UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Germo Hünerson

Comparison of Ethereum and Corda Platforms

Bachelor's Thesis (9ECTS)

Supervisors: Luciano García-Bañuelos, PhD
Fredrik P. Milani, PhD

Tartu 2018

Comparison of Ethereum and Corda Platforms

Abstract:

This bachelor's thesis provides a comparative analysis for creating decentralized applications focusing on two platforms Ethereum and Corda. Its purpose is to provide developers deciding on a platform with information to deduce an educated opinion. It starts of by giving a brief explanation of the blockchain and distributed ledger technologies with related terminology. Then comes a more in depth view of the selected platforms with the motivation for choosing these two for the thesis. The analysis is conducted based on almost identical application developed on both of the platforms. The comparative analysis brings out the biggest differences between on main use case and architecture among others.

Keywords:

Blockchain, Decentralised applications, distributed ledger technology, Ethereum, Corda

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

Ethereumi ja Corda plokiahela platvormide võrdlev analüüs

Lühikokkuvõte:

Käesolev bakalaureusetöö annab analüütilise ülevaate detsentraalse rakenduse loomisest kahel platvormil: Ethereum ja Corda. Töö eesmärgiks on anda platvormidest ülevaade ning sellega lihtsustada arendajatel platvormi valiku otsustusprotsessi. Töö raames tutvustatakse plokiahela ja hajusraamatu tehnoloogiaid, käsitledes lähemalt valitud platforme. Analüüs põhineb nendel platformidel implementeeritud rakendustel. Sellega tuuakse välja peamised platvormide erinevused, muu hulgas peamiste kasutusjuhtude ning arhitektuuri vallas.

Võtmesõnad:

Plokiahel, detsentraalsed rakendused, hajutatud arveraamatu tehnoloogia, Ethereum, Corda

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Table of Contents

Introduction **5**

1. Background **7**
1.1 Blockchain and Distributed Ledger Technology 7
2.2 Public and Private Blockchains 9
2.3 Smart Contracts and Decentralised Applications 10

2. Case Study **12**
2.1 Pet Shop Use Case 12
2.2 Selected Platforms 12
2.2.1 Ethereum 13
2.2.1.1 Architecture 13
2.2.2 Corda 14
2.2.2.1 Architecture 15

3. Implementation **17**
3.1 Ethereum 17
3.2.1 Smart Contract and Deployment 17
3.1.2 Front-end 19
3.2 Corda 25
3.2.1 State 26
3.2.2. Contract 27
3.2.3.Flow 29
3.2.4. Local Nodes, Front-end and Interaction 31

4. Comparison **33**
4.1 Main Use Case 34
4.2 Architecture 35
4.3 Consensus 36
4.4 Language 36
4.5 Implementing Pet Shop Use Case 37

Conclusion **39**

References **40**

Licence **43**

Introduction

In the recent years there has been a huge increase in usage of blockchain technology. From Bitcoin launch in january of 2009 to more than 1500 cryptocurrencies listed on CoinMarketCap.com at the time of the writing of this thesis (July 2018). The media has covered blockchain and cryptocurrencies also quite heavily. Some of the major newspapers and magazines, for example Fortune [1] and Economist [2], are even calling it the next big and world changing thing in technology. This has created a lot of interest in topic and even the governments and public institutions are assessing the possibilities. The World Bank [3], European Parliament [4], US Department of Treasury [5], United Kingdom's [6] and Australian government [7] to name the few. Even Estonia has funded research of blockchain conducted by University of Oxford [8] and an local company Cybernetica [9] . This shows that the whole world is fascinated and involved in figuring out the best use cases for this up-and-coming technology. The widespread adoption and popularization of blockchain based technology has created many platforms which all offer similar possibilities in principle such as Ethereum, Neo, Hyperledger etc. Theses platforms offer the structure and means to companies from various industries to launch a decentralised applications (dApps). This creates a unique situation within the developer community where they need to decide on a platform to launch their applications without any or with small amount of experience. Furthermore a research paper on the state of research in blockchain technologies published in 2016 [10] stated that one of the biggest research gap within the industry is the lack of research on usability. The writers had found papers that discussed usability form user perspective but nothing from developers perspective or related to developmental aspects. They also brought to attention that most of the research is done in the Bitcoin environment and suggested that further research should be carried out to increase the knowledge outside of cryptocurrencies.

In this thesis two different platforms will be analyzed and compared. The comparison is based on implementing dApps with as similar functionality as the compared platforms allow. A application for tracking adopted pets and adoption transactions of a pet shop is taken as baseline for the analysis. The application is suggested as an example project by the team of

Truffle framework which is a development and testing tool used by Ethereum developers [11]. The analysis will cover platforms: Ethereum and Corda. The selected baseline project will help to assure the comparability of the platforms and will make the results of the analysis more relatable within the Ethereum community which currently is the biggest and most mature within the decentralised application ecosystem.

The results of this thesis should clarify the differences of the selected platforms and give suggestions based on the results of comparison and experience gained through implementing benchmark application.

The thesis will begin with familiarizing the reader with the topic, explaining the core terminology and concepts. This should give a basic understanding of distributed ledger technology and blockchain together with smart contracts and decentralised applications. The second paragraph describes the use case of the baseline application. Also, it covers the selected platforms by giving a more in depth view of the technology, architecture and platform specific terms. The third paragraph contains walkthroughs of implementing the application explaining all the necessary tools and development processes used on the platform. The fourth paragraph contains the comparison of the platforms and analysis of the implementation differences. Finally a conclusion and suggestions for further research are offered for the readers.

1. Background

The terms blockchain and distributed ledger technology are often mixed up or regarded as the same thing. This paragraph will try to define the both and explain the related terminology and concepts used in rest of the thesis.

1.1 Blockchain and Distributed Ledger Technology

A distributed ledger is in a sense a type of database that is shared over the network to all connected computers referred to as nodes. Every node replicates and saves an identical copy of the ledger. These nodes collectively maintain the database and this eliminates the need for central authority. Every node can independently construct a update to the ledger. Each update needs an acceptance from majority of the nodes which is accomplished basically via voting. Voting and reaching to agreement is called consensus and all of that is usually achieved through some kind of automated consensus algorithm. When consensus is reached in the network the ledger is updated and each node can save an updated copy of the ledger locally [12].

The concept of blockchain first surfaced in 2008 when person or a group of people under the name of Satoshi Nakamoto released a paper called “Bitcoin: A Peer-to-Peer Electronic Cash System”. The idea was to create a payment system backed by cryptographic proof where two willing parties could conduct transactions without a trusted third party. The fraudulent activity would be considered computationally impractical since any group of attackers would need more CPU power then all the honest nodes in the system combined [13]. This is the first time the double-spending problem is solved within the digital currency system without third party. This is accomplished by distributing the ledger among all the users of the system via peer-to-peer network. Every transaction that is made is saved into the public, distributed ledger. Added transactions are checked against the blockchain to make sure none of the coins have not been spent previously already. This eliminates the double-spending problem [14].

Blockchain can be considered as one of the ways to implement distributed ledger. Blockchain, as the name applies is a sequence of blocks containing complete history of

transaction records. Each block consists of two parts, the block header and block body. Block header stores the data necessary for the chain to function:

1) Parent block hash - a 256-bit hash for referencing previous block in the chain.
2) Timestamp - the time in seconds from January 1. 1970 UTC.
3) Merkel tree root hash - which is a hash value of all transactions in the block.
4) Nonce - an 4-byte field, usually it starts of as 0 and is increased by each hash calculation during consensus algorithms.
5) A block version - for stating which set of block validation rules to follow.
6) The block body holds data of all the transactions and transaction counter. The maximum number of transactions per block depends on the block size and the size of each transaction [15].

Reaching consensus among the untrustworthy nodes in peer-to-peer network is relatable to a Byzantine Generals Problem [16]. In blockchain terms this means that some form of protocol is needed to be able to add new block to the chain. This would ensure the data consistency across the all of the nodes. Some of the most common protocols and algorithms to achieve consensus are Proof of work, Proof of stake, Practical byzantine fault tolerance.

