart contracts used for managing support code, e.g., process instantiation, and process-specific code, e.g., control-flow, process data, etc. The On-chain modules are deployed on all the public nodes of the Ethereum, a public blockchain network. "Ethereum Logs" are responsible for monitoring the events for transactions and smart contracts executions logs as well as the data related to them. "Process Repository" is available to store compiled data like bytecode for easy access [8].

The layer in the middle is referred to as "Off-chain Runtime". It is a REST-based API which provides an interface to access the functionalities of Caterpillar. It includes tools to compile, deploy, execute, and monitor business processes on Caterpillar's On-chain Runtime in the form of smart contracts. For instance, the BPMN compiler, Deployment Mediator, Execution Monitor, and Event Monitor are different modules which are part of the Off-chain Runtime.

Finally, the top layer act like a client layer (web portal), which gives the end-user the functionality to provide the necessary configurations and instructions to execute and monitor business processes.

In a nutshell, Caterpillar's main feature is to take the BPMN model, compile it into Solidity smart contracts and then execute these smart contracts on the Ethereum blockchain network using an Ethereum adapter. These generated smart contracts preserve the BPMN model's behaviours and help run the business processes on the network.

The BPMN compiler, which is part of the Off-chain layer, is responsible for generating smart contracts from the process model [8]. The compilation process consists of two steps (refer to Figure 5).

1) In the first step, the BPMN compiler takes the BPMN model, parses it and then translates it into Solidity smart contracts (indicated as **BPMNSol** in Figure 5). Moreover, it includes additional metadata required to execute and monitor business processes on the blockchain network.

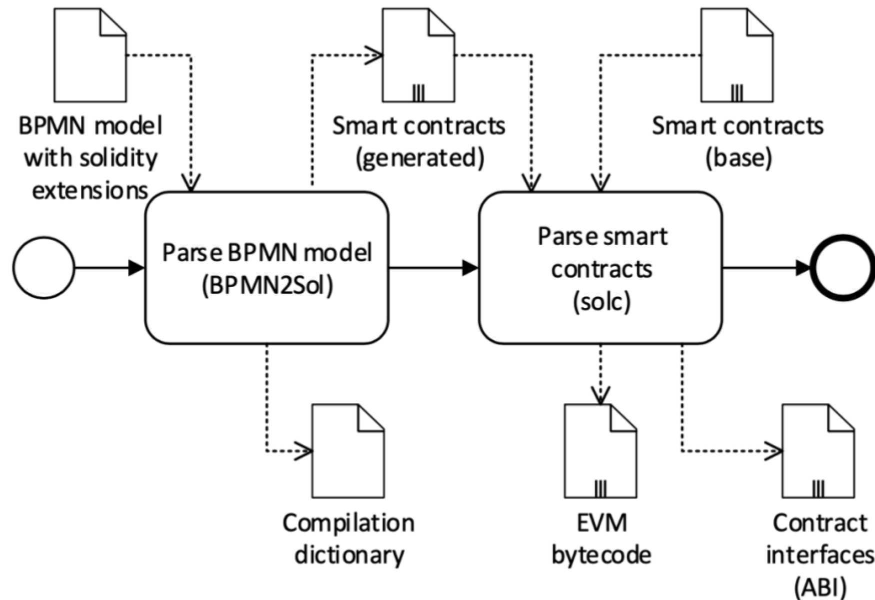

Figure 5: Caterpillar's compilation process [8]

2) Secondly, the Solidity compiler (**solc**) compiles the generated Solidity smart contracts and generates EVM bytecode and ABI definitions. The EVM bytecode is required to deploy smart contracts on the Ethereum blockchain.

At a higher-level view, the goal of the thesis is to run generated Solidity smart contracts on Hyperledger Fabric. Caterpillar has implemented an Ethereum adapter which uses Web3.js to manage all the interactions of the Ethereum blockchain with Caterpillar. Similarly, a Hyperledger adapter is required in our solution, which can support the execution of Solidity smart contracts generated by Caterpillar on the Hyperledger Fabric network, as shown in Figure 6.

4.2 Our Approach

As discussed in section 4.1, the Caterpillar BPMS prototype generates smart contracts from the BPMN process model and provides functionality to execute them on Ethereum, a permissionless blockchain. Currently, Caterpillar utilises an Ethereum adapter to perform all blockchain already functionalities. Since the goal of the thesis is to integrate permissioned blockchain (Hyperledger Fabric) into Caterpillar, a similar Hyperledger adapter is required. This adapter will be responsible for processing all the operations of Caterpillar on the Fabric network. To explain our approach in detail, the extended architecture of Caterpillar (as shown in Figure 6) will be discussed, which is organised into three main layers.

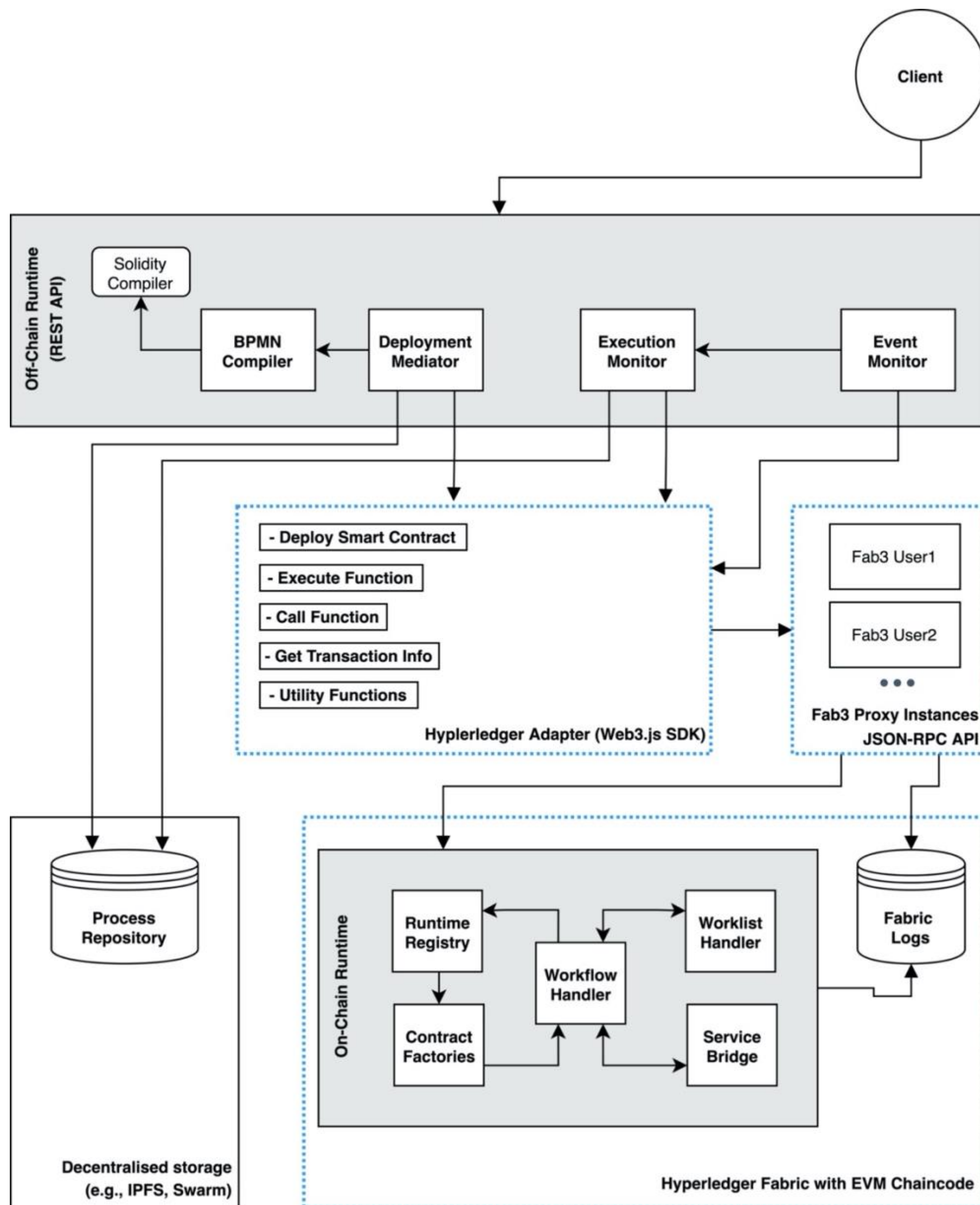

Figure 6: Thesis work architecture

The Blue dotted component in the bottom layer, which is termed as “On-chain runtime”, is responsible for smart contracts used for managing and executing business processes. As

described in section 4.1, the primary role of this component in Caterpillar is to instantiate processes and keep track of specific process data using a different set of smart contacts. The internal workflow of the On-chain runtime will remain the same as in Caterpillar. The main difference is currently, Caterpillar uses the Ethereum blockchain network to deploy all these modules; on the other hand, the extended Caterpillar will utilise Hyperledger Fabric blockchain to manage all the smart contacts. Like "Ethereum Logs", "Fabric Logs" are responsible for monitoring the events for transactions, fetching block information, smart contracts executions logs, and data related to it.

The middle layer is organised into the following two dotted components:

(i) Hyperledger adapter
"Hyperledger adapter" acts as a bridge between On-chain runtime and Off-chain runtime components of extended Caterpillar. It provides a list of functions (e.g., deploy smart contract, execute smart contract functions, read data from the blockchain, etc.), which Off-chain runtime use to interact with the Fabric network. The Hyperledger adapter uses Web3.js SDK to provide these functions, which abstracts JSON-RPC API operations for developers.

(ii) Fab3 proxy instances
In its original state, Hyperledger Fabric doesn't provide support for JSON-RPC API to interact with its network. However, Fab3 is a service provided by Hyperledger Fabric, which acts as a proxy between adapter and network. For example, when the Hyperledger adapter wants to initiate a request to the network, the intended request will be sent to the Fab3 proxy instance first using Web3.js SDK. Then the Fab3 proxy will structure this request in an accepted format and forward it to the Fabric network.

In a nutshell, middle layer components are responsible for providing an interface for the interaction between Off-chain and On-chain runtimes.

Finally, the layer at the top is referred to as "Off-chain runtime". As discussed in section 4.1, Caterpillar implements REST API to interact with its functionalities and provides tools to manage the business process on On-chain runtime. However, the Off-chain runtime is extended to provide integration with the Hyperledger adapter, as indicated in Figure 6. Afterwards, in addition to Ethereum network support, users/organisations can deploy and manage their business process on Hyperledger Fabric, a permissioned blockchain, which they couldn't be able to previously inside Caterpillar.

4.3 On-Chain Runtime: Hyperledger Fabric

The "On-Chain Runtime" of Caterpillar is responsible for providing a blockchain network to manage and execute smart contracts. These smart contracts enable mutually untrusted parties to implement and execute collaborative business processes without requiring third authorities. Previously, Caterpillar's "Off-Chain Runtime" was utilising public Ethereum blockchain, which has drawbacks like lack of data privacy, low transactional speeds and public network. In contrast, our approach to extending the Caterpillar BPMS utilises the Hyperledger Fabric network, a permissioned blockchain.

Hyperledger Fabric provides organisations with a consortium network, enabling them to configure a membership-based blockchain network, thus solving the data-privacy issue because everyone cannot join the network. Additionally, membership-based networks have fewer nodes, enabling organisations to perform transactions much faster.

As discussed in Section 4.2, the "On-chain Runtime" needs the required functionalities (i.e., Fabric network, EVM Chaincode to support Solidity smart contracts). So, the main concepts of Hyperledger Fabric architecture and the interaction of Solidity smart contracts with Hyperledger Fabric through EVM Chaincode need to be discussed. Hyperledger Fabric is the consortium network that Caterpillar will use to deploy and manage smart contracts using the Hyperledger adapter.

4.3.1 Hyperledger Fabric Architecture

Hyperledger Fabric is a very famous project under Hyperledger, a permissioned blockchain. Its architecture includes peers, certificate authority, membership service provider, orderer, organisation, consortium, channel, and ledger [54].

1) *Peer*

"Peers are a fundamental element of the network because they host ledgers and smart contracts" [54]. Peers are also termed as nodes in Hyperledger Fabric Blockchain. There are different types of peers, and each of them has specific features and responsibilities within the network [55].

a) **Anchor Peer**: The peers in the Fabric network configured at the time of Channel configuration are called Anchor peers. These peers are responsible for receiving the updates and then broadcasting them to other network peers. These peers are discoverable as well.
b) **Endorser Peer**: An endorser peer in Fabric receives a "transaction invocation request" from the client application. The endorser peer has a smart contract (Chaincode), and its responsibility is to verify the certificate details. It either accepts or rejects the request. Additionally, only the endorser node is responsible for executing the Chaincode, so adding Chaincode at every node is unnecessary. This characteristic of Fabric enhances the scalability of this network.
c) **Orderer Peer**: is responsible for central communication for the Fabric network. It is also responsible for constant Ledger state across the network. It generates the block, which is delivered to all the peers afterwards.