Proof of Work (PoW) is the algorithm used for example in Bitcoin and Ethereum. This means that the nodes in the network have to perform computer calculations to prove the validity and trustworthiness of the transactions. The goal of these calculations is to find a hash value for the block header. The hash value has to be equal to or smaller than the given hash value. This is accomplished by changing the nonce value in the block header which changes the outcome of hashing as well. When the requirements are met, the hash value is broadcasted to other nodes in the network for review. When the value is accepted then everybody will append the block to their chain.

Proof of Stake (PoS) is often named as energy efficient PoW algorithm. When PoW relies heavily on computing power to select the node who can add a block to the chain, then PoS uses various formulas and parameters to decide who gets to append the block. For example PPCoin adopted a concept for coin age, which basically makes it easier to become the accountant for a specific block by measuring how long the node has held onto ones coins

[17]. Other platforms like BlackCoin [18] and Nxt [19] use account balance and randomness to grant accounting rights.

Practical Byzantine Fault Tolerance (PBFT) is an algorithm for distributed systems which first reasonably efficient implementation was suggested in 1999 [20]. It basically accepts faulty responses from nodes but still proceeds with the algorithm as long as the majority approves the responses. The algorithm consists of 5 steps:

1) Request - the process starts with client sending the request to master node where the request is timestamped.
2) Pre-prepare - the master nodes allocates an order number for the request and shares the request to other nodes in the network. Other nodes have an option to decline or accept the request.
3) Prepare - when the node has accepted the request, it will send out a message to other nodes and waits message from other nodes. When the node has collected 2f+1 (f- faults) messages, in other words when the network has ⅓ of faults and ⅔ of accepted messages, it proceed to commit stage.
4) Commit - each node will send a commit messages to other nodes and when the network has collected also 2f+1 commit messages the network assumes that the nodes have came to agreement and found consensus amongst themselves. After that the node executes the instructions in the request.
5) Reply - The server replies to the client. [21]

2.2 Public and Private Blockchains

Blockchains can be divided into two major groups: public and private blockchains. The concept of the blockchain remains the same. The goal usually is to offer decentralized peer to peer network, where all participants hold copy of the ledger and every change to the ledger has to go through some sort of consensus algorithm [22].

Public blockchains like the name applies are public, which means that everybody can access them. The goal of this is to be transparent and gain trust via the publicly available data or state while offering some anonymity to the users.

Private blockchains have some authentication process where users have to prove they have permission to access the network. The data on private networks usually is not publicly available. Private networks could be used in areas that require user identities or deal with sensitive data for example healthcare and financial sectors.

2.3 Smart Contracts and Decentralised Applications

Smart contract is concepts first introduced by Nick Szabo in 1996 with the article "Smart Contracts: Building Blocks for Digital Markets". Where he pointed out that many kind of real life contractual clauses could be embedded into software. The contracts are executed when the pre-defined conditions are met [23]. Smart contracts work basically independently and controlling specific digital assets and managing them based on programmers instructions. Once smart contracts are implemented into blockchain platforms they are automatically applied until pre established final goals are met or the resources are consumed [24]. An analogy from an average web store would have a price and quantity for a product as some of the parameters in a smart contract. So a customer can purchase the product only if the he/she has sufficient funds to cover the price and and the store can only sell the product if the quantity in stock is greater than zero. In blockchain these kind of contracts can be performed automatically and with the same rules applying to everybody until some of the parameters are not met anymore.

Decentralized applications (dApps) in an essence are applications that are not controlled by single entity and usually run on peer-to-peer network. Great examples of dApps are Popcorn Time, BitTorrent and Tor. Theses kinds of decentralized applications that do not rely on blockchain are considered as traditional dApps [25].

In this thesis we are focusing on blockchain enabled dApps. Siraj Raval in the book "Decentralized Applications" has listed four features that any dApp should have to be considered dApp in the context of blockchain and also be profitable. Every dApp should:

1) Be Open Source - This requirement is suggested to gain trust among the users. For users to believe that the application is truly decentralized they should be able to see the source code.
2) Have internal currency - this point mainly covers the profitability aspect of the dApp. Since the source code is public, nothing will stop users from forking or copying it,

thus rendering any traditional monetization options worthless. When the internal currency is launched, often in the form of ICO (initial coin offering) the users and supporters can invest into the application and buy a portion from limited or scarce pool of coins. Owners and developers of the application are paid with the same currency.

3) Have decentralized consensus - this is covered with blockchain consensus algorithms of the platform and smart contracts implemented into the application.
4) Not have central point of failure - Every node in the network has to be independent.

2. Case Study

The present thesis will use one use case covering small portion of the platforms functionality yet grasping some of the core concepts to emphasize the differences. The thesis will focus mostly on the platform's architecture and concepts used for development and does not go into very detail about other aspects of the platforms such as currency aspect or minging.

2.1 Pet Shop Use Case

The Pet shop use case application is suggested as a tutorial in Truffle Frameworks website. This specific example was picked because many of the Ethereum developers should be familiar with it or should easily understand it. Yet it is comprehensive enough to cover various aspects of the platform.

The Pet shop is basically a webstore for pets. In this specific scenario pets are up for adoption with the price of zero Ether also they transaction is one sided which means that when an actor wants to adopt a pet, the actor just needs to send the message and the smart contract does the rest.

This example covers accounts, transactions, smart contracts and interacting with the application.

2.2 Selected Platforms

This bachelor's thesis will focus on differentiating between two platforms: Ethereum and Corda. Ethereum was chosen because it currently has one of the biggest following among users and developers in the blockchain scene. Furthermore Ethereum has in the thesis authors opinion single handedly popularised the development of decentralised applications. When Bitcoin sparked the interest, then Ethereum provided the means. Currently there are more than 1500 applications built on top of the Ethereum [26] which is highest among similar platforms. The Ethereum development team has said that the goal was to provide a general tools for wide range of applications and make it easy enough to gain popularity.

Corda was chosen because it is a distributed ledger technology backed by many big financial institutions and banks such as JPMorgan [27] , Bank of America and even nordic

banks like SEB and Nordea [28] . This makes it interesting because most previous blockchain networks were advertising themselves as decentralised platforms to cut the banks and other middleman out of the transaction equation.

2.2.1 Ethereum

Ethereum is a blockchain platform which was first made public in January 23, 2014 when the founder Vitalik Buterin announced it in their blog [29] .The goal of the platform is to offer alternative protocol for developing decentralised applications. This is accomplished by abstract foundational layer built by Ethereum's team which is based on blockchain with built-in Turing-complete programming language Solidity. This allows anyone to easily develop decentralised applications and smart contracts where they can define rules for ownership, transaction formats and state transition functions [30]. All the code is compiled down to Ethereum Virtual Machine and deployed to the blockchain for execution. Besides solidity there are few other options which the Ethereum team has built compilers for such as:

- Serpent - a python like language.
- LLL - a low level Lisp like language.
- Mutant - a language based on Go which is deprecated by now.

Solidity remains the most popular and best supported option from the listed languages. [31]

Ethereum currently uses Proof of Work consensus algorithm but the first version of Casper, a nickname for a much anticipated update, has been released and with it Ethereum is a step closer to transitioning to PoS algorithm [32].

2.2.1.1 Architecture

Simplicity, universality and modularity are couple of main design principles the Ethereum architecture follows [33]. This means that the Ethereum team has made the platform as easy as possible in the hopes it would achieve a widespread adoption among developers. Also, the platform is said to not focus on built-in features but provide tools with the internal scripting language to build any type of dapp.

One of the core concept in Ethereum used to define the state is called "account". Every account has an address and all the state transitions basically mean transfering a value and information between accounts. Every account consists of four fields:

1) The nounce
2) Current ether balance
3) Contract code (only if it is contract account)
4) Storage (empty by default)

As the third field already implied, there are two types of accounts: contract and externally owned. Basically external accounts are something that people manage. They are controlled by private keys and do not have any code within them. They can send messages by just creating and signing a transaction. Contract accounts on the other hand are necessary for the dapp to function. They are ran by their internal contract code, every time the contract account receives a message, the code activates. This allows them to read and write the internal storage, send messages or even create new contracts.