2) *Certificate Authority*

Everyone who wants to interact with the Fabric network needs an identity. Each user gets a verifiable digital unique identity from Certificate Authority (CA). It has also implemented the membership mechanism, which makes Hyperledger Fabric a permissioned blockchain.

3) *MSP*

Membership Service Provider (MSP) is a component which provides an abstraction to membership operations by simplifying the cryptographic mechanisms and protocols behind issuing digital certificates, accepting or rejecting the certificates, and user authentication.

4) *Organisation*

An organisation in Hyperledger Fabric is defined as a virtual entity that has access to the ledger. Organisations grant identities to the participants to make every

transaction occurring on the blockchain identifiable and transparent. It shows that only permissioned participants can perform the transaction on the Hyperledger Fabric blockchain, and these permissions are issued by the Fabric certificate authority.

5) *Consortium*

A group of organisations that share a need to perform a transaction. Moreover, it could also share the set of permissions over the network.

6) *Channel*

Channels on Fabric are a form of the ledger or methods for organising and securing data. Different organisations connect with channels and gain access to specific ledgers. Individuals are then granted access rights depending on categories. For instance, in a hospital, a doctor may have a higher level of access, allowing them to read the patients' data, but patients cannot read the data of others.

7) *Ledger*

It is stored over the peer and consists of the World State of the blockchain. All the transactions executed on the blockchain network are appended to the ledger. It is immutable.

Besides explaining the architecture of Hyperledger Fabric, other important concepts like consensus mechanism and smart contracts need to be discussed, which are as follows:

8) *Consensus Mechanism*

The consensus mechanism in Fabric can be divided into 3 phases: Endorsement, Ordering, and validation [13, 44, 56].

- Endorsement is performed by a policy, for instance, m out of n signatures. The participants will endorse a transaction based on the policy.
- The ordering phase admits the endorsed transactions and approves the order which will be committed to the ledger.
- The last phase is the validation phase which takes a block of ordered transactions. It will validate the rightness of the results by checking the endorsement policy and double-spending.

9) *Chaincode*

Chaincode is a "smart contract" of a Fabric network typically written in node.js, Go, or Java. A secure Docker container abstracted from the endorser peer is used to run the Chaincode. It is responsible for initiating and managing the ledger state using transactions requested by client applications [54].

Finally, the basic transaction workflow is explained in Figure 7 to better understand the components and concepts discussed above.

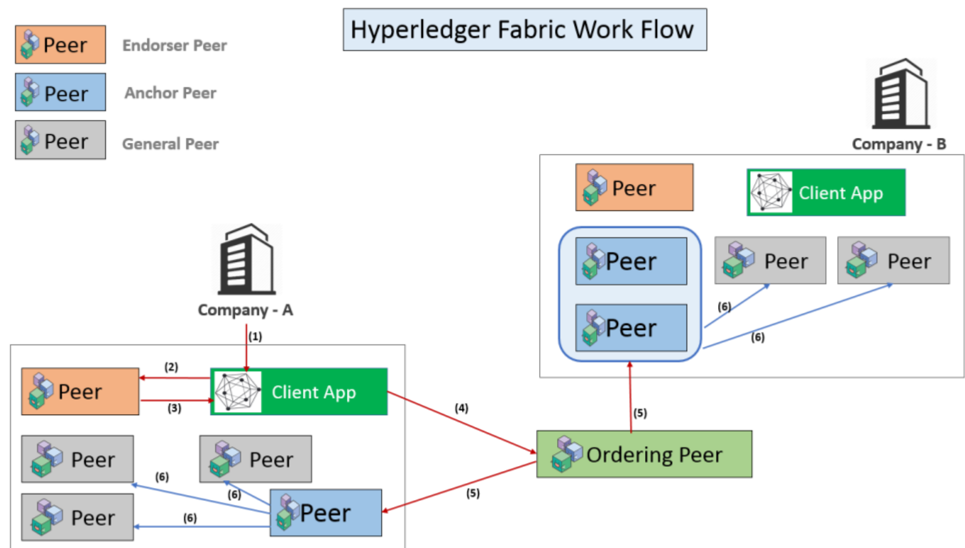

Figure 7: Hyperledger Fabric transaction workflow[55]

10) Transaction Workflow

First, a participant requests a transaction using the client application. The next step is the client's transaction broadcasted to the endorser peer. Consequently, the endorser peer validates or rejects the transaction after examining the certification details and sends the response to the client application. The client transfers this transaction to the orderer peer so that it can be placed appropriately in the block. The orderer node embraces the transaction into a block and assigns the block to the Anchor peer nodes. These anchor nodes are members of the Hyperledger Fabric Network and play a role in broadcasting the block to all the peers inside their organisation. The transaction completes when all the network peers update their ledgers and add the latest block. Lastly, all the network is synchronised, and the transaction workflow is completed [55].

4.3.2 Solidity Smart Contracts on Hyperledger Fabric

As discussed in section 4.3, Fabric's smart contract is called Chaincode, and a Chaincode is usually written in JavaScript, Go, and Java programming languages. By default, Solidity smart contracts are not supported on Fabric. But recently, IBM technical ambassadors have added support to execute Solidity smart contracts on Hyperledger Fabric utilising the Burrow EVM that wraps around the Chaincode, called EVM Chaincode [15]. To increase the number of options on the blockchain network, Hyperledger Fabric supports smart contracts of EVM bytecode. With this addition, we can write contracts in Vyper and Solidity as well. Apart from the capability of smart contracts, Fabric has provided a web3 proxy called Fab3, which enables developers to program web applications using existing Ethereum's web3.js SDK easily.

To integrate EVM on Hyperledger, EVM has been divided into two parts now. One crucial part is EVM User Chaincode, and another one is Fab3 (a web3 provider) [15].

1) *User Chaincode*

The EVM User Chaincode wraps Hyperledger Burrow EVM[2] to provide the functionality to run Ethereum based smart contracts, as shown in Figure 8. Additionally, the functionalities to accept queries regarding the contract code and accounts have been added as well. Some basic designs for this process are discussed below.

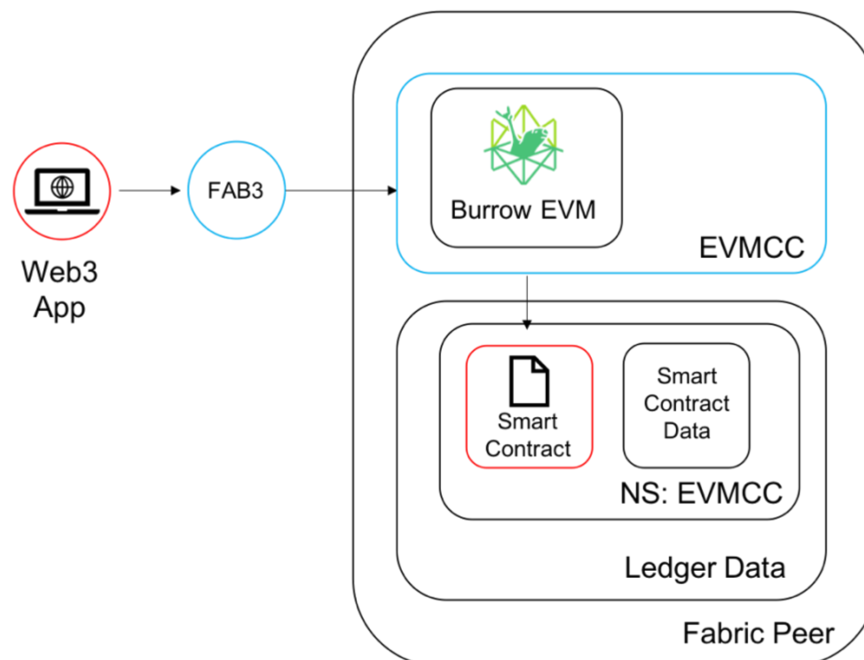

Figure 8: Burrow EVM integration with Fabric [57]

- ***Accounts***

  Ethereum possesses two types of accounts: Contract accounts and Externally Owned Accounts (EOA). EOA is a form of the address produced using the user's public key and some amount of Ether. In the recent update, though Fabric has introduced EVM, Ether or any other currencies/tokens were not included, so EOA is not stored clearly. In contrast, Contract accounts include only the bytecode of EVM runtime regarding a contract. Following Ethereum, these accounts will be stored on the EVM Chaincode. Smart contracts deployment through EVM will proceed automatically because it excludes the manual process of installing smart contracts like in the Fabric workflow.

- ***Gas***

  Gas is the amount of Ether required on EVM to execute a transaction. The miners do not want to risk their amount of Ether because if the transaction is not completed, then computational power (Ether) will go in vain. In the recent

[2] https://github.com/hyperledger/burrow

update, the EVM Chaincode provides a good amount of gas so that transactions will proceed automatically.

2) *Fab3*

JSON-RPC API is another essential component adopted from the Ethereum network ecosystem called Fab3. As shown in Figure 8, Fab3 is a proxy service that provides a bridge interface between EVM Chaincode and Web.js SDK. It is developed based on Fabric GO SDK. Currently, Fab3 is not supporting the complete API capabilities due to some critical differences in Ethereum and Fabric networks. However, it still supports adequate instructions to permit the dApps written in the web3.js library.

Note that the Fab3 service needs to be started separately for each user in the Hyperledger Fabric. And it uses the Fabric discovery service to map out user addresses (Ethereum like address) from Fabric Identity x.509 public certificates.

5 Solution: Hyperledger Adapter

This chapter will discuss the Hyperledger adapter solution and how its integration into Caterpillar BPMS will help the "Off-chain Runtime" manage and execute generated Solidity smart contracts on the "On-chain Runtime".

5.1 Hyperledger Adapter

Caterpillar is a BPMS prototype that generates Solidity smart contracts from the BPMN model. These smart contracts are executed on the Ethereum blockchain by an adapter called Ethereum adapter. Ethereum adapter on a higher level creates a bridge between Off-chain & Off-chain runtimes. Similarly, a Hyperledger adapter is required to interact with the Hyperledger Fabric network.

As described in section 4.3.2, Hyperledger Fabric provides the functionality to run Solidity smart contracts by utilising EVM Chaincode. Additionally, Fab3 proxy is also available that provides JSON-RPC API support. It works as a bridge between the Ethereum world and the Hyperledger Fabric one. It means Web3.js SDK can be utilised to develop functions to deploy, execute and monitor Solidity smart contracts. For example, Web3.js initiates a request to Fab3 proxy URL, which generates user address from Fabric Identity x.509 certificates using discovery service and then forward the request to Fabric network using GO SDK.

Based on the existing Ethereum adapter, below is a list of main functionalities required that the Hyperledger adapter will rely on to interact with the Fabric network.

1) *Set Provider*

The initialising phase of the Hyperledger adapter will require a function that will set the URL of the Fab3 proxy service for future network calls. It is essential because different Fab3 proxies are running for each user. For example, if User1 wants to make a transaction, Fab3 for User1 will be used. On the other hand, if User2 intends to interact with the network, Fab3 for User2 is required.

2) *Get Default Account*

During interaction with the network, there are scenarios where we require the user address that is making that transaction. As in EVM Chaincode, user addresses are mapped on the fly from their public keys; this function becomes very important.

3) *Deploy Smart Contract*

Caterpillar BPMS compiler engine generates smart contracts from the BPMN process model. These smart contracts need to be deployed on the network. Like Ethereum adapter, the "deploy smart contract" function should be implemented to deploy bytecode of smart contracts on Fabric Network. This function will return deployed contract address to execute smart contract actions later.

4) *Execute Smart Contract Function*

Ethereum adapter has functionality available to run smart contract functions using the signature from ABI. Similarly, execute smart contract function feature will be developed in the Hyperledger adapter. It will help the Off-chain runtime of Caterpillar to execute smart contract functions and get the results from its execution.

5) *Retrieve Transaction Info*

The Off-chain runtime of Caterpillar performs specific operations if the transaction is successfully performed on the blockchain network. Therefore, a mechanism is required to determine whether the transaction is mined and retrieve its associated data.

6) *Retrieve Event Info*

Caterpillar implements and executes collaborative business processes in the form of smart contracts. When some particular tasks are completed, smart contracts emit specific events during execution. A function needs to be implemented in the Hyperledger adapter that monitors the logs ("Fabric Logs") and determines if the provided event happened.