Another core concept is called “transaction” which in an essence is a signed data package that store a message to be sent. The data package stores among recipient address, sender's signature and the ether amount to transfer. Also data to protect the platform against denial-of-service attacks in the form of *startgas* and *gasprice*. *Startgas* is value defining the maximum number of computational steps the execution of the transaction can take. *Gasprice* indicates the fee per computational step the sender has to pay. Usually a computational step costs 1 gas, (gas stands for computational unit in Ethereum).

Smart contracts in solidity are just like classes in object-oriented programming. Each contract can contain state variables, functions, events etc.

2.2.2 Corda

Corda is a distributed ledger platform with the main goal for processing and recording financial agreements. The consortium for Corda was created in September 15, 2015 including nine of some of the biggest banks in the world including JP Morgan, Goldman Sachs and Barclays [34]. The project itself was made publicly available and open source on November 30, 2016 [35]. The objective of the the project was basically to reinvent the financial data and agreement management systems. The Bitcoin and Ethereum platforms which came before Corda, even though not really directly applicable to financial institutions, helped the team understand a new way to build distributed systems. Corda platform also supports smart contracts which are automatable or can work with human input and control. A key concept

here is that Cordas smart contracts can represent rights and obligations expressed in legal verses and could be legally enforceable [36].

2.2.2.1 Architecture

Corda network consists of five major components:

1) Nodes which are communicating over Advanced Message Queuing Protocol (AMQP) with TLS. Nodes have a relational database for data storage.
2) Permissioning service for TLS (Transport Layer Security) certificates.
3) Network mapping service, which is used for publishing node information on the network.
4) One or more notary service, which may be distributed over several nodes.
5) Zero or more oracle services. Which helps the ledger to connect to the real world by signing transactions if the facts stated in them are considered to be true.

As pointed out with the second component, corda is a permissioned blockchain network. Each party's identity is known and used when communicating with others. The access to the networks is also controlled by a doorman.

Corda uses point-to-point communication instead of global broadcast like many other blockchain platforms. Theses point-to-point communications are called "flows". Flows let the developer define who are the relevant parties in a transaction and what kind of information is needed. Like any typical real world transaction, for example leasing a car can involve negotiation with multiple steps and a legal parties. Corda's flows let the developer define all these kinds of use cases and steps. Flows can run concurrently and this allows for nodes to have many flows active at one, since some of the flows could last for days, while surviving a node restart or an upgrade. [37]

Corda uses unspent transaction model (UTXO) which means that for conducting a transaction some kind of existing state must be consumed and new state must be outputted. Tied to that is the issue of consensus. Corda uses two main methods to find consensus between parties for ledger updates. Firstly, a validity consensus, this requires that transactions are accepted by the input and output state contracts. Secondly, it checks if the transaction has every required signature. After passing this, a transaction can be considered valid, which

makes it prerequisite for every transaction, but this is not enough to verify the transaction itself. The second consensus, a already valid transaction has to achieve, is uniqueness consensus. This requires that none of the proposed transaction inputs have previously been consumed by another signed transaction. The uniqueness consensus eliminates the double spending problem on Corda network. Uniqueness consensus is enforced by the notaries. [38]

3. Implementation

This paragraph gives all the information necessary to replicate the results. It covers the tools used for writing and testing the applications and explains the process step by step.
All the applications have been written and tested on MacBook Pro 2017 running macOS Sierra version 10.12.6. MacOS built in terminal is used run all of the used commands and scripts. The implementation will focus on the back-end of the dApps - smart contracts, deployment etc.

3.1 Ethereum

This paragraph will explain what kind of tools are need for running a local dApp on the Ethereum chain and what does it consist of. The Visual Studio Code is used as the integrated development environment (IDE) with Solidity plugin.

3.2.1 Smart Contract and Deployment

To start off, we will need Truffle. Truffle is used for contract compilation, migration and deployment. For installation Node package manager(npm) was used:

```
npm install -g truffle
```

Truffle provided example project was used for the baseline which is offered in the form of Truffle Box. Installing the boilerplate code of the example requires to run:

```
truffle unbox pet-shop
```

In dedicated folder. This will download and unpack the folder structure with some of the initial files and test data. As seen on the figure 1, contracts folder contains all the smart contract files written in solidity. Migrations scripts are written in Javascript placed in migrations folder. Src catalog holds all of the front-end components from index.html to styling and javascript files. Finally in the root folder the package.json files contain npm dependency versions and truffle.js holds deployment configuration.

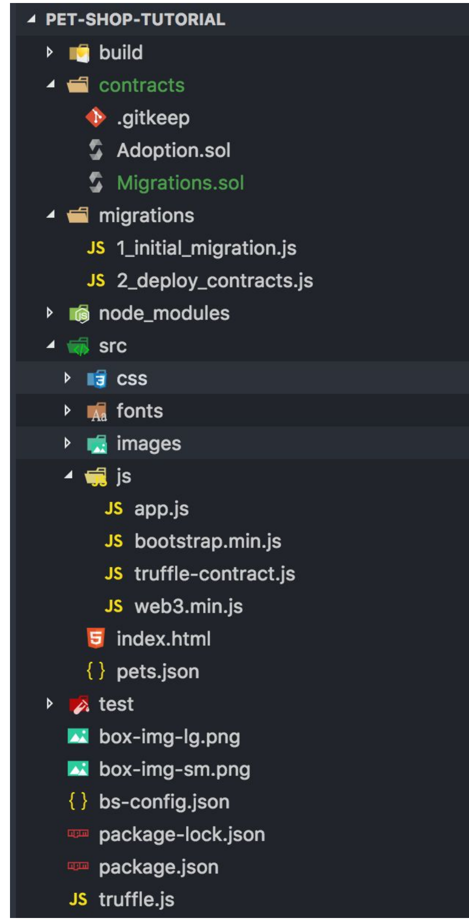

Figure 1. Finished project structure

First thing is to define smart contracts. This project needs two contracts: Migrations.sol and Adoption.sol. First one ensures that rest of the migrations are valid and none of them would be double migrated. Adoption.sol has all of the business logic in this case.

```
pragma solidity ^0.4.17;

contract Adoption {
    address[16] public adopters;


    // Adopting a pet
    function adopt (uint petId) public returns (uint){
        require(petId >= 0 && petId <= 15);
        adopters[petId] = msg.sender;

        return petId;
    }

    function getAdopters() public view returns (address[16]) {
        return adopters;
    }
}
```

Figure 2. Adoption smart contract at /contracts/Adoption.sol

As shown on figure 1 the contract creates an list of 16 addresses for adopters and has basic set and get functions. The number 16 is chosen just because the project has 16 pets within its test data. Adopt function takes the adopted petId, checks if it is in range of the list length and adds the adopters address to the list. Next the *getAdopters* returns the list.

After that the smart contracts have to be compiled to bytecode for Ethereum Virtual Machine. Truffle has a built in compiler which can be started in the project directory with command:

truffle compile

After compiling the contracts, the next step is to deploy them to blockchain. Development and testing requires a separate blockchain environment. Truffle framework offers tool named Ganache[1] for that. This sets up a local temporary Ethereum blockchain that can be used for running the project. When the local blockchain is up and running, then the contracts can be deployed. The migration scripts under migration/ directory deploy the contracts, first the Migrations contract to ensure the following ones and then the Adoption contract.

```
truffle migrate
```

Command is used for running the scripts.

3.1.2 Front-end

Now the smart contract have been applied to the blockchain. This means that our actions that fit into the contracts parameters can be recorded to the chain. To perform and handle these actions a interface is needed. The pet shop application provides the front-end which needs to be connected with the blockchain and contracts.

[1] "Ganache | Truffle Suite - Truffle Framework." https://truffleframework.com/ganache.

```html
<body>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-push-2">
        <h1 class="text-center">Pete's Pet Shop</h1>
        <hr/>
        <br/>
      </div>
    </div>

    <div id="petsRow" class="row">
      <!-- PETS LOAD HERE -->
    </div>
  </div>

  <div id="petTemplate" style="display: none;">
    <div class="col-sm-6 col-md-4 col-lg-3">
      <div class="panel panel-default panel-pet">
        <div class="panel-heading">
          <h3 class="panel-title">Scrappy</h3>
        </div>
        <div class="panel-body">
          <img alt="140x140" data-src="holder.js/140x140" class="img-rounded img-center"
           style="width: 100%;" src="https://animalso.com/wp-content/uploads/2017/01/Golden-Retriever_6.jpg"
            data-holder-rendered="true">
          <br/><br/>
          <strong>Breed</strong>: <span class="pet-breed">Golden Retriever</span><br/>
          <strong>Age</strong>: <span class="pet-age">3</span><br/>
          <strong>Location</strong>: <span class="pet-location">Warren, MI</span><br/><br/>
          <button class="btn btn-default btn-adopt" type="button" data-id="0">Adopt</button>
        </div>
      </div>
    </div>
  </div>

  <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
  <!-- Include all compiled plugins (below), or include individual files as needed -->
  <script src="js/bootstrap.min.js"></script>
  <script src="js/web3.min.js"></script>
  <script src="js/truffle-contract.js"></script>
  <script src="js/app.js"></script>
</body>
</html>
```

Figure 3. HMTL structure of the page - /src/index.html

The index.html has basic page structure and template for displaying the pet information, this page also imports javascript from app.js file. The app.js file links the front-end with the back-end by making the button on *petTemplate* div element interactable. The functionality in *btn-adopt* class is covered in figure 8. It starts off by calling *App.init* function:

```javascript
$(function () {
  $(window).load(function () {
    App.init();
  });
});
```

Figure 4. Function called on every time the page loads - /srt/js/app.js

The App variable first initializes two global variables for later usage and then defines the init function. Function *init* takes the test data and creates pet objects with the template and then returns *App.initWeb3* function call.

```
App = {
  web3Provider: null,
  contracts: {},

  init: function () {
    // Load pets.
    $.getJSON('../pets.json', function (data) {
      var petsRow = $('#petsRow');
      var petTemplate = $('#petTemplate');

      for (i = 0; i < data.length; i++) {
        petTemplate.find('.panel-title').text(data[i].name);
        petTemplate.find('img').attr('src', data[i].picture);
        petTemplate.find('.pet-breed').text(data[i].breed);
        petTemplate.find('.pet-age').text(data[i].age);
        petTemplate.find('.pet-location').text(data[i].location);
        petTemplate.find('.btn-adopt').attr('data-id', data[i].id);

        petsRow.append(petTemplate.html());
      }
    });

    return App.initWeb3();
  },
```

Figure 5. The beginning of app object, with init function /src/js/app.js

Web3 is a JavaScript library for interacting with Ethereum blockchain. It provides an API for receiving and sending data to the back-end. The *initWeb3* creates an new instance of Web3 referencing to the port where the Ganache is running the blockchain in the end it returns *initContract* function.

```
initWeb3: function () {
  if (typeof web3 !== 'undefined') {
    App.web3Provider = web3.currentProvider;
  } else {
    // If no injected web3 instance is detected, fall back to Ganache
    App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
  }
  web3 = new Web3(App.web3Provider);

  return App.initContract();
},
```

Figure 6. App.initWeb3 function - /src/js/app.js

InitContract functions uses Adoption.json file, the result of previously compiling Adoption contract from Adoption.sol. The function initiates the data from the file and sets it as a property to the contracts object. It also sets a provider for that contract and marks all the pets adopted that have account address assigned to it.

```
54    bindEvents: function () {
55      $(document).on('click', '.btn-adopt', App.handleAdopt);
56    },
57
58    markAdopted: function (adopters, account) {
59      var adoptionInstance;
60
61      App.contracts.Adoption.deployed().then(function (instance) {
62        adoptionInstance = instance;
63
64        return adoptionInstance.getAdopters.call();
65      }).then(function (adopters) {
66        for (i = 0; i < adopters.length; i++) {
67          if (adopters[i] !== '0x0000000000000000000000000000000000000000') {
68            $('.panel-pet').eq(i).find('button').text('Success').attr('disabled', true);
69          }
70        }
71      }).catch(function (err) {
72        console.log(err.message);
73      });
74    },
75
```

Figure 7. markAdopted function disables already adopted pets - /src/js/app.js

HandleAdopt function which is bound as *.btn-adopt* class' click event handler takes first account form the web3 provider and assigns it as adopter for the selected pet. This happens with the *adopt* function implemented in the Adoption.sol smart contract. If the function does not throw errors the selected pet is marked as adopted.

```
76    handleAdopt: function (event) {
77      event.preventDefault();
78
79      var petId = parseInt($(event.target).data('id'));
80
81      var adoptionInstance;
82
83      web3.eth.getAccounts(function (error, accounts) {
84        if (error) {
85          console.log(error);
86        }
87
88        var account = accounts[0];
89
90        App.contracts.Adoption.deployed().then(function (instance) {
91          adoptionInstance = instance;
92
93          // Execute adopt as a transaction by sending account
94          return adoptionInstance.adopt(petId, { from: account });
95        }).then(function (result) {
96          return App.markAdopted();
97        }).catch(function (err) {
98          console.log(err.message);
99        });
100     });
101   }
```

Figure 8. handleAdopt function connecting petIds with accountIds - srs/js/app.js

Like discussed together with the smart contract migration, Ganache is used for locally running Ethereum chain. To allow the browser to communicate with the local chain a extension MetaMask is used. This is required to let the browser retrieve data from the blockchain and let the user sign transactions to send to the blockchain. A connection is accomplished by setting the custom RPC URL to the address and port found in truffle.js configuration file. After that a local server has to be started to serve user the interface. For this the developer has to run the following command in the root folder:

```
npm run dev
```

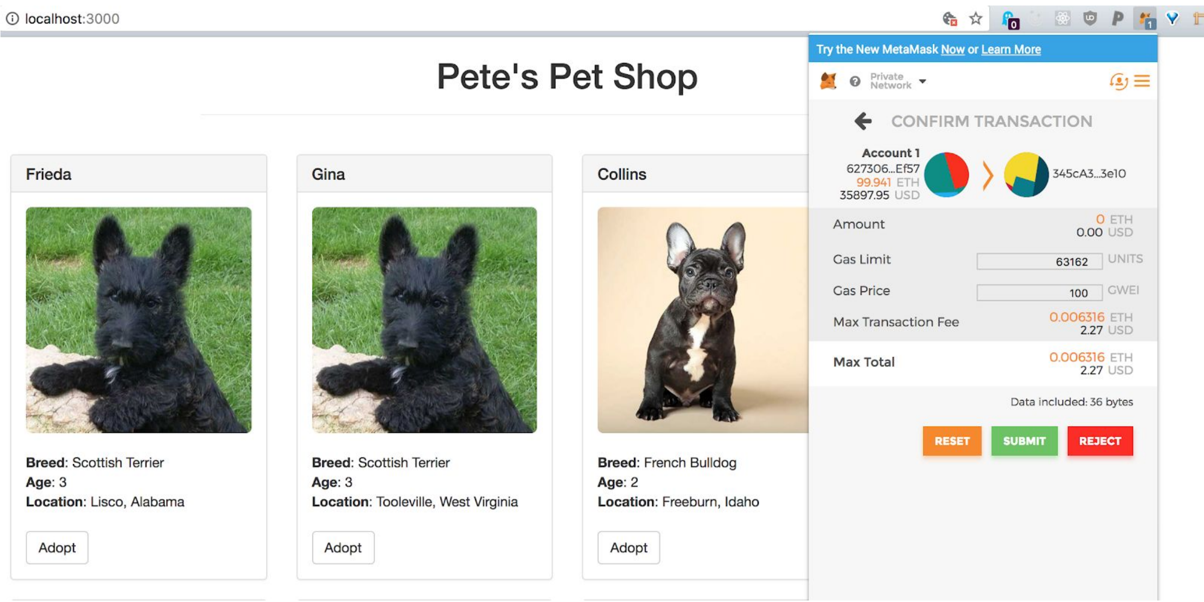

Figure 9. Running application in local machine with Metamask's transaction signing window open.

3.2 Corda

For implementing the pet shop application in Corda a java template, which is provided in Cordas public github[2] repository was used. Corda framework itself has been developed in Kotlin and most of the sample applications are implemented in Kotlin as well. Their official documentation and tutorials provide samples both in Java and Kotlin. For this thesis the author decided to use Java programming language because of the previous experience with the language.