7) *Encode & Decode Functions*

Ethereum EVM supports parameters in HEX format. Therefore, encoding and decoding function parameters to and from HEX format functionality are required. Moreover, other encoding ABI signature and events capabilities will be added as well.

5.1.1 Fab3 Proxy Integration

Hyperledger Fabric configured together with EVM Chaincode provides the functionality to execute Solidity based smart contracts on the Fabric network. However, running a Solidity smart contract operation on Fabric is different from Ethereum. To deploy a Solidity smart contract, the *peer-cli* command bundled with "*channel_name*", "*orderer*", and "*certificates*" needs to be invoked. The example peer chaincode command is shown in Figure 9.

```
peer chaincode invoke -n evmcc -C <channel-name>  -c '{"Args":["0000000000000000000000000000000000000000","
<byte_code>"]}' -o orderer.example.com:7050 --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.exampl
e.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

Figure 9: Example Peer Chaincode command

However, Hyperledger Fabric foundation has provided SDKs for different languages like Golang, Node.js, etc. These SDKs abstract the complexities and provide easy ways for users to execute operations. Although these tools remove a lot of complexities, none of them supports Solidity smart contracts execution like Ethereum does, using Web3.js SDK. Fab3 is a Golang-based service that creates an HTTP interface to communicate with Hyperledger Fabric, just like Ethereum's JSON-RPC API. As discussed in Section 4.3.2, Fab3 implements some of the compatible JSON-RPC API operations from Ethereum like *eth_accounts*, *eth_call*, *eth_sendTransaction*, etc. Internally these proxy functions will structure the commands in Fabric *peer-cli* format and execute them on the Fabric network.

Hyperledger adapter will utilise Web3.js SDK and Fab3 proxy service instances to communicate with EVM Chaincode running inside Fabric network, as shown in Figure 10. The detailed flow of the integration is as follows:

1. Caterpillar's Off-Chain runtime asks Hyperledger adapter to perform an operation (e.g., deploy smart contract, execute smart contract function, etc.) on Fabric network.
2. Based on the type of operation, the Hyperledger adapter will perform prerequisite operations internally, like encoding the data.

3. Afterwards, the Hyperledger adapter will initialise the Web3.js SDK instance with the Fab3 proxy instance
4. Hyperledger adapter will call the appropriate function of the Web3.js, which then invokes the Fab3 proxy service
5. Fab3 proxy then format the operation to Fabric compatible command and execute it on Fabric network
6. Finally, the Hyperledger adapter will get the response after processing the operation. In case of an error response, it will throw an exception. On the other hand, if the response is successful, the Hyperledger adapter will perform post-operation like decoding the response data, etc.

5.1.2 Handling multiple Fab3 Instances

Based on the implementation of EVM Chaincode and Fab3 discussed in section 4.3.2, Fab3 needs to be run separately for each identity in the Fabric network. Therefore, a mechanism is required to let Caterpillar BPMS know from which user the request is initiated. Caterpillar exposes REST API to interact with its BPMN prototype. Similarly, the same API endpoints with different signatures will be available for Hyperledger Fabric network support. To solve the user discovery problem, the middleware will be added, which will take *FAB3-URL* as header input to the request and call the "Set Provider" method of the Hyperledger adapter. This way Hyperledger adapter will be notified from which user the request is initiated, and it will pass that information to the Fabric network via Fab3 proxy. This flow is indicated in Figure 10.

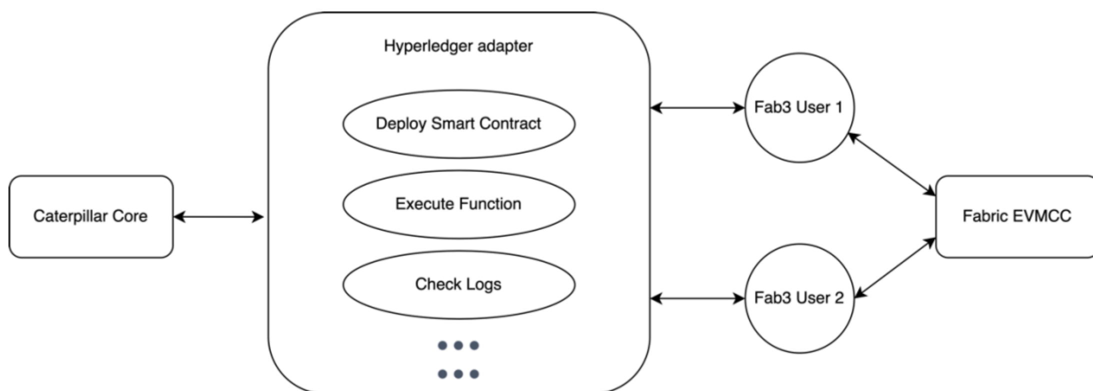

Figure 10: Hyperledger Fabric adapter

After the completion of the Hyperledger adapter and user discovery middleware, all the core services available need to be refactored so that they can support both the Ethereum and Hyperledger adapter based on the user requirement.

5.2 Hyperledger Adapter Integration

All the services and modules in Caterpillar directly instantiate the instance of the Ethereum adapter to interact with the Ethereum network. To support both Ethereum and Hyperledger adapters, a *dependency injection*[3] design pattern is introduced. All the services and modules will get the relevant adapter based on the type of request. In general, all the Hyperledger Fabric network calls will inject a Hyperledger adapter to interact with the network. Below

[3] **Dependency injection** is basically providing the objects that an object needs (its dependencies) instead of having it construct them itself

we will discuss how the Hyperledger adapter facilitates Caterpillar's Off-Chain runtime to execute smart contracts on the On-chain runtime, Hyperledger Fabric.

The off-Chain runtime component of Caterpillar consists of three primary services, i.e., "*Deployment Mediator*", "*Execution Monitor*", and "*Event Monitor*", as shown in Figure 6. Each of these services uses a Hyperledger adapter to execute their operation on On-Chain runtime, which is Hyperledger Fabric in our case.

Off-Chain runtime is the starting point of Caterpillar, where users interact to perform business process operations. In the case of process model registration, Off-Chain runtime first compiles the provided BPMN model and generates Solidity smart contracts using BPMN Compiler. Next, the Off-Chain runtime initiates the Deployment Mediator to deploy these smart contracts. The deployment mediator takes this process model and smart contracts and carries out all the internal processes. Afterwards, Deployment Mediator compiles the smart contracts into bytecode and will initiate a Hyperledger adapter to deploy the bytecode of smart contracts. From this point, the Hyperledger adapter is responsible for encoding the smart contract into a contract instance using web3.js SDK. Next, the encoded smart contract is deployed using the Fab3 proxy, and the Hyperledger adapter receives *contractAddress*, the smart contract's address, for future reference. This *contractAddress* is used later to execute smart contract functions.

In addition to deployments, Deployment Mediator also links deployed smart contracts and processes to Runtime Registry, which keeps track of all the instances and their relationships on the blockchain. For example, after the deployment of the BPMN model, Deployment Mediator saves its reference in Runtime Registry. This is done by executing a Runtime Registry's smart contract function. Deployment Mediator asks Hyperledger adapter to execute the function of a deployed smart contract by providing the *contractAddress,* ABI, function name, and parameters required. Next, the Hyperledger adapter encodes the function using Web3.js (*encodeFunctionCall*) and executes the function by sending the transaction to Fab3. Fabric proxy then processes this operation on the Fabric network and returns *transactionHash*, which can be used to retrieve the data related to a particular transaction on the chain.

Execution Monitor, a component of Off-Chain runtime, keeps track of the transactions happening on the blockchain. The execution monitor subscribes to the particular transaction using *transactionHash* and performs post tasks once the transaction is fully mined on the chain. Although, this is more applicable in Ethereum's case because transactions take longer to get mined due to the size of the public network. However, this process is mitigated in Hyperledger adapter in a synchronous manner because Fab3 currently doesn't support subscription-based operations. So, once a transaction is submitted to the Fabric network, Execution Monitor waits for its completion instead of subscribing to it and asking the Hyperledger adapter to give its data. After successfully getting the data back from the Hyperledger adapter, Execution Monitor performs the post-processing as needed. Note that making transactions synchronous is not an ideal behaviour because delays are possible in Hyperledger Fabric as well if the network is large, but it can be solved by creating a blockchain polling service on top of Fab3. However, working on this solution is out of the scope of this thesis as our main goal is the execution of smart contracts on Hyperledger Fabric.

Finally, Event Monitor is a component of Off-Chain runtime, which is responsible for triggering operations when a particular event happens on the blockchain by a smart contract. As discussed previously, the Hyperledger adapter doesn't support listening to the events.

This problem is solved by making transactions synchronous and manually processing the "Hyperledger logs" to decide when to emit the event (e.g., when a process has been completed) to "Event Monitor" or any application consuming Caterpillar's REST API.

5.2.1 Extended REST API Overview

Caterpillar exposes all its functionalities through REST API, which will be extended to support the execution of processes on Hyperledger Fabric using the Hyperledger Adapter. Caterpillar's extended REST API consists of four components: (i) *Runtime Registry*, which is responsible for the deployment of runtime registry smart contract and retrieving information related to it. (ii) *Compilation-based Engine*, which involves BPMN models, the associated compilation artefacts and their instances, processes and their states on the blockchain, and worklists that involve user interactions. (iii) *Interpretation-based Engine,* which involves the deployment of "BPMN interpreter", deployment and interaction with BPMN elements, creating new process instances, and verifying the process states etc. (iv) *Dynamic Access Control*, which involves the deployment and interaction with role-based policy, task-role-map, and access control.

Caterpillar's Repository[4] provides detailed resources and documentation to understand the components mentioned earlier. Overview of REST API for each component in extended Caterpillar is as follows:

Table 1 summarises the list of REST API resources available for the Runtime Registry component.

Table 1: Caterpillar's Runtime Registry REST API (extends from [8])

Verb	URI	Description
POST	/hyperledger/registries	Compiles the Solidity smart contract available in Caterpillar's Repository into bytecode and ABI, deploys the bytecode on Hyperledger Fabric and saves the compilation artefacts and contract address in the process repository.
GET	/hyperledger/registries/:registryAddress/address	Retrieve registry information saved in process repository
GET	/hyperledger/registries/:addressId	Retrieve registry information saved in the process repository

Table 2 summarises the list of REST API resources available for the Compilation-based engine component.

Table 2: Caterpillar's Compilation-based REST API (extends from [8])

Verb	URI	Description
POST	/hyperledger/models	Compiles and Register a BPMN model
GET	/hyperledger/models	Retrieve a list of registered BPMN models
GET	/hyperledger/models/:mHash	Retrieve metadata of a BPMN model

[4] https://github.com/orlenyslp/Caterpillar

POST	/hyperledger/models/:mHash/processes	Create a new process instance for a provided process model
GET	/hyperledger/models/:mHash/processes	Retrieved all created process instances for a given process model
GET	/hyperledger/processes/:pAddress	Get the current state of a process instance
PUT	/hyperledger/worklists/:wlAddress/workitems/:wiIndex	Checks in a work item

Table 3 summarises the list of REST API resources available for Interpreter-based engine component.

Table 3: Caterpillar's Interpreter-based REST API (extends from [8])

Verb	**URI**	**Description**
POST	/hyperledger/interpreter	Creates a new instance of a "BPMN Interpreter"
POST	/hyperledger/interpreter/models	Compiles and Register a BPMN model
GET	/hyperledger/interpreter/models	Retrieve list of registered BPMN models
GET	/hyperledger/interpreter/models/:mHash	Retrieve metadata of a BPMN model
POST	/hyperledger/i-flow	Create a new empty IFLOW node.
PATCH	/hyperledger/i-flow/:cfAddress/element	Updates a BPMN element into a given IFLOW node
PATCH	/hyperledger/i-flow/:cfAddress/child	Links a child node in a given IFLOW node
PATCH	/hyperledger/i-flow/:cfAddress/factory	Relates a factory with a sub-process in a given IFLOW node
GET	/hyperledger/i-flow/:cfAddress	Retrieve the information of a given IFLOW node
POST	/hyperledger/i-flow/:cfAddress/i-data	Creates a new process case from a given IFLOW root node.
GET	/hyperledger/i-flow/:cfAddress/i-data	Retrieves all the process cases created from a IFLOW root node
GET	/hyperledger/i-flow/:eIndex/i-data/:pcAddress	Checks-out a task in a given process case
PATCH	/hyperledger/i-flow/:eIndex/i-data/:pcAddress	Checks-in a task in a given process case
GET	/hyperledger/i-data/:pcAddress	Retrieves the current state of a given process case

Table 4 summarises the list of REST API resources available for Dynamic Access Control.

Table 4: Caterpillar's Dynamic Control REST API (extends from [8])

Verb	**URI**	**Description**

POST	/hyperledger/rb-policy	Parse and deploy role-based policy
GET	/hyperledger/rb-policy/:rbPolicyAddr	Retrieve role-based policy metadata
POST	/hyperledger/task-role-map	Parse and deploy task role map
GET	/hyperledger/task-role-map/:taskRoleMAddr	Retrieve task role map
POST	/hyperledger/access-control	Deploy access control
GET	/hyperledger/access-control/:accessCtrlAddr	Retrieve access control metadata
GET	/hyperledger/rb-opertation/:pCase	Retrieve policy address for a given process case
GET	/hyperledger/rb-opertation/:pCase/state	Retrieve role state for a given process case
PATCH	/hyperledger/rb-opertation/:pCase/nominate-creator	Nominate a case creator for a given process case
PATCH	/hyperledger/rb-opertation/:pCase/nominate	Nominate a user
PATCH	/hyperledger/rb-opertation/:pCase/release	Release
PATCH	/hyperledger/rb-opertation/:pCase/vote	Vote

6 Implementation And Evaluation

In this chapter, we will discuss the technologies used and the work produced to implement the solution. Furthermore, this chapter will also provide information on how we tested and analysed the solution implemented.

6.1 Technologies Used

Our thesis implementation has two main components (i.e., Hyperledger Fabric Network and Caterpillar). Technologies used for the setup of Fabric network and development of Hyperledger adapter are listed below:

- **Docker**[5]: It is a containerisation platform that will run Fabric network components.
- **Golang**[6]: It is a programming language that will be used to build and run Fab3 proxy.
- **NodeJS**[7]: a JavaScript runtime that Caterpillar uses to run its services.
- **ExpressJS**[8]: It is a NodeJS framework that allows the development of REST API
- **TypeScript**[9]: JavaScript with the support for types
- **Jest**[10]: A JavaScript-based testing framework designed to ensure the correctness of any JavaScript application.

6.2 Hyperledger Fabric Network Setup

Hyperledger Fabric is a permissioned blockchain that provides Docker (a containerisation platform) images and **example network configurations** to facilitate the development. In this thesis, Hyperledger Fabric **release-1.3** is explicitly used. The network architecture in Hyperledger Fabric is highly modular and scalable. Hyperledger provides developers with a set of developed test-networks[11] that facilitate them to understand the structure and workflow better. With the help of **fabric-samples**, bootstrapping the network is relatively easy. In a nutshell, **fabric-samples** provide all the essential tools and examples to kickstart the development.

All the prerequisites and tools can be installed easily by following the documentation (https://hyperledger-fabric.readthedocs.io/en/release-1.3/install.html).

For the further development of the feature, the **first-network** sample will be followed, which, by default, consists of one channel, one orderer and two organisations with two peers each, as shown in Figure 11. Moreover, each organisation has its own certificate authority (CA), which manages the user credentials.

[5] https://www.docker.com/
[6] https://go.dev/
[7] https://nodejs.org/en/
[8] https://expressjs.com/
[9] https://www.typescriptlang.org/
[10] https://jestjs.io/
[11] https://github.com/hyperledger/fabric-samples/tree/release-1.3

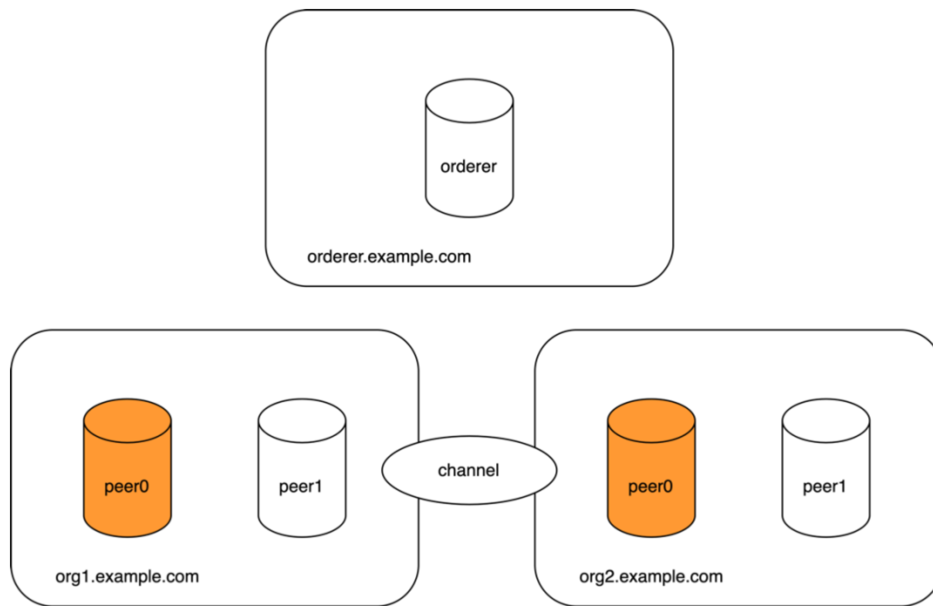

Figure 11: Hyperledger Fabric First Network

Additionally, each peer has one user/participant available, which can be increased by editing the "*fabric-samples/first-network/crypto-config.yaml*" configuration file, as indicated in Figure 12.

```
77    - Name: Org2
78      Domain: org2.example.com
79      EnableNodeOUs: true
80      Template:
81        Count: 2
82      Users:
83        Count: 4
84
```

Figure 12: Crypto Configuration

Fabric First Network example installs an example chaincode (smart contract) and runs a few tests. For simplicity and to make the network as fast as possible, some lines in the "*fabric-samples/first-network/scripts/script.sh*" file need to comment out, which will stop instantiating chaincode and running chaincode test (refer to Figure 13).

```
84
85  # ## Install chaincode on peer0.org1 and peer0.org2
86  # echo "Installing chaincode on peer0.org1..."
87  # installChaincode 0 1
88  # echo "Install chaincode on peer0.org2..."
89  # installChaincode 0 2
90
91  # # Instantiate chaincode on peer0.org2
92  # echo "Instantiating chaincode on peer0.org2..."
93  # instantiateChaincode 0 2
94
95  # # Query chaincode on peer0.org1
96  # echo "Querying chaincode on peer0.org1..."
97  # chaincodeQuery 0 1 100
98
99  # # Invoke chaincode on peer0.org1 and peer0.org2
100 # echo "Sending invoke transaction on peer0.org1 peer0.org2..."
101 # chaincodeInvoke 0 1 0 2
102
103 # ## Install chaincode on peer1.org2
104 # echo "Installing chaincode on peer1.org2..."
105 # installChaincode 1 2
106
107 # # Query on chaincode on peer1.org2, check if the result is 90
108 # echo "Querying chaincode on peer1.org2..."
109 # chaincodeQuery 1 2 90
110
```

Figure 13: *script.sh* file

6.2.1 Running the network

Fabric First network provides a "*byfn.sh*" script, which will bundle all the configurations and run the network. Few of the main inputs available for "byfn.sh" script are as follows:

a) "*byfn.sh up*" is available, which will run all the docker containers and scripts based on the configuration provided.
b) "*byfn.sh down*" will switch off the network and remove all the docker containers.

When the "*byfn.sh up*" script is run, the process will be as follow:

a) **Run Docker Containers:**
First, it will run all the docker containers for the components of the network. Figure 14 shows the related output.
b) **Initialise the organisations and join the channel**:
Once all the intended docker containers are running, the organisations (two in our case, i.e., org1 and org2) will be initialised, and each peer owned by the respected organisation will join the channel. This is all based on the configuration provided, which can be easily modified depending on the use case. Figure 15 shows the related output.

```
first-network — jazibsawar@Jazibs-MBP — ..first-network — -zsh — 131×34

######################################################################
#######     Generating anchor peer update for Org2MSP    ##########
######################################################################
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Or
g2MSP
2022-05-02 04:48:54.459 EEST [common/tools/configtxgen] main -> INFO 001 Loading configuration
2022-05-02 04:48:54.487 EEST [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2022-05-02 04:48:54.487 EEST [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update
+ res=0
+ set +x

/Users/jazibsawar/Library/Python/3.7/lib/python/site-packages/requests/__init__.py:91: RequestsDependencyWarning: urllib3 (1.26.3)
or chardet (3.0.4) doesn't match a supported version!
  RequestsDependencyWarning)
Creating network "net_byfn" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer1.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating volume "net_peer1.org2.example.com" with default driver
Creating peer1.org1.example.com ... done
Creating peer1.org2.example.com ... done
Creating peer0.org2.example.com ... done
Creating orderer.example.com    ... done
Creating peer0.org1.example.com ... done
Creating cli                    ... done
```

Figure 14: Run of docker containers

```
first-network — jazibsawar@Jazibs-MBP — ..first-network — -zsh — 137×49
Channel name : mychannel
Creating channel...
+ peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls true --cafile /opt/gopath/src/gith
ub.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert
.pem
+ res=0
+ set +x
2022-05-02 01:49:00.116 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 01:49:00.168 UTC [cli/common] readBlock -> INFO 002 Got status: &{NOT_FOUND}
2022-05-02 01:49:00.188 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2022-05-02 01:49:00.398 UTC [cli/common] readBlock -> INFO 004 Received block: 0
===================== Channel 'mychannel' created =====================

Having all peers join the channel...
+ peer channel join -b mychannel.block
+ res=0
+ set +x
2022-05-02 01:49:00.561 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 01:49:00.644 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
===================== peer0.org1 joined channel 'mychannel' =====================

+ peer channel join -b mychannel.block
+ res=0
+ set +x
2022-05-02 01:49:03.811 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 01:49:03.861 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
===================== peer1.org1 joined channel 'mychannel' =====================

+ peer channel join -b mychannel.block
+ res=0
+ set +x
2022-05-02 01:49:07.008 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 01:49:07.060 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
===================== peer0.org2 joined channel 'mychannel' =====================

+ peer channel join -b mychannel.block
+ res=0
+ set +x
2022-05-02 01:49:10.210 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 01:49:10.255 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
===================== peer1.org2 joined channel 'mychannel' =====================

Updating anchor peers for org1...
+ peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org1MSPanchors.tx --tls true --cafile /opt/gopath/s
rc/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem
+ res=0
+ set +x
2022-05-02 01:49:13.435 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
```

Figure 15: Organisation setup and channel joins

Docker dashboard shows all the running containers, as indicated in Figure 16.

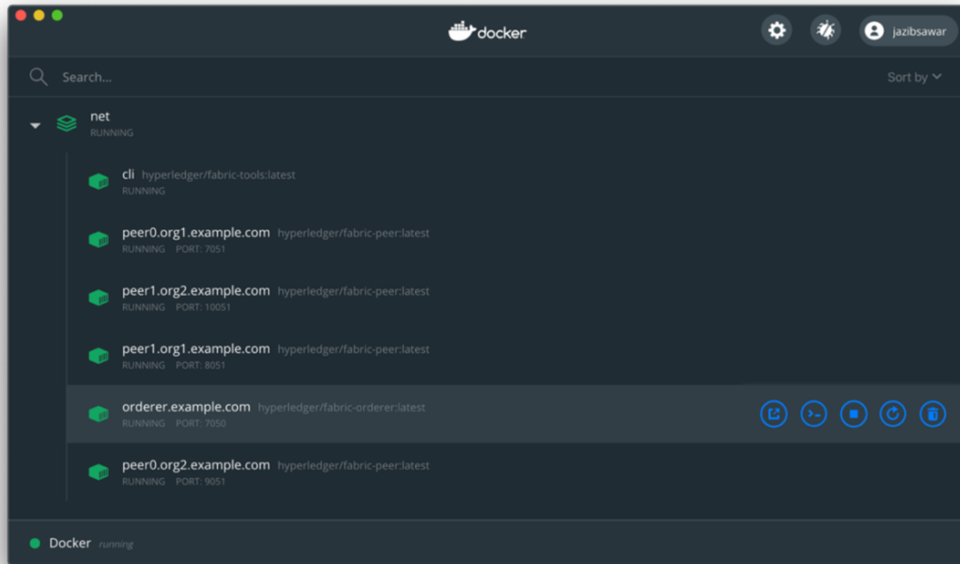

Figure 16: Running docker containers

Additionally, the "*peer-cli*" docker container is also running, as shown in Figure 16. It will be used to SSH into the Fabric network to configure EVM Chaincode, which will provide the functionality to deploy and execute Solidity smart contracts on Hyperledger Fabric.

6.3 Configure EVM Chaincode

As previously described, Hyperledger Fabric doesn't support Ethereum based smart contract execution out of the box. However, it introduced the functionality to support Solidity and Vyper based smart contracts by integrating Burrow EVM in the *fabric-chaincode-evm*[12] codebase, also referred to as EVM Chaincode (EVMCC) in Figure 8.

Firstly, EVM Chaincode needs to be mounted to the Fabric network by updating the *"fabric-samples/first-network/docker-compose-cli.yaml"* with volumes to include the *fabric-chaincode-evm*, as shown in Figure 17.