Since Corda is permissioned and private network designed for financial institutions, the pet shop application had to be modified. Transactions in Corda can take place between only predefined number of parties and the results of the transaction are not publicly broadcasted to the world. Because of that two party nodes are defined in the application where one is the shop owner, who has all the pets and other party can initiate the adoption flow, where the owner always accepts when the transaction conditions are met.

To start off, the cordapp-template-java project is cloned from the public repository . First opening it in Intellij and importing the Gradle project results in project with the structure shown on figure 10. The AdoptionState under cordapp-contracts-states subfolder is the definition of the state that that is transitioned during the contract activation, basically it represents the data model. The concepts of flows and contracts are introduced more in detail under paragraph 2.2.2.1

---

[2]"GitHub-corda/cordapp-template-java : A Java CorDapp Template." https://github.com/corda/cordapp-template-java.

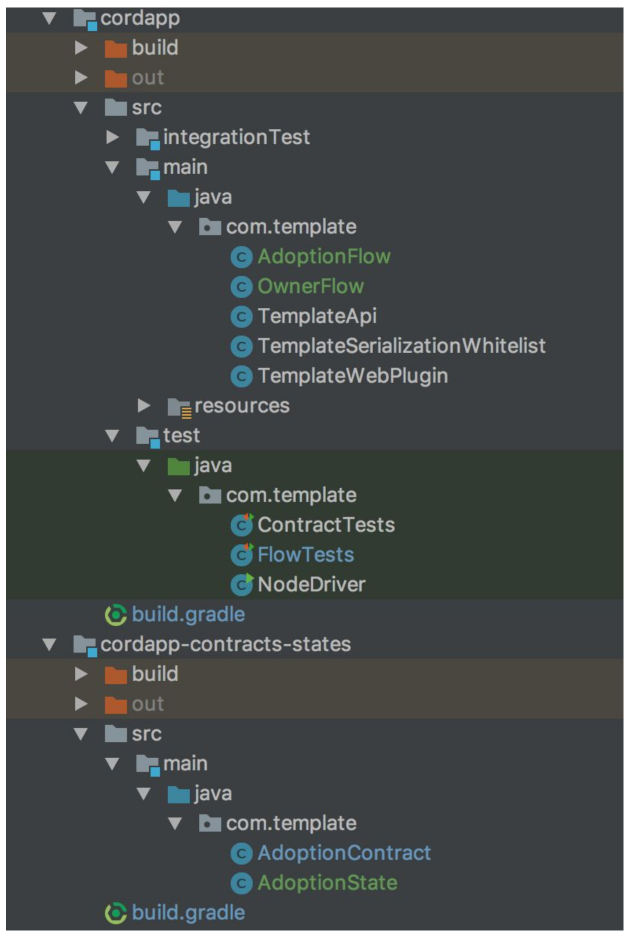

Figure 10. Structure of the Corda implementation, green and blue colors indicate modified and new files

3.2.1 State

Defining a state for the application is the first step. TemplateState.java is deleted and AdoptionState.java is created instead of it. The state represents the shared facts on the ledger so state schema and getters are defined here. State does not contain setters since state objects in Corda are immutable, meaning that values of one instance of state could not be modified. Instead state sequence is used, this means that if a state is consumed the old state is marked as historic and a new state with new values is put into the sequence. All of the historic states and current unconsumed states are kept in a Vault - the node specific relational database mentioned in 2.2.2.1 paragraph. This helps to ensure the data uniformity among parties. To keep the similarity between the two implementations this will not impact Cordas app since the pets can be adopted only once, so the initially created state will be last as well.

```java
import com.google.common.collect.ImmutableList;

import net.corda.core.contracts.ContractState;
import net.corda.core.identity.AbstractParty;
import net.corda.core.identity.Party;

import java.util.List;

/**
 * Define your state object here.
 */
public class AdoptionState implements ContractState {
    private final int petId;
    private final Party owner;
    private final Party adopter;


    public AdoptionState(int petId, Party owner, Party adopter) {
        this.petId = petId;
        this.owner = owner;
        this.adopter = adopter;
    }

    public int getPetId() { return petId; }

    public Party getOwner() { return owner; }

    public Party getAdopter() { return adopter; }

    @Override
    public List<AbstractParty> getParticipants() { return ImmutableList.of(owner, adopter); }
    /** The public keys of the involved parties. */
}
```

Figure 11. Adoption state definition with getters - cordapp-contracts-state/src/ .../AdoptionState.java

The AdoptionState class implements cordas ContractState interface and uses built in Party classes to define the transaction parties. The *getParticipants* function is returning the list of parties who are notified of the adoption transaction. This is necessary so that the new state would be sent to all of the participants in the transaction.

3.2.2. Contract

The TemplateContract.java is replaced with the AdoptionContract.java file. This is used to define how an associated state evolves over time. Contracts in Corda are different from other DLT platform smart contracts in terms of usage. In Corda the contracts impose what kind of transactions are allowed. When the input or output state do not satisfy the contract then the transaction is considered invalid.

```
public class AdoptionContract implements Contract {
  // This is used to identify our contract when building a transaction.
  public static final String ADOPTION_CONTRACT_ID = "com.template.AdoptionContract";

  /**
   * A transaction is considered valid if the verify() function of the contract of each of
   * the transaction's input and output states does not throw an exception.
   */
  @Override
  public void verify(LedgerTransaction tx) {
    final CommandWithParties<AdoptionContract.Create> command = requireSingleCommand(tx.getCommands(), AdoptionContract.Create.class);

    requireThat(check -> {
      check.using( $receiver: "No inputs when issuing adoptation", tx.getInputs().isEmpty());
      check.using( $receiver: "One output",  expr: tx.getOutputs().size() == 1);

      final AdoptionState output = tx.outputsOfType(AdoptionState.class).get(0);
      final Party owner = output.getOwner();
      final Party adopter = output.getAdopter();

      check.using( $receiver: "the petId has to be between 0 and 15, we only have 16 pets",
         expr: output.getPetId() >= 0 && output.getPetId() <= 15);
      check.using( $receiver: "owner and adopter cant be same",
         expr: output.getOwner() != output.getAdopter());

      // Constraints on the signers.
      final List<PublicKey> signers = command.getSigners();

      check.using( $receiver: "owner and adopter have to sign",
        signers.containsAll(ImmutableList.of(owner.getOwningKey(), adopter.getOwningKey())));
      check.using( $receiver: "there must be two signers",
         expr: signers.size() == 2);
      return null;
    });

  }
```

Figure 12. Adoption contract and verify function - cordapp-contracts-state/src/ .../AdoptionContract.java

The AdoptionContract class has a public method Verify seen on figure 9, which should only throw exceptions when the transaction is invalid. To start, the AdoptionContract *Create* command has to be called only once. This is required to assure that the contract which does not take any inputs but returns and output state is created and called once to avoid duplicate entries. After that comes mostly business logic. To keep the application similar to the benchmark application, the author has required that the pets can only be adopted once. This ensures that Cordas versions follows the Truffle frameworks example as closely as possible. Therefor the contract does not consume any input states and has one output state with the type of AdoptionState. Here is also defined that the owner can not be the adopter and the pet adopted can only be one of 16. The contract also requires that both parties, the adopter and the owner have signed the transaction.