```
cli:
  volumes:
    - ./../../fabric-chaincode-evm:/opt/gopath/src/github.com/hyperledger/fabric-chaincode-evm
```

Figure 17: EVM Chaincode mounting

After that, EVMCC needs to be installed on peers using peer-cli, enabling the deployment and execution of smart contracts on Hyperledger Fabric. This process will be carried out by following the steps below:

a) **SSH into peer-cli:**
   To run scripts/commands on the Fabric network, it is required to SSH into peer-cli, which will be done by running the command "*docker exec -it cli bash*".

[12] https://github.com/hyperledger/fabric-chaincode-evm

```
Last login: Mon May  2 05:11:38 on ttys001
❯ docker exec -it cli bash
root@797a5867d475:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

Figure 18: SSH into peer-cli

The resulting output of this is shown in Figure 18.

b) **Install EVMCC on Peers**:
After SSH into peer-cli, the next step is to install EVMCC on all the peers. This process requires setting a few environment variables like *CORE_PEER_MSPCONFIGPATH*, *CORE_PEER_ADDRESS*, *CORE_PEER_LOCALMSPID*, and *CORE_PEER_TLS_ROOTCERT_FILE.* After that, the EMVCC will be installed using the "*peer chaincode install*" command. For simplicity and ease of use, the "*fabric-samples/first-network/scripts/install-evm.sh*" script is produced together with some modification in the "*fabric-samples/first-network/scripts/utils.sh*" file as shown in Figure 19 and Figure 20 respectively.

```
fabric-samples > first-network > scripts > install-evm.sh
1   scripts_dir="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null && pwd )"
2
3   # import utils
4   . $scripts_dir/utils.sh
5
6   export CC_SRC_PATH=github.com/hyperledger/fabric-chaincode-evm/evmcc
7   export CHANNEL_NAME=mychannel
8   export VERSION=1.0
9   export LANGUAGE=golang
10
11
12  # installChaincode {Org} {Peer}
13  installChaincode 1 0
14  installChaincode 1 1
15  installChaincode 2 0
16  installChaincode 2 1
```

Figure 19: Install-evm script

```
116  installChaincode() {
117    ORG=$1
118    PEER=$2
119    setGlobals $PEER $ORG
120    VERSION=${3:-1.0}
121    set -x
122    peer chaincode install -n evmcc -v 1.0 -l golang -p ${CC_SRC_PATH}
123    res=$?
124    set +x
125    cat log.txt
126    verifyResult $res "Chaincode installation on peer${PEER}.org${ORG} has failed"
127    echo "===================== Chaincode is installed on peer${PEER}.org${ORG} ===================== "
128    echo
129  }
```

Figure 20: Utils script modifications

Finally, the "*./scripts/install-evm.sh*" script will be run in peer-cli, which will install EVMCC on all four peers in two organisations available in our network. Figure 21 shows the related output.

```
jazibsawar — root@797a5867d475: /opt/gopath/src/github.com/hyperledger/fabric/peer — docker exec -it cli bash — 120×40
root@797a5867d475:/opt/gopath/src/github.com/hyperledger/fabric/peer# ./scripts/install-evm.sh
++ peer chaincode install -n evmcc -v 1.0 -l golang -p github.com/hyperledger/fabric-chaincode-evm/evmcc
2022-05-02 12:41:37.223 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2022-05-02 12:41:37.224 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2022-05-02 12:43:27.656 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
++ res=0
++ set +x
2022-05-02 12:09:30.296 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 12:09:30.335 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
==================== Chaincode is installed on peer0.org1 ====================

++ peer chaincode install -n evmcc -v 1.0 -l golang -p github.com/hyperledger/fabric-chaincode-evm/evmcc
2022-05-02 12:43:27.825 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2022-05-02 12:43:27.826 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2022-05-02 12:45:50.534 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
++ res=0
++ set +x
2022-05-02 12:09:30.296 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 12:09:30.335 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
==================== Chaincode is installed on peer1.org1 ====================

++ peer chaincode install -n evmcc -v 1.0 -l golang -p github.com/hyperledger/fabric-chaincode-evm/evmcc
2022-05-02 12:45:50.764 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2022-05-02 12:45:50.764 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2022-05-02 12:48:23.523 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
++ res=0
++ set +x
2022-05-02 12:09:30.296 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 12:09:30.335 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
==================== Chaincode is installed on peer0.org2 ====================

++ peer chaincode install -n evmcc -v 1.0 -l golang -p github.com/hyperledger/fabric-chaincode-evm/evmcc
2022-05-02 12:48:23.759 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2022-05-02 12:48:23.760 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2022-05-02 12:50:48.446 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
++ res=0
++ set +x
2022-05-02 12:09:30.296 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2022-05-02 12:09:30.335 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
==================== Chaincode is installed on peer1.org2 ====================
```

Figure 21: Install EVMCC on peers result

c) **Instantiate Orderer**
The last step is to instantiate the orderer with EVMCC within the default channel. Similarly, a utility script "*fabric-samples/first-network/scripts/instantiate-orderer.sh*" is produced for ease of configuration. After running the "*./scripts/instantiate-orderer.sh*" script, the orderer will be appropriately configured to accept EVM Chaincode calls in the Hyperledger Fabric network.

Finally, Hyperledger Fabric can now interact with Ethereum based smart contracts using peer-cli. Even though peer-cli can invoke smart contract operations, Fab3 proxy for Web3.js makes these operations calls much more accessible.

6.4 Setup Fab3 Proxy

Caterpillar uses Web3.js SDK to interact with the Ethereum blockchain. Similarly, Figure 8 shows that Fab3 provides a similar interface to support JSON-RPC API to invoke smart contract operations on the Fabric network. It is a Golang based service which can be run outside the Fabric network. Fab3 proxy service can be easily configurable by providing environment variables. For example, if an organisation wants to run Fab3 service for different users (i.e. User1, User2), multiple instances should be run by providing the user as an attribute. Additionally, it can be run in various organisations and channels if required. The following environment variables provide the mandatory configuration to run the Fab3 proxy instance [57]:

- **CONFIG**: It is a path to a compatible Fabric SDK Go config file. The *fabric-chaincode-evm* codebase has an example config available for *first-network*.
- **USER**: User identity being used for the proxy. It matches the users' names in the crypto-config directory specified in the config (i.e., User1, User2, etcetera).

- **ORG**: Organisation of the specified user.
- **CHANNEL**: Channel to be used for the transactions.
- **CCID**: ID of the EVM Chaincode deployed in your Fabric network.
- **PORT**: Port the proxy will listen on. If not provided default is 5000.

Like 6.3, the utility script "*fabric-chaincode-evm/scripts/fab3-vars.sh*" is created, which will help to run Fab3 instances for multiple users with ease, as shown in Figure 22.

```
fabric-chaincode-evm > scripts > fab3-vars.sh
1   USER=$1
2   PORT=$2
3
4   export FAB3_CONFIG=${GOPATH}/src/github.com/hyperledger/fabric-chaincode-evm/examples/first-network-sdk
5   # export FAB3_USER=${USER} # User identity being used for the proxy (Matches the users names in the cry
6   export FAB3_ORG=Org1  # Organization of the specified user
7   export FAB3_CHANNEL=mychannel # Channel to be used for the transactions
8   export FAB3_CCID=evmcc # ID of the EVM Chaincode deployed in your fabric network. If not provided defau
9   # export FAB3_PORT=${PORT} # Port the proxy will listen on. If not provided default is 5000.
10
11  ./bin/fab3 -u $USER -p $PORT
```

Figure 22: Running Fab3 proxy script

Finally, the "*fab3-vars.sh*" script is used to run Fab3 proxy by provider USER and PORT as inputs. Fab3 service instance for User1 on Port 5000 can be run by using "*./scripts/fab3-vars.sh User1 5000*" command. Figure 23 shows the related output.

```
❯ ./scripts/fab3-vars.sh User1 5000
{"level":"info","ts":1651498709.3587358,"logger":"fab3","caller":"cmd/main.go:149","msg":"Starting Fab3","port":5000}
```

Figure 23: Fab3 service for User1

At this point, the Fabric network, together with EVM Chaincode and Fab3 proxy, is properly set up to support Solidity smart contracts. Later, we will discuss the implementation of the Hyperledger adapter that is added to the Caterpillar.

6.5 Hyperledger Adapter Implementation

Caterpillar is a BPMS prototype that currently runs on top of the Ethereum blockchain and compiles process models specified in BPMN into smart contracts that translate the underline behaviours. It develops an Ethereum adapter using Web3.js that creates an interface that helps in blockchain actions like deploying smart contracts and executing functions. Similarly, a Hyperledger adapter is required to interact with the Fabric network.

As described in Section 4.3.2, Fab3 is a proxy service which provides JSON-RPC API capabilities to communicate with Hyperledger Fabric EMVCC, as shown in Figure 8. This means, in addition to peer-cli, Web3.js SDK can be utilised as well to deploy Solidity smart contracts and execute operations on Hyperledger Fabric. Based on the discussion in Chapter 5, the process of implementation will be divided into the following steps:

- Hyperledger Adapter Implementation
- Fab3 Middleware Implementation
- Hyperledger Adapter Integration

6.5.1 Hyperledger Adapter Implementation

Hyperledger adapter will be developed using Web3.js SDK and TypeScript. Below we will discuss the implementation of the essential functions:

Set Web3/Fab3 Provider
The "setProvider" function takes the Fab3 proxy instance URL as an input and sets it as a new provider for Web3 for future transactions. ***Note****: It will be an HTTP provider as Fab3 only exposes HTTP URLs for now. Additionally, it will not support sockets to listen to events and transactions as the Ethereum adapter does.*

```
1 export let setProvider = (newProvider: string) => {
2   web3.setProvider(newProvider);
3 };
```

Source Code 1: Set Web3/Fab3 Provider

Get Web3/Fab3 Provider
This function returns the currently set Provider in the Hyperledger adapter.

```
1 export let getProvider = () => {
2   return web3.currentProvider;
3 };
```

Source Code 2: Get Web3/Fab3 Provider

Deploy Smart Contract
This function takes smart contract information (i.e., bytecode, ABI) together with account information like user address and deploys the smart contract on the network. First, Web3 SDK converts input data (i.e., ABI, bytecode and arguments) into contract objects using the "*web.eth.Contract*" method, refer to Line number 8 of Source Code 3. Next, it deploys the smart contract using the encoded contract "deploy" method. Finally, on successful deployment, it will return a consumable Promise with the data (i.e., transaction hash and smart contract address on the blockchain). Later, the Off-chain Runtime of Caterpillar will use the returned smart contract address to invoke smart contract functions.

```
1 export let deploySmartContractSync = (
2   contractInfo: CompilationResult,
3   accountInfo: AccountInfo,
4   args?: any[]
5 ) => {
6   return new Promise<DeploymentOutput>((resolve, reject) => {
7     try {
8       let contractEncoding = encodeSmartContract(
9         contractInfo.abi,
10        contractInfo.bytecode,
11        args
12      );
13
14      let deploymentResult = new DeploymentResult(contractInfo.contractName);
15      contractEncoding[0]
16        .deploy(contractEncoding[1])
17        .send(formatJsonInput(accountInfo))
18        .on("error", (error: any) => {
19          reject(
20            new DeploymentError(contractInfo.contractName, error.toString())
21          );
22        })
23        .on("receipt", (receipt: any) => {
24          deploymentResult.gasCost = receipt.gasUsed;
25          deploymentResult.transactionHash = receipt.transactionHash;
26        })
27        .then((contractInstance: any) => {
28          deploymentResult.contractAddress = contractInstance.options.address;
29          resolve(deploymentResult);
30        })
31        .catch((error: any) => {
32          console.log(error);
33          reject(
34            new DeploymentError(contractInfo.contractName, error.toString())
35          );
36        });
37    } catch (error) {
38      reject(new DeploymentError(contractInfo.contractName, error.toString()));
39    }
40  });
41 };
```