3.2.3.Flow

This part allows the developer to define the sequence of steps a Corda node can and need to perform to update the ledger in a specific manner. Basically flows allow nodes to issue new AdoptionStates to the ledger. In the pet shop case it is divided into 2 flows. Firstly, the AdoptionFlow which deals with the adopter side. OwnerFlow is a subflow on Adoption flow which checks if the owner specific requirements are filled and signs the contract automatically.

```
@InitiatingFlow
@StartableByRPC
public class AdoptionFlow extends FlowLogic<Void> {
  private final Integer petId;

  private final Party ownerParty;


  public AdoptionFlow(Integer petId, Party ownerParty) {
    this.petId = petId;
    this.ownerParty = ownerParty;
  }

  private final ProgressTracker progressTracker = new ProgressTracker();

  @Override
  public ProgressTracker getProgressTracker() { return progressTracker; }
```

Figure 13. First half of AdoptionFlow class - cordapp/src/.../AdoptionFlow.java

Every flow has to be a FlowLogic subclass which override FlowLogic.call method. The annotations for the flow state that the flow could be started through a Remote-Procedure-Call (RPC) call which is similar in this case to any Representational State Transfer (REST) API call. For example user can send a GET request to a defined path like *baseUrl/example/me* via *http* protocol to the node and receive a response with data about itself. In terms of this application a PUT request should be send with query parameters for petId and owner to satisfy the constructor. ProgressTracker is for providing checkpoints for the flow observers. Observers are not used in this use case, but for example when banks move money from one country to another checkpoints have to be defined for legislators. Then legislateros can run their own node as an observer and various institutions can set checkpoints in the flow which would be broadcasted to the observer nodes.

```java
@Suspendable
@Override
public Void call() throws FlowException {

  final Party notary = getServiceHub().getNetworkMapCache().getNotaryIdentities().get(0);

  // transaction builder
  final TransactionBuilder transactionBuilder = new TransactionBuilder();
  transactionBuilder.setNotary(notary);

  // transaction components
  AdoptionState outputState = new AdoptionState(petId, ownerParty, getOurIdentity());
  StateAndContract outputContractAndState = new StateAndContract(outputState, AdoptionContract.ADOPTION_CONTRACT_ID);
  List<PublicKey> requieredSigners = ImmutableList.of(ownerParty.getOwningKey(), getOurIdentity().getOwningKey());

  Command command = new Command<>(new AdoptionContract.Create(), requieredSigners);

  // add the items to the builder.
  transactionBuilder.withItems(outputContractAndState, command);

  // Verify that the transaction is valid.
  transactionBuilder.verify(getServiceHub());

  // adopter signs
  final SignedTransaction adopterSignedTransaction = getServiceHub().signInitialTransaction(transactionBuilder);


  // create session with owner
  FlowSession ownerPartySession = initiateFlow(ownerParty);

  // get signature from owner
  SignedTransaction fullySignedTransaction = subFlow(
    new CollectSignaturesFlow(adopterSignedTransaction, ImmutableList.of(ownerPartySession), CollectSignaturesFlow.tracker()));


  // finalize
  subFlow(new FinalityFlow(fullySignedTransaction));


  return null;
}
```

Figure 14. Second half of AdoptionFlow class with call method - cordapp/src/ .../AdoptionFlow.java

The custom call method holds most of the logic. Adding a Suspendable annotation to the call method lets the call be suspended when a long lasting action is encountered so the node can move on to other flows meanwhile. For example when Bob wants lend money from Alice, Bob's node can initiate the flow with sending a signed transaction to Alice. While Alice is processing the transaction Bob's side of the flow can be suspended and start other flows meanwhile.

First line of the code creates a separate party instance for the notary. Every transaction requires one, this is for timestamping and preventing double-spending. Most of the information related to the node itself comes from the ServiceHub. Then a new transaction builder is created and the notary is set to it. Then the transaction components: output, StateAndContract, required signers and the command variables are defined. All of the components are then added to the builder. Then the transaction is verified with the node data.

If everything is valid so far, the flow moves on to signing the transaction. Firstly, the adopter signature is added. Since adopter is initiating the flow and everything is valid so far, nothing else has to be checked. Then the flow for ownerParty is created.

```
@InitiatedBy(AdoptionFlow.class)
public class OwnerFlow extends FlowLogic<Void> {
  private final FlowSession ownerPartyFlow;

  public OwnerFlow(FlowSession ownerPartyFlow) { this.ownerPartyFlow = ownerPartyFlow; }

  @Suspendable
  @Override
  public Void call() throws FlowException {
    class SignTxFlow extends SignTransactionFlow {
      private SignTxFlow(FlowSession ownerPartyFlow, ProgressTracker progressTracker) {
        super(ownerPartyFlow, progressTracker);
      }

      @Override
      protected void checkTransaction(SignedTransaction stx) throws FlowException {
        requireThat(require -> {
          ContractState output = stx.getTx().getOutputs().get(0).getData();
          require.using( $receiver: "must be Adoption transaction", output instanceof AdoptionState);
          AdoptionState adoptionState = (AdoptionState) output;
          require.using( $receiver: "petId must be between 0 and 15",
              expr: adoptionState.getPetId() >= 0 && adoptionState.getPetId() <= 15);
          return null;
        });
      }
    }
    subFlow(new SignTxFlow(ownerPartyFlow, SignTransactionFlow.Companion.tracker()));

    return null;
  }
}
```

Figure 15. OwnerFlow class - cordapp/src/.../AdoptionFlow.java

As annotation implies this is a a subflow initiated by the AdaptionFlow. The *Call* method is similar to the one in parent flow class. Addition of *SingTxFlow* lets the subflow to automatically verify previous signatures and the transaction. Even though it might be contractually valid a double check is added. CheckTransaction function checks if the output really is instance of AdoptionState and if the petId is in correct margin. If everything passes without exceptions it is signed and passed back to AdoptionFlow where the flow is finalized.

3.2.4. Local Nodes, Front-end and Interaction

The front-end and api layer are not implemented for Corda application since the author considers them to be outside of the scope and they do not differ from average web application. If front-end would be implemented it would be functional with a form with a single input for petId. The application would send a basic http PUT request to the server over

API call with the selected petId and ownerId as query parameters and wait for http response. The API controller would check the sent parameters and start the applications flow. Rest of the logic is already defined in the flow components.

Within the authors implementation the network is interactable via command line interface (CLI). For that the *deployNodes* task in the build.grade file is edited to represent the current use case figure 16.

```
tasks.withType(JavaCompile) {
    options.compilerArgs << "-parameters" // Required for passing named arguments to your flow via the shell.
}

task deployNodes(type: net.corda.plugins.Cordform, dependsOn: ['jar']) {
    directory "./build/nodes"
    networkMap "O=Controller,L=Estonia,C=EE"
    node {
        name "O=Controller,L=Estonia,C=EE"
        advertisedServices = ["corda.notary.validating"]
        p2pPort 10002
        rpcPort 10003
        cordapps = [
                "$project.group:cordapp-contracts-states:$project.version",
                "$project.group:cordapp:$project.version",
                "net.corda:corda-finance:$corda_release_version"
        ]
    }
    node {
        name "O=Owner,L=Estonia,C=EE"
        advertisedServices = []
        p2pPort 10005
        rpcPort 10006
        webPort 10007
        cordapps = [
                "$project.group:cordapp-contracts-states:$project.version",
                "$project.group:cordapp:$project.version",
                "net.corda:corda-finance:$corda_release_version"
        ]
        rpcUsers = [[ user: "user1", "password": "test", "permissions": []]]
    }
    node {
        name "O=Adopter,L=Estonia,C=EE"
        advertisedServices = []
        p2pPort 10008
        rpcPort 10009
        webPort 10010
        cordapps = [
                "$project.group:cordapp-contracts-states:$project.version",
                "$project.group:cordapp:$project.version",
                "net.corda:corda-finance:$corda_release_version"
        ]
        rpcUsers = [[ user: "user1", "password": "test", "permissions": []]]
    }
}
```

Figure 16. DeployNodes task in build.gradle file.

Next we need can run it in our terminal window with the gradle command:

```
./gradlew clean deployNodes
```

This is creates the build catalogues to the structure like seen on figure 10 with that all the java code is compiled as well. To start all the nodes a second command is necessary to run in the same root folder:

```
build/nodes/runnodes
```