Source Code 3: Deploy Smart Contract Function

Execute Smart Contract Function

This procedure executes the smart contract function deployed on the Hyperledger Fabric. It takes the contract address, function name, and ABI. ABI provides the binary interface to interact with the smart contract. First, Web3 SDK encodes the function information using ABI. Next, the Hyperledger adapter executes the specified function of the smart contract by sending the transaction to the contract address, refer to Line 10-16 of Source Code 4. Finally, on successful execution, it will return a consumable Promise with the data (i.e., transaction hash). Note that the execute function of the smart contract performs a set of operations and alters data in the contract.

```
1 export let execContractFunctionSync = (
2   contractAddress: string,
3   contractAbi: string,
4   functionInfo: FunctionInfo,
5   accountInfo: AccountInfo,
6   args?: Array<any>
7 ) => {
8   return new Promise<DeploymentOutput>((resolve, reject) => {
9     let encodedFunction = encodeFunctionCall(functionInfo, contractAbi, args);
10    web3.eth
11      .sendTransaction({
12        from: accountInfo.from,
13        to: contractAddress,
14        data: encodedFunction,
15        gas: accountInfo.gas,
16      })
17      .then((receipt) => {
18        resolve(
19          new DeploymentResult(
20            functionInfo.functionName,
21            receipt.transactionHash,
22            contractAddress,
23            receipt.gasUsed
24          )
25        );
26      })
27      .catch((error: any) => {
28        reject(
29          new DeploymentError(functionInfo.functionName, error.toString())
30        );
31      });
32  });
33 };
```

Source Code 4: Execute Smart Contract Function

Call Contract Function

Theoretically speaking, this method is quite similar to executing smart contract functions. However, the difference is that "call functions" only return data from the smart contract without performing any operations (getter functions). So, it takes the same inputs as executive functions and encodes the function information using ABI and Web3 SDK. Next, the Hyperledger adapter invokes the specified function of the smart contract using "web3.eth.call", refer to Line 11-23 of Source Code 5. Finally, it will return a consumable Promise with the data (i.e., decode output result) on successful execution.

```
1 export let callContractFunction = (
2   contractAddress: string,
3   contractAbi: string,
4   functionInfo: FunctionInfo,
5   accountInfo: AccountInfo,
6   args?: Array<any>
7 ) => {
8   return new Promise<any>((resolve, reject) => {
9     try {
10      let encodedFunction = encodeFunctionCall(functionInfo, contractAbi, args);
11      web3.eth
12        .call({
13          to: contractAddress,
14          data: encodedFunction,
15          gas: accountInfo.gas,
16        })
17        .then((callResult: string) => {
18          resolve(
19            functionInfo.fullInfo
20              ? decodeParametersFull(functionInfo.returnType, callResult)
21              : decodeParameter(functionInfo.returnType, callResult)
22          );
23        })
24        .catch((error: any) => {
25          reject(
26            new PromiseError(
27              `Error calling function ${functionInfo.functionName}`,
28              error,
29              [new Component("hyperledger-adapter", "callContractFunction")]
30            )
31          );
32        });
33    } catch (error) {
34      reject(
35        new PromiseError(
36          `Error Encoding function ${functionInfo.functionName}`,
37          error,
38          [new Component("hyperledger-adapter", "callContractFunction")]
39        )
40      );
41    }
42  });
43 };
```

Source Code 5: Call Smart Contract Function

Get Event Info From Logs

As discussed in Section 5.2, Event Monitor component of Off-Chain Runtime in Caterpillar listens for events triggering on the blockchain. To provide this functionality, the Hyperledger adapter implements a function that takes ABI, contract address and event information and checks logs if the provided event has happened. In case an event has been triggered, this function gets the associated transaction data. The developed code for this function can be referred to in the Source Code 6.

```
1 export let subscribeToLog = async (
2   contractAddress: string,
3   contractAbi: string,
4   eventInfo: FunctionInfo,
5   functionCallback: any
6 ) => {
7   try {
8     web3.eth.getPastLogs({
9       address: contractAddress,
10      topics: [encodeEventFromAbi(eventInfo, contractAbi)],
11    }).then((results) => {
12      if (results && results.length) {
13        const result = results[0];
14        web3.eth
15          .getTransactionReceipt(result.transactionHash)
16          .then((transactionInfo) => {
17            functionCallback(
18              result.transactionHash,
19              transactionInfo.gasUsed,
20              decodeEventLogFromAbi(eventInfo, contractAbi, result.data)
21            );
22          });
23      }
24    })
25  } catch (error) {
26    print("ERROR IN HYPERLEDGER ADAPTER", TypeMessage.error);
27    print(error, TypeMessage.error);
28    printSeparator();
29  }
30 };
```

Source Code 6: Get Event information from Hyperledger logs

Get Transaction Info

As the name suggests, this function provides the information and data about the transaction on the blockchain network. It takes the transaction hash as input and retrieves the details using "web3.eth.getTransactionReceipt" method of Web3 SDK. The developed code for this function can be referred to in the Source Code 7.

```
1 export let getTransactionInfo = (transactionHash: string) => {
2   return new Promise((resolve, reject) => {
3     try {
4       web3.eth
5         .getTransactionReceipt(transactionHash)
6         .then((transactionInfo: any) => {
7           if (transactionInfo) {
8             resolve(transactionInfo);
9           } else {
10            throw new Error();
11          }
12        });
13    } catch(e) {
14      reject();
15    }
16  })
17 };
```

Source Code 7: Get Transaction Info

Utility functions

In addition to core functions, Hyperledger adapter needs to implement utility function like "*encodeSmartContract*", "*encodeFunctionCall*", "*decodeParametersFull*", and etcetera. These functions help in encoding the information before sending it to the blockchain network and decoding the response from the network. The list of utility functions are summarised as follows:

- The "*encodeSmartContract*" function instantiates the contract object from smart contract ABI.
- The "*encodeFunctionCall"* function takes function information (i.e., name, input parameters types, input parameters values, and output parameters types) and contract ABI as input and encode them using "*web3.eth.abi.encodeFunctionCall" method*.
- The "decodeParameter" and "*decodeParametersFull"* functions decode the values from HEX to UTF-8 format.

After the implementation of the Hyperledger adapter, the solution to handle multiple Fab3 instances will be implemented.

6.5.2 Fab3 Middleware Implementation

Fab3 proxy instances need to be run for all users who want to interact with Hyperledger Fabric EVMCC. To let the Hyperledger adapter know which user is invoking the request, the "*setProvider*" function needs to be called to set the FAB3-URL, which is provided as a header to all the REST API calls for the Fabric network. As Caterpillar API is built using the ExpressJS framework, a middleware (*validateFab3UrlHeader*) is added, which will check if the FAB3-URL header is available. In case this header is not present, it will return an error response. Source Code 8 and Source Code 9 show the validation middleware and example usage, respectively.

- **Validate Fab3 URL middleware**

```
1 import { Request, Response, NextFunction} from "express";
2
3 const validateFab3UrlHeader = (
4   request: Request,
5   response: Response,
6   next: NextFunction
7 ) => {
8   const fab3Url = request.header("FAB3-URL");
9   if (!fab3Url) {
10    return response.status(400).json({
11      message: "FAB3-URL is missing from request header"
12    });
13  }
14  next();
15 }
16
17 export default validateFab3UrlHeader;
```

Source Code 8: Validate Fab3 URL Middleware

- **Example Route**

```
1 // Runtime Registry Route (Hyperledger Fabric)
2 router.post(
3     '/hyperledger/registries',
4     validateFab3UrlHeader,
5     registryHyperledgerCtrl.deployRuntimeRegistry
6 );
```

Source Code 9: Example Middleware Usage

After validating the FAB3-URL header, the "*setProvider*" function of the Hyperledger adapter will be called, which will set the provider URL to the Web3 SDK. The resulting code is shown in Source Code 10.

```
1 import { Request } from "express";
2
3 export const setWeb3Provider = (request: Request, adapter) => {
4   const fab3Url = request.header("FAB3-URL");
5   adapter.setProvider(fab3Url);
6 }
```

Source Code 10: Set Web3 Provider in Middleware

At this point, all the important implementation for the Hyperledger adapter is completed. The next step is to refactor all the routes, services, and controllers to support both the Ethereum and Hyperledger adapters in Caterpillar BPMS.

6.5.3 Hyperledger Adapter Integration

Caterpillar implements REST API to interact with its core functionalities. To make these functionalities compatible with the Hyperledger adapter, routes, controllers, and services need to be refactored so that this will inject the adapter as a dependency. An example implementation is listed below.

- **Common Controller**

```
1 function ProcessInstancesCtrl(adapter) {
2   const runtimeRegistryService = RegistryService(adapter);
3   const executionService = ExecutionService(adapter);
4   const eventMonitor = EventMonitor(adapter);
5   let runtimeRegistry: ContractInfo;
```

Source Code 11: Caterpillar Example Common Controller

- **Ethereum Specific Controller**

```
1 const commonHandlers = ProcessInstancesCtrl(ethereumAdapter);
2
3 export let createNewProcessInstance = (
4   request: Request,
5   response: Response
6 ) => {
7   commonHandlers.createNewProcessInstanceHandler(request, response);
8 };
```

Source Code 12: Caterpillar Handling Ethereum Adapter

- **Hyperledger Specific Controller**

```
1 const commonHandlers = ProcessInstancesCtrl(hyperledgerAdapter);
2
3 export let createNewProcessInstance = (
4   request: Request,
5   response: Response
6 ) => {
7   setWeb3Provider(request, hyperledgerAdapter);
8   commonHandlers.createNewProcessInstanceHandler(request, response);
9 };
```

Source Code 13: Caterpillar Handling Hyperledger Adapter

In the above example, a "*common controller*" is created with all core functionalities, which takes the *adapter* as a dependency. After that, separate controllers for Ethereum and Hyperledger Fabric create an instance of a "*common controller*" that injects Ethereum and Hyperledger adapter as a dependency, respectively.

The resulting REST API after integration of the Hyperledger adapter into Caterpillar BPMS are represented in Table 1, Table 2, Table 3, and Table 4.

6.6 Evaluation

Since the proposed work is a software solution, the criteria being selected to determine whether the goal has been achieved are also defined by the commonly used metrics in software development. Hence, the criteria for the valuation of the “Hyperledger adapter” is as follow:

- Unit Testing
- Integration Testing

6.6.1 Evaluation Setup

In order to evaluate the developed Hyperledger adapter integrated into Caterpillar BPMS, this section sets up the environment to perform Unit and Integration tests. A required set of steps for performing the evaluation are as follows:

- Hyperledger Fabric release 1.3 was installed.
- Customised Sample "first-network" was configured and started.
- EVM Chaincode was configured and installed on the network mentioned above.
- For evaluation purposes, two Fab3 proxy instances were set up for different users on different PORTs.
- Caterpillar's environment was set up with its required tools.

Note that the evaluation process was carried out on Mac OSX 10.15.7 with 8GB of memory. The setup mentioned above is common for both Unit and Integration tests. Appendix II provides the repository for this thesis contribution, which includes “hyperledger-documentation.pdf” for detailed technical documentation to reproduce this environment.

6.6.2 Unit Testing

Unit testing[13] is a software development process in which the smallest testable parts of the application are individually evaluated. Since the Hyperledger adapter is responsible for interacting with the Fabric network, unit tests to scrutinize deploying a smart contract, calling smart contract function, and executing smart contract functions are being created.

For testing purposes, an example “*SimpleStorage.sol*” smart contract has been used, which has one *uint256* typed variable and three functions to save and retrieve the value (i.e., set, multiply, get). To test the Hyperledger adapter, the following unit tests have been written:

a) deploySmartContractSync

This unit test checks if the smart contract is deployed successfully or not. The success criteria must satisfy the following conditions:

- Smart contract deployed successfully
- Output result shouldn’t be undefined
- Output result shouldn’t be null
- Output result should be of an object type
- Output result object must have *contractAddress* as a string value

b) callContractFunction

The implementation of the *callContractFunction* unit test should satisfy the following conditions:

- *get* function of smart contract should be called
- Output result should be defined

[13] https://en.wikipedia.org/wiki/Unit_testing

- Output result should be 0. (It is by default)

c) **execContractFunctionSync**
This unit test will check if a provided function is successfully executed or not. The success criteria must meet the following conditions:
  - *set* function of the smart contract is called
  - Output result should be defined
  - Output result should be of an object type
  - Output result object must have *transactionHash* as a string value

d) **testCases**
Based on the above-mentioned unit tests, some other test cases have been implemented as well, where *set* and *multiply* functions of the smart contract are executed, and then the output of *get* function is compared with the expected value.