This runs all the separate nodes defined in the task on figure 16 and opens separate terminal windows for all of them. Those are different nodes running in the same computer just for

testing purposes. In the Adopter's node terminal window we can interact with the Adopter's node by initiating the AdoptionFlow and giving it the required parameters: integer for petId as *arg0* and human readable id for the node, defined as *name* in *deployNodes* task as *arg1*.

```
Listening for transport dt_socket at address: 5005

                                    I won $3M on the lottery so I donated a quarter
                                    of it to charity. Now I have $2,999,999.75.

--- Corda Open Source 2.0.0 (f91995b) ----------------------------------------------


Logs can be found in                : /Users/germo/Documents/Kool/thesis/corda/cordapp-template-java/build/nodes/Adopter/logs
Database connection url is          : jdbc:h2:tcp://192.168.0.12:65125/node
Incoming connection address         : localhost:10008
Listening on port                   : 10008
RPC service listening on port       : 10009
Loaded CorDapps                     : corda-finance-2.0.0, cordapp-0.1, cordapp-contracts-states-0.1, cordapp-template-java-0.1,
corda-core-2.0.0
Node for "Adopter" started up and registered in 20.04 sec

Welcome to the Corda interactive shell.
Useful commands include 'help' to see what is available, and 'bye' to shut down the node.

Thu Aug 09 22:40:36 EEST 2018>>> flow list
com.template.AdoptionFlow
net.corda.core.flows.ContractUpgradeFlow$Authorise
net.corda.core.flows.ContractUpgradeFlow$Deauthorise
net.corda.core.flows.ContractUpgradeFlow$Initiate
net.corda.finance.flows.CashConfigDataFlow
net.corda.finance.flows.CashExitFlow
net.corda.finance.flows.CashIssueAndPaymentFlow
net.corda.finance.flows.CashIssueFlow
net.corda.finance.flows.CashPaymentFlow

Thu Aug 09 22:51:49 EEST 2018>>> start AdoptionFlow  arg0: "1", arg1: "O=Owner,L=Estonia,C=EE"

✅  Done

Thu Aug 09 22:52:00 EEST 2018>>> start AdoptionFlow  arg0: "17", arg1: "O=Owner,L=Estonia,C=EE"

🚫  Done
☠   Contract verification failed: Failed requirement: the petId has to be between 0 and 15, we only have 16 pets, contract: com.templ
ate.AdoptionContract@3a863136, transaction: B4D10F0B2779C8BF7758940389A2C01A6F38CBB3211F534E905A5CC0B3C181FE
```

Figure 17. Adopter nodes CLI window starting the AdotpionFlow.

With that we can see that the verify method in AdoptionContract works as expected.

4. Comparison

This paragraph contains the comparison of the two previously discussed platforms. The platforms are compared in five aspects: main use case, architecture, consensus, language support and the pet shop application implementation. These points should cover common questions developers might have when choosing a platform for dApp development. Main use

case will hint the developer which type of application the platform is intended for. Architecture paragraph will bring out the most influential aspects of the development cycle and how to approach it. Consensus algorithms are discussed next. Language support helps the developers finalize the decision if the platform is appropriate for the team or if new languages should be adopted. And finally the implementation comparison brings out some of the advantages and disadvantages the author encountered during the implementation.

4.1 Main Use Case

The main use case of the selected platforms differ a lot. Ethereum was initially visoned to be the new alternative protocol for building decentralised applications. So it is built as a general purpose platform for various types of application. There are applications reanging from games such as CryptoKitties to decentralized exchange platforms such as IDEX. To ensure the simplicity and universality of the platform some trade offs were made regarding performance. This makes it ideal platform for companies to launch an Initial Coin Offering (ICO). The number of total applications launched on Ethereum chain is a great proof for that.

Corda on other hand has a really specific goal of unifying and simplifying the transactions between financial institutions with use cases covering simple money transfer to more complex bond, stock and loan transfers. Every transaction in corda can be can be enforced with real life contract which is necessary for financial institutions dealing with large amounts of money. This makes the platform to a quite niche product which would indicate less users having a purpose for it. On the other hand, since major banks are supporting Corda shows that there definitely is interest for this kind of product. In the future we could see more Corda related job listing form traditional financial companies or even fintech startups.

Another difference is that Ethereum platform has issued their own cryptocurrency. It is not per se a main use case of the platform, yet it has various uses on its own for example as a means of investment or currency for online trading. On the other hand, the fact that Ethereum is publicly traded could influence the direction of the platforms development to raise the exchange prices since this is a primary funding for the development team aswell or the platform could be considered dead when the price drops drastically. Corda as a platform does not have anything related to its own currency. It is funded by the consortium which

mostly consists of banks. Since the source code of the platform is public the author does not see any downsides in that. Inversely this could be an good thing, since this helps the development team to focus on specific needs and offer a tool those companies would benefit from.

4.2 Architecture

Ethereum is built as a permissionless and public blockchain, which means that everybody can be part of the network and contribute to finding the longest chain necessary for the data validation. Also, all of the ledger updates are broadcasted to all of the nodes in the network. Some security is provided via node anonymity. Smart contracts in Ethereum represent so called autonomous agents in the network which automatically verify and facilitate transaction between nodes. When transactions are completed the outcome state is associated with corresponding accounts. Accounts in Ethereum can represent both the client nodes and contracts. When choosing the the platform one should consider the transaction throughput. Since Ethereum is public and permissionless and the number of applications running on the chain is constantly growing this could become an issue. The data changes are heavily related to the mining aspect. This suggests that the adding new blocks could become slower and slower. On the other hand, the overall architecture can be considered easier than Cordas since there are only smart contract to perform the logic and accounts to hold the state.

Corda is permissioned and private blockchain platform. This means that the entrance to the network is blocked by the “doorman” where every node has to identify themselves and provide proof. This is necessary to accommodate the needs of financial use-cases and this also makes Sybil attacks unlikely. Corda also uses point-to-point communication which transfers data only to authorized parties. This provides data security and some form of transaction anonymity. There is also no central point of data storage. Every node has their own subset of the shared facts on the ledger. Contracts in Corda represent the transaction validity verification and are ran automatically. Like mentioned in chapter 4.1 contracts in Corda can be legally binding which makes conflict resolution easier between parties since they can rely on conventional legal system. Flows are the representation of transaction process. They determine who are the associated parties. What kind of contracts the transaction has to fulfil and who need to sign it. Subflows can be created to require party specific contract verification. This shows that the architecture has more components to it, on

one side it makes it more difficult to understand or plan. On the other hand, it provides all the tools necessary for the applications fitting with the main use case discussed in previous chapter. This is usually needed for building more complex applications without having to write too much boilerplate code.

4.3 Consensus

Ethereum currently uses Proof-of-Work algorithm. This requires that nodes verify the transaction collectively and broadcast the new global state of the chain to network. Like mentioned in architecture discussion this can hinder the performance of the platform. A future versions of Ethereum are expected to introduce Proof of Stake algorithm. This would make Ethereum less reliant of the networks collective computing power and would use various algorithms for adding a new block of transaction to the chain. This indicates the active development around the platforms core components. Since the final solution is yet to be released, the change could result in huge fluctuation in the user base or the platforms popularity.

Corda has introduced the concept of notaries for achieving consensus. Notaries test that a given transaction does not consume any input states which have been consumed by any previous already signed transaction. Second part of consensus is achieved through Cordas contracts which check the validity of transaction inputs and outputs and require signatures from related parties. Concerned parties can run the same contract code independently which furthermore increases the trust. This results in constant speed during transactions. The amount of users in the network has no influence and the speed mostly depends on the transaction parties. Since the main use case requires that the contracts would be legally binding aswell, the notion that opposing parties can run their contract code independently is greatly beneficial.

4.4 Language

Ethereum has an independent programming language Solidity. Solidity is easy to use and learn since it heavily influenced by other scripting languages such as Python and

JavaScript. Its main purpose is to make writing contracts for Ethereum chain easy and accessible. All of the code is targeted to Ethereum Virtual Machine. Since Solidity has the biggest user base and also support it is highly suggest to adopt it for the developers planning on using Ethereum. This would mean extra effort and consumed time to achieve efficiency.

Cordas core code is mostly developed in Kotlin and is targeted to Java Virtual Machine. This means that basically any JVM language could be used but the officially support Java and Kotlin. Both of them have samples and tutorials on Cordas documentation and github. Since Java is mostly dominant on enterprise applications development and is ranked as the top programming language in Tiobe indexes, it should be easier to currently working in financial enterprises to start using Corda

4.5 Implementing Pet Shop Use Case

The implementation between the platforms is hugely different. It is influenced a lot by the main goal of the platforms and the resulting architecture. Writing a smart contract on Ethereum and with Solidity was fast and easy but connecting it all to the chain required various tools and help of a framework. Since Ethereum's community and ecosystem is so extensive, this also was a breeze. The are tools for specific use cases like MetaMask browser extension which helped easily connect to local blockchain and Ganache which allowed to deploy said local blockchain with virtual accounts. Also, giving Ethereum advantage over here is its bigger user base and longer time on market. Forums and message boards like StackOverflow have many discussions and suggestions covering wide range of most common questions.