As unit tests are implemented in Jest, running them is straightforward and can be achieved using the "*npm run test*" command. All unit tests were successful during our testing, as shown in Figure 24.

```
caterpillar-core — jazibsawar@Jazibs-MBP — ..erpillar-core — -zsh — 122×37
Last login: Sat May  7 02:40:17 on ttys001
❯ cd Projects/thesis/Caterpillar/caterpillar-full\ \(REST\ API\ -\ backend\)/caterpillar-core
❯ npm run test

> node-express-typescript@0.0.0 test
> jest --verbose

 PASS  __tests__/contractDeployment.test.js (11.482 s)
  deploySmartContractSync
    ✓ Deploy SimpleStorage contract (2413 ms)

 PASS  __tests__/callAndExecute.test.js (15.616 s)
  callContractFunction
    ✓ Default value should be zero (443 ms)
  execContractFunctionSync
    ✓ Set x value in smart contract to 1 (3814 ms)

 PASS  __tests__/testCases.test.js (22.117 s)
  test set & get function
    ✓ Default value should be zero (340 ms)
    ✓ set(1) (3171 ms)
    ✓ get() should be 1 (1363 ms)
    ✓ set(100) (2689 ms)
    ✓ get() should be 100 (17 ms)
  test multiply & get function
    ✓ multiply(5) (2624 ms)
    ✓ get() should be 100 * 5 = 500 (360 ms)

A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to im
proper teardown. Try running with --detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref
() was called on them.
Test Suites: 3 passed, 3 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        23.195 s
Ran all test suites.
```

Figure 24: Unit tests output

6.6.3 Integration Testing

Integration Testing[14] is a type of testing in which different modules or components of a software application are tested as a combined solution. As the work produced in this thesis is the execution of Solidity based smart contracts generated from the BPMN model in Caterpillar on the Hyperledger Fabric network, all the components (Hyperledger adapter, Caterpillar, Fab3, EVM Chaincode, and Fabric network) need to be tested in an integrated way. However, testing will be limited to checking whether generated Solidity smart contract

[14] https://en.wikipedia.org/wiki/Integration_testing

can be run on Hyperledger Fabric or not. It will not check if generated smart contracts from the BPMN model are correct and bug-free because it is out of the scope of this thesis.

To evaluate the integration of the different components, integration tests are divided into two categories which are as follows:

- Deployments
- Invocation

For testing purposes, the "Invoice Handling" BPMN model is being used, as shown in Figure 25. All these integration tests will be carried out using extended Caterpillar's REST API, as discussed in Section 5.2.1.

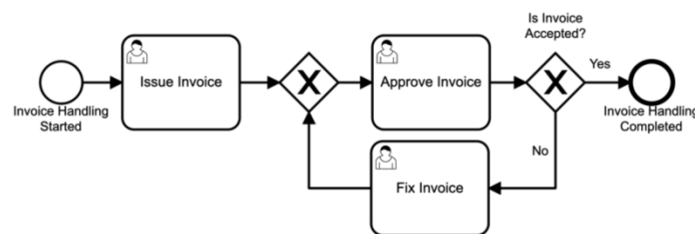

Figure 25: Invoice Handling BPMN model

Deployments

In the Deployment category, the components that are being tested are as follows:

- Deploy Runtime Registry
- Deploy Compilation-based Model
- Deploy Interpretation-based Model
- Dynamic Access Control
  - o Deploy Role-based Policy
  - o Deploy Task Role Map
  - o Deploy Access Control

Runtime Register is a prerequisite to deploying model and dynamic access control. Hence, first, the "*Runtime Registry*" is deployed, and then the rest of the models and dynamic access control will be deployed using that registry. Integration tests carried out are as follow:

1. Deploy Runtime Registry
   a. **URI**: HTTP POST "*hyperledger/registries*"
   b. **Output Parameters**: *ID* (Id of runtime registry in "Process Repository")
2. Retrieve deployed Runtime Registry's *contractAddress* on the blockchain using the ID from the last step. This *contractAddress* will be used in the upcoming tests.
   a. **URI**: HTTP GET "*hyperledger/registries/:registryId*"
   b. **Output Parameters**: *contractName, abi, bytecode, solidityCode, address* (This is *contractAddress* on the blockchain)
3. Deploy a Compilation-based model by providing the registry's contract address
   a. **URI**: HTTP POST "*hyperledger/models*"

   b. **Input Parameters**: *name*, *registeryAddress, bpmn* (XML format)
   c. **Output Parameters**: *bundleId.* This *bundleId* can be used to query metadata and artefacts of the model.
4. Deploy the Interpretation-based model by providing the registry's contract address
   a. **URI**: HTTP POST "*hyperledger/interpreter/models"*
   b. **Input Parameters**: *registeryAddress, bpmn* (XML format)
   c. **Output Parameters**: *transactionHashes* (i.e., IFLOW, Interpreter, and factory)
5. Deploy Role based binding policy
   a. **URI**: HTTP POST "*hyperledger/rb-policy"*
   b. **Input Parameters**: *policy* (Policy used for testing is shown in Figure 26)
   c. **Output Parameters**: *transactionHash*
6. Deploy Task role map
   a. **URI**: HTTP POST "*hyperledger/task-role-map"*
   b. **Input Parameters**: *contractName, registryAddress, roleTaskPairs* (i.e., "*[{\"taskIndex\": 1, \"roleIndex\": 1}]*")
   c. **Output Parameters**: *transactionHash*
7. Finally, Access control will be deployed
   a. **URI**: HTTP POST "*hyperledger/access-control"*
   b. **Input Parameters**: *registryAddress*
   c. **Output Parameters**: *transactionHash*

```
{
    "policy": "{ Invoicer is case-creator; Invoicer nominates Invoicee }{}",
    "registryAddress": ""
}
```

Figure 26: Role-based binding policy

Integration tests of all the above-mentioned components in the deployments category have been passed successfully. These integration tests can be run using the "*npm run itest:deployments*" command. Figure 27 shows the related output.

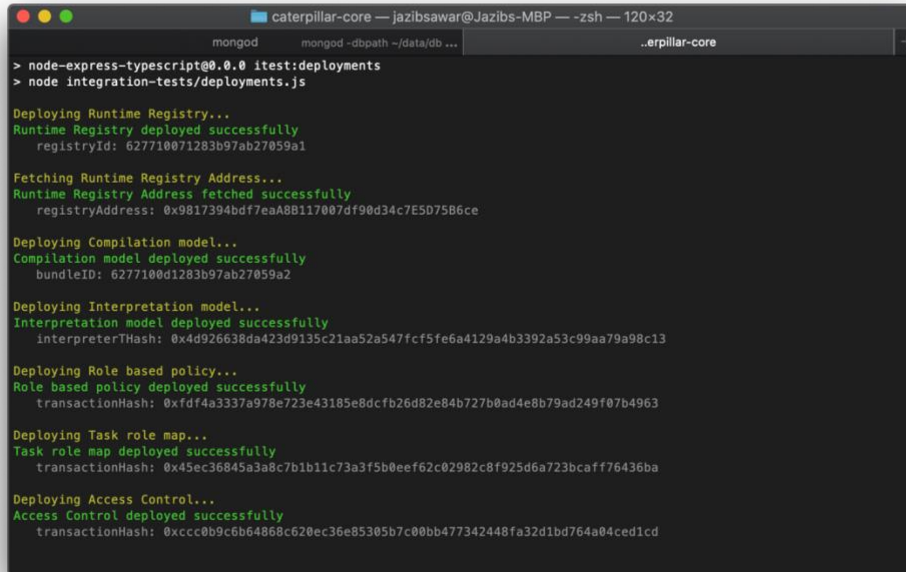

Figure 27: Deployments integration tests output

Invocation

Previously, deployments of Runtime Registry, BPMN models (compilation-based, interpretation-based), and dynamic access control were successfully evaluated. Next, the invocation of smart contracts by Caterpillar will be evaluated, which will conclude our integration testing.

These tests involve creating a process instance for an interpretation-based model, nominate a user as a case-creator, nominate a user with a role, and checking role states before and after performing these operations. Note that Runtime Register, Interpretation-based model, Role-based policy, Task role map, and access control are required before invoking these operations, which were deployed in the Deployment phase of integration testing. Steps carried out in Invocation integration tests are as follow:

1. **Retrieve IFLOW of Deployed Interpretation-based model**
   First, the ID of the interpretation-based model deployed is fetched. To achieve this, a HTTP GET request on the URL "*hyperledger/interpreter/models*" with *registryAddress* as HEADER is sent, which will return IDs of models linked in Runtime Registry. Next, the IFLOW address will be extracted from the metadata of the model by sending another HTTP GET request on the URL "*/hyperledger/interpreter/models/:mHash*". This request is made with the ID as input, which we got in the previous request.
2. **Create a new Process Case**
   After extracting the IFLOW address, a new process case will be created by sending a HTTP POST request on the URL "*hyperledger/i-flow/:iFlowAddress/i-data*". This request will be made by providing *registryAddress*, *rbPolicyAddr*, *taskRoleMapAddr*, and *taskRoleMapAddr* as JSON data and "*IFLOW address*" as path parameter. On success, this request will output a process case address (*pcAddress*), shown in Figure 28.

```
POST /hyperledger/i-flow/0xa6393f6D732174100Cf8CB25c1dA046E68252c3e/i-data 202 2620.334 ms - 88
SUCCESS: New Process Instance created from IFlow running at 0xf58D86218a3076bcC3981ddb18c82583Db496f21
  TransactionHash: 0x3cac6fd0bddf21ea36866df323676a49223b4613592048e29e2ee53d5e0aafdc
  Address: 0xd147E96742ACd15f5C70d6CDA1D20b127b8F1E4C
  GasUsed: 0 units
```

Figure 28: Process Case Request Output

3. **Nominate a User with a case-creator Role**

   Figure 26 indicates the role-based binding policy used in our testing. It has two roles (i.e., Invoicer and Invoicee). Since the "Invoicer" role is not yet assigned, the role state should be "UNBOUND". Role state can be evaluated at any point by sending an HTTP GET request on URL *"/hyperledger/rb-opertation/:pCase/state*" with *pcAddress* as path parameter and "*role=Invoicer*" as request HEADER. Figure 29 shows the intended "UNBOUND" state.

```
1 {
2     "role": "Invoicer",
3     "state": "UNBOUND"
4 }
```

Figure 29: UNBOUND Role state

   Now, a HTTP PATCH request is sent on URL *"/hyperledger/rb-opertation/:pCase/nominate-creator*" to assign a case-creator "Invoicer" role to a user (User address on the blockchain). This request is made with "rNominee=*Invoicer*" and "*nominee=<user_address>"* as JSON body. Afterwards, the role state is re-checked, and it is stated as "BOUND", as shown in Figure 30.

```
1 {
2     "role": "Invoicer",
3     "state": "BOUND"
4 }
```

Figure 30: BOUND Role state

4. **Nominate a User with a role**

   Finally, the nomination process of a user with the "Invoicee" role will be evaluated, which is carried out by the case-creator ("Invoicer") user. Before nomination, evaluation of the "Invoicee" role state is "UNBOUND". To nominate a user with a user, a HTTP PATCH request is sent on the URL "*/hyperledger/rb-opertation/:pCase/nominate*" with the request body as a JSON object which includes the following properties as shown in Figure 31.

```
{
    "rNominator": "Invoicer",
    "nominator": "<invoicer_user_address>",
    "rNominee": "Invoicee",
    "nominee": "<new_invoicee_user_address",
    "registryAddress": "<registry_address"
}
```

Figure 31: Nominate Example Request

After the completion of the nominate request, the “Invoicee” role state is “BOUND”.

In this section, we successfully performed integration testing by deploying Runtime Registry, BPMN models and Dynamic Access Control. Additionally, we also invoked some of the operations of the deployed models and policies, which internally executed functions of smart contracts using the Hyperledger adapter.

6.7 Discussion

This chapter discusses the implementation of an adapter that extends Caterpillar's capability to execute smart contracts generated from the BPMN model on the Hyperledger Fabric blockchain. Moreover, multiple Unit and Integration tests were performed to determine whether the Hyperledger adapter works and is correctly integrated into the Caterpillar BPMS prototype.

Previously, Caterpillar only supported the execution of business processes in the form of smart contracts on the Ethereum blockchain. Ethereum being a permissionless blockchain, provides decentralisation and security. However, organisations do not control who is joining the network due to its public nature, which causes a considerable data privacy risk for them. Additionally, due to the size of the network and the existence of a proof-of-work consensus mechanism, transactions' speed is usually slow.

With the integration of Hyperledger Fabric, Caterpillar can execute business processes on permissioned blockchain as well. Now, users and organisations have different options of blockchains available to choose from based on their requirements. Hyperledger Fabric brings the characteristics (i.e., high throughput, membership, data privacy) of permissioned blockchain to Caterpillar. As only allowed members can join the network, the number of nodes will be relatively quite less. This will enable organisations to perform their blockchain transactions faster and have greater data privacy.

6.7.1 Limitations

Ethereum and newly integrated Hyperledger based solutions are pretty similar in their workflow. Both support Solidity smart contracts. However, the Hyperledger Fabric network has some limitations, which are as follows:

- A separate Fab3 proxy instance needs to be running for each peer/identity, making it harder to develop dApps.
- EVM Chaincode and Fab3 proxy have not fully implemented all the functionalities of Ethereum's Solidity smart contracts. For instance, Fab3 does not support listening to the events happening in the smart contracts, subscribing to the transactions, and subscribing to the logs.

In addition to these limitations, Ethereum's Solidity-based smart contract feature of Hyperledger Fabric has very minimal developer support.

7 Conclusion and Future work

In this thesis, the Hyperledger Fabric blockchain has been integrated into Caterpillar to execute collaborative business processes on permissioned blockchain. The contributions of this thesis are three folded. Firstly, EVM Chaincode and Fab3 proxy has been configured into the Hyperledger Fabric to support the execution of Solidity smart contract because Caterpillar generates smart contracts from the BPMN model in Solidity language. Secondly, a Hyperledger adapter has been developed, providing functionalities to deploy smart contracts, execute smart contract functions, and monitor transactions and logs on the Fabric network using Fab3 proxy. Finally, the Hyperledger adapter has been integrated into Caterpillar to create an interface between Off-Chain and On-Chain components of the Caterpillar BPMS. This integration enables Caterpillar to deploy business process models and then interact with them on permissioned blockchain, Hyperledger Fabric. The key advantages of Hyperledger Fabric integration over the already existing Ethereum based solution are summarized as follows:

- Organisations can have complete control over their blockchain network. They can decide who can join or what type of permissions members can have.
- Due to the limited number of nodes and ability to implement a custom consensus mechanism, the network will have higher throughput resulting in higher transaction speeds.
- Being a private permissioned blockchain, Hyperledger Fabric can provide higher data privacy.

7.1 Future Work

The contributions described in the thesis open up some possibilities for future work, which are discussed below.

In this thesis, we integrated Hyperledger Fabric blockchain in Caterpillar to execute business processes translated to Solidity smart contracts. However, Hyperledger Ledger support for Solidity smart contacts is limited and has some limitations. For instance, the Fab3 proxy service, which is responsible for providing Ethereum like JSON-RPC API, does not support listening to events, logs, and transactions. For that reason, currently, the developed Hyperledger adapter only supports synchronous transactions to the blockchain. Asynchronous behaviour could be achieved by developing an event pooling service on Hyperledger Fabric. Investigating this problem can be considered as the future scope of the work.

Finally, blockchain is a rapidly developing technology, and it is improving day by day. So, investigating Caterpillar's performance on different blockchain platforms and integration can be a direction for future work.

8 References

[1] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of business process management*. Springer, 2013.

[2] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, "Business Process Management: A Survey," in *Business Process Management*, Berlin, Heidelberg, A. ter Hofstede, W. M. P. van der Aalst, and M. Weske, Eds., 2003// 2003: Springer Berlin Heidelberg, pp. 1-12.

[3] C. Di Ciccio *et al.*, "Blockchain support for collaborative business processes," vol. 42, no. 3, pp. 182-190, 2019.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," p. 21260, 2008.

[5] M. H. Miraz and M. Ali, "Applications of blockchain technology beyond cryptocurrency," 2018.

[6] J. Mendling *et al.*, "Blockchains for Business Process Management - Challenges and Opportunities," *ACM Trans. Manage. Inf. Syst.,* vol. 9, no. 1, p. Article 4, 2018, doi: 10.1145/3183367.

[7] M. Wohrer and U. Zdun, "Smart contracts: security patterns in the ethereum ecosystem and solidity," in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018: IEEE, pp. 2-8.

[8] O. López Pintado, "Collaborative business process execution on the blockchain: the caterpillar system," University of Tartu, 2020. [Online]. Available: https://dspace.ut.ee/handle/10062/68766

[9] M. Dabbagh, M. Kakavand, M. Tahir, and A. Amphawan, "Performance Analysis of Blockchain Platforms: Empirical Evaluation of Hyperledger Fabric and Ethereum," in *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, 26-27 Sept. 2020 2020, pp. 1-6, doi: 10.1109/IICAIET49801.2020.9257811.

[10] K. Wüst and A. Gervais, "Do you need a Blockchain?, Crypto Valley Conference on Blockchain Technology (CVCBT)," ed: IEEE, 2018.

[11] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, H. J. I. J. o. W. Wang, and G. Services, "Blockchain challenges and opportunities: A survey," vol. 14, no. 4, pp. 352-375, 2018.

[12] J. Polge, J. Robert, and Y. J. I. E. Le Traon, "Permissioned blockchain frameworks in the industry: A comparison," vol. 7, no. 2, pp. 229-233, 2021.

[13] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on distributed cryptocurrencies and consensus ledgers*, 2016, vol. 310, no. 4: Chicago, IL, pp. 1-4.

[14] M. Valenta and P. J. F. S. B. C. Sandner, "Comparison of ethereum, hyperledger fabric and corda," vol. 8, pp. 1-8, 2017.

[15] R. Swetha. "Hyperledger Fabric Now Supports Ethereum." https://www.hyperledger.org/blog/2018/10/26/hyperledger-fabric-now-supports-ethereum (accessed August 12, 2021, 2021).

[16] M. Indulska, P. Green, J. Recker, and M. Rosemann, *Business Process Modeling: Perceived Benefits*. 2009.

[17] C. Liu, Q. Li, and X. J. I. S. F. Zhao, "Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue," vol. 11, no. 3, pp. 201-209, 2009.

[18] Q. Mo, W. Song, F. Dai, L. Lin, and T. J. I. T. o. S. C. Li, "Development of collaborative business processes: a correctness enforcement approach," 2019.

[19] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, A. J. S. P. Ponomarev, and Experience, "Caterpillar: a business process execution engine on the Ethereum blockchain," vol. 49, no. 7, pp. 1162-1193, 2019.

[20] X. Xu, I. Weber, and M. Staples, *Architecture for blockchain applications*. Springer, 2019.

[21] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where Is Current Research on Blockchain Technology?—A Systematic Review," *PLOS ONE,* vol. 11, 10/03 2016, doi: 10.1371/journal.pone.0163477.

[22] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," vol. 2, no. 6-10, p. 71, 2016.

[23] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," ed, 2016.

[24] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254-269.

[25] F. Idelberger, G. Governatori, R. Riveret, and G. Sartor, *Evaluation of Logic-Based Smart Contracts for Blockchain Systems*. 2016.

[26] C. Sillaber and B. J. D. u. D.-D. Waltl, "Life cycle of smart contracts in blockchain ecosystems," vol. 41, no. 8, pp. 497-500, 2017.

[27] R. J. S. Koulu, "Blockchains and online dispute resolution: smart contracts as an alternative to enforcement," vol. 13, p. 40, 2016.

[28] Z. Zheng *et al.*, "An overview on smart contracts: Challenges, advances and platforms," vol. 105, pp. 475-491, 2020.

[29] F. Jake. "Consensus Mechanism (Cryptocurrency)." https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp (accessed 2021).

[30] J. Garay, A. Kiayias, and N. Leonardos, *The Bitcoin Backbone Protocol with Chains of Variable Difficulty*. 2017, pp. 291-323.

[31] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual international cryptology conference*, 2017: Springer, pp. 357-388.

[32] X. Xu *et al.*, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE international conference on software architecture (ICSA)*, 2017: IEEE, pp. 243-252.

[33] S. Franzoni, "Blockchain and smart contracts in the Fashion industry," Politecnico di Torino, 2020.

[34] V. J. w. p. Buterin, "A next-generation smart contract and decentralized application platform," vol. 3, no. 37, p. 2.1, 2014.

[35] F. Ma *et al.*, "Security reinforcement for ethereum virtual machine," vol. 58, no. 4, p. 102565, 2021.

[36] W. Hu, Z. Fan, and Y. J. I. P. Gao, "Research on smart contract optimization method on blockchain," vol. 21, no. 5, pp. 33-38, 2019.

[37] D. Vujičić, D. Jagodić, and S. Ranđić, "Blockchain technology, bitcoin, and Ethereum: A brief overview," in *2018 17th international symposium infoteh-jahorina (infoteh)*, 2018: IEEE, pp. 1-6.

[38] D. D. Wood, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER," 2014.

[39] Y. Hirai, *Defining the Ethereum Virtual Machine for Interactive Theorem Provers*. 2017, pp. 520-535.

[40] R. Taş and T. Ö. Ö, "Building A Decentralized Application on the Ethereum Blockchain," in *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 11-13 Oct. 2019 2019, pp. 1-4, doi: 10.1109/ISMSIT.2019.8932806.

[41] P. Hartel and M. Staalduinen, *Truffle tests for free -- Replaying Ethereum smart contracts for transparency*. 2019.

[42] R. M. Amir Latif, K. Hussain, N. Z. Jhanjhi, A. Nayyar, and O. Rizwan, "A remix IDE: smart contract-based framework for the healthcare sector by using Blockchain technology," *Multimedia Tools and Applications,* 2020/11/10 2020, doi: 10.1007/s11042-020-10087-1.

[43] S. Panda and S. Satapathy, "An Investigation into Smart Contract Deployment on Ethereum Platform Using Web3.js and Solidity Using Blockchain," 2021, pp. 549-561.

[44] S. Aggarwal and N. Kumar, "Hyperledger," in *Advances in Computers*, vol. 121: Elsevier, 2021, pp. 323-343.

[45] M. A. Zafar, F. Sher, M. U. Janjua, and S. Baset, "Sol2js: translating solidity contracts into javascript for hyperledger fabric," in *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, 2018, pp. 19-24.

[46] M. Krstić and L. Krstić, "Hyperledger frameworks with a special focus on Hyperledger Fabric," *Vojnotehnicki glasnik,* vol. 68, pp. 639-663, 07/01 2020, doi: 10.5937/vojtehg68-26206.

[47] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, *FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second*. 2019, pp. 455-463.

[48] "Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2019 International Workshops, DPM 2019 and CBT 2019, Luxembourg, September 26–27, 2019, Proceedings," Luxembourg, Luxembourg, 2019: Springer-Verlag.

[49] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," in

International conference on business process management, 2016: Springer, pp. 329-347.

[50] C. Prybila, S. Schulte, C. Hochreiner, and I. J. F. G. C. S. Weber, "Runtime verification for business processes utilizing the Bitcoin blockchain," vol. 107, pp. 816-831, 2020.

[51] L. García-Bañuelos, A. Ponomarev, M. Dumas, and I. Weber, "Optimized execution of business processes on blockchain," in *International conference on business process management*, 2017: Springer, pp. 130-146.

[52] V. Pourheidari, S. Rouhani, and R. Deters, *A Case Study of Execution of Untrusted Business Process on Permissioned Blockchain*. 2019.

[53] S. Punathumkandi, V. M. Sundaram, and P. J. S. Panneer, "Interoperable Permissioned-Blockchain with Sustainable Performance," vol. 13, no. 20, p. 11132, 2021.

[54] "Hyperledger Fabric Docs." https://hyperledger-fabric.readthedocs.io/en/release-1.3/ (accessed 2022).

[55] M. Mamun. "How does Hyperledger Fabric work?" https://medium.com/coinmonks/how-does-hyperledger-fabric-works-cdb68e6066f5 (accessed 2022).

[56] "Hyperledger Architecture, Volume 1." www.hyperledger.org. https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf (accessed 2022).

[57] "Hyperledger Fabric EVM chaincode." https://github.com/hyperledger/fabric-chaincode-evm (accessed 2022).

Appendix

I. Abbreviations

BPMN: Business Process Model and Notation

BPMS: Business Process Management System

BPM: Business Process Management

IT: Information Technology

CA: Certificate Authority

MSP: Membership Service Provider

EOA: Externally Owned Accounts

EVM: Ethereum Virtual Machine

REST: Representational State Transfer

API: Application Programming Interface

SDK: Software Development Kit

CLI: Command Line Interface

SSH: Secure Shell Protocol

EVMCC: EVM Chaincode

ABI: Application Binary Interface

II. Repository

The source code of this thesis contribution can be downloaded from https://github.com/jazibsawar/Caterpillar. The contribution was carried out in the "*Caterpillar-full/caterpillar-core*" folder. The list of work produced in this repository is as follows:

- "*__tests__*" directory provides all the Unit Tests performed.
- "*integration-tests*" directory includes Integration tests (partially).
- "*src*" directory includes the Hyperledger adapter and its integration contribution.
- "*hyperledger-scripts*" directly include all the scripts produced or modified that are necessary to configure Hyperledger Fabric and associated tools.
- The "*hyperledger-documetation.pdf*" file provides all the technical documentation and steps to reproduce the environment.

III. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Jazib Sawar,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, “Blockchain-based business process execution on Hyperledger”, supervised by Marlon Dumas and Orlenys Lopez Pintado.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons’ intellectual property rights or rights arising from the personal data protection legislation.

Jazib Sawar

17/05/2022