Implementing the same application in Corda was more time consuming and alterations had to be made to accommodate it with Cordas architecture. Biggest issue came from corda being private and permissioned network. Implementing a webstore like application would render the platforms advantages pointless when one central node would hold all the pets and involve all the other nodes into every transaction. To use the concepts provided by the platform a smaller network was created with two participants, pet owner and an adopter. Regardless of that, the core implementation in corda needed four classes and nearly 250 lines of code when including all the imports when Ethereum only needed one contract file with 17 lines of code. This is mostly influenced by the fact that the application

did not fit with the Cordas use case and the application being fairly small and simple. Results with more complex and bigger applications from other fields could differ a lot. Since Corda has more of niche market there were much fewer discussions online. This could also be caused by the fact that applications on corda are meant for private networks.

Conclusion

The goal of this thesis was to analyze and differentiate two platforms Ethereum and Corda. For that, the author provided an overview of blockchain and distributed ledger technologies covering core concepts, main characteristics of decentralised applications and a simple explanation for smart contracts.

The analysis covered theoretical aspects of the platforms: main use case, architecture, consensus algorithm and supported programming languages. To compare the platforms from developer standpoint and experience their features and quirks a simple application for pet adoption was implemented on both of them. The application was meant to be identical on both of the platforms, but it was not completely possible and reasonable as explained in fifth paragraph .

In conclusion, the selected two platforms are quite different from one another. Both of them try to automate and secure online transaction, when Ethereum provides tools to do it publicly and broadcast it to everyone on the network, then Corda is doing it privately and sharing the data to parties with permissions. In terms of the implemented application use case, Ethereum is much reasonable option, since the use case matched with the platforms case and it was quick and required less effort to get running locally.

This thesis will hopefully provide a great summary and overview for developers choosing a blockchain to develop their decentralised application on. For future research the author suggests comparing Corda with other private blockchain platforms, since these platforms serve rather different purpose than public chains.

References

1. Leaf C. Believe the Hype: Here's the Actual Next Big Thing in Tech. In: Fortune [Internet]. 22 Aug 2017 [cited 20 Apr 2018]. Available: http://fortune.com/2017/08/22/blockchain-next-big-thing-tech/

2. Economist T. The next big thing. In: The Economist [Internet]. The Economist; 9 May 2015 [cited 20 Apr 2018]. Available: https://www.economist.com/news/special-report/21650295-or-it-next-big-thing

3. Natarajan H, Krause S, Gradstein H. Distributed Ledger Technology (DLT) and Blockchain [Internet]. The World Bank; 2017. Report No.: 1. Available: http://documents.worldbank.org/curated/en/177911513714062215/pdf/122140-WP-PUBLIC-Distributed-Ledger-Technology-and-Blockchain-Fintech-Notes.pdf

4. Boucher P. How blockchain technology could change our lives [Internet]. European Parliamentary Research Service; 2017 Feb. Available: http://www.europarl.europa.eu/RegData/etudes/IDAN/2017/581948/EPRS_IDA(2017)581948_EN.pdf

5. Distributed Ledger Technology's Transformational Impact on Federal Financial Management. In: The Bureau of the Fiscal Service [Internet]. [cited 20 Apr 2018]. Available: https://www.fiscal.treasury.gov/fsservices/gov/fit/fit_jffm.htm

6. UK Government Chief Scientific Adviser. Distributed Ledger Technology: beyond block chain [Internet]. Government Office for Science; 2016. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf

7. Staples M, Chen S, Falamaki S, Ponomarev A, Rimba P, Tran AB, et al. Risks and opportunities for systems using blockchain and smart contracts [Internet]. CSIRO; 2017. Available: http://www.data61.csiro.au/en/Our-Work/Safety-and-security/Secure-Systems-and-Platforms/Blockchain

8. Martinovic I, Kello L, Sluganovic I. Blockchains for Governmental Services: Design Principles, Applications, and Case Studies [Internet]. University of Oxford; 2017. Report No.: 7. Available: https://www.ctga.ox.ac.uk/sites/default/files/ctga/documents/media/wp7_martinovickellosluganovic.pdf

9. Ansper A, Buldas A, Willemson J. Krüptograafiliste algoritmide elutsükkel [Internet]. Cybernetica; 2017. Available: https://www.ria.ee/public/RIA/kruptograafiliste_algoritmide_elutsukli_uuring_2017.pdf

10. Yli-Huumo J, Ko D, Choi S, Park S, Smolander K. Where Is Current Research on Blockchain Technology?—A Systematic Review. PLoS One. Public Library of Science; 2016;11: e0163477.

11. Ethereum Pet Shop -- Your First Dapp | Truffle Suite. In: Truffle Suite [Internet]. [cited 20 Apr 2018]. Available: http://truffleframework.com

12. Mills, David, Kathy Wang, Brendan Malone, Anjana Ravi, Jeff Marquardt, Clinton Chen, Anton Badev, Timothy Brezinski, Linda Fahy, Kimberley Liao, Vanessa Kargenian, Max Ellithorpe,

Wendy Ng, and Maria Baird (2016). "Distributed ledger technology in payments, clearing, and settlement," Finance and Economics Discussion Series 2016-095. Washington: Board of Governors of the Federal Reserve System, https://doi.org/10.17016/FEDS.2016.095. https://www.federalreserve.gov/econresdata/feds/2016/files/2016095pap.pdf

13. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System [Internet]. 2008 [cited 24 Mar 2018]. Available: https://bitcoin.org/bitcoin.pdf

14. Brito J, Castillo A. Bitcoin: A primer for Policymakers [Internet]. 2013 [cited 24 Mar 2018]. Available: https://www.mercatus.org/system/files/Brito_BitcoinPrimer.pdf

15. Zheng Z, Xie S, Dai H, Chen X, Wang H. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. 2017 IEEE International Congress on Big Data (BigData Congress). 2017. doi:10.1109/bigdatacongress.2017.85

16. Lamport L, Shostak R, Pease M. The Byzantine Generals Problem [Internet]. [cited 13 Apr 2018]. Available: https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf

17. King S, Scott N. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake [Internet]. 19 Aug 2012 [cited 13 Apr 2018]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf

18. Vasin P. BlackCoin's Proof-of-Stake Protocol v2 [Internet]. [cited 13 Apr 2018]. Available: https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf

19. Nxt community. Nxt Whitepaper [Internet]. 12 Jul 2014 [cited 13 Apr 2018]. Available: https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf

20. Castro M, Liskov B. Practical Byzantine Fault Tolerance. [cited 13 Apr 2018]. Available: http://pmg.csail.mit.edu/papers/osdi99.pdf

21. Mingxiao D, Xiaofeng M, Zhe Z, Xiangwei W, Qijun C. A review on consensus algorithm of blockchain. 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2017. doi:10.1109/smc.2017.8123011

22. Jayachandran P. The difference between public and private blockchain - Blockchain Unleashed: IBM Blockchain Blog. In: Blockchain Unleashed: IBM Blockchain Blog [Internet]. 31 May 2017 [cited 4 Jul 2018]. Available: https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/

23. Szabo N. Smart Contracts: Building Blocks for Digital Markets [Internet]. [cited 13 Apr 2018]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

24. Cuccuru P. Beyond bitcoin: an early overview on smart contracts. Int J Law Info Tech. Oxford University Press; 2017;25: 179–195.

25. Decentralized Applications – dApps. In: BlockchainHub [Internet]. [cited 13 Apr 2018]. Available: https://blockchainhub.net/decentralized-applications-dapps/

26. State of the ĐApps — A List of 1,687 Projects Built on Ethereum [Internet]. [cited 4 Jul 2018]. Available: https://www.stateofthedapps.com/

27. Kelly J. Three banks join R3 blockchain consortium taking total to 25. In: Reuters [Internet]. 28

Oct 2015 [cited 4 Jul 2018]. Available: https://uk.reuters.com/article/uk-global-banks-blockchain/three-banks-join-r3-blockchain-consortium-taking-total-to-25-idUKKCN0SM1UH20151028

28. SEB: Blockchain Could Make Banks 'Radically More Efficient'. In: CoinDesk [Internet]. 30 Sep 2015 [cited 4 Jul 2018]. Available: https://www.coindesk.com/swedens-seb-blockchain-could-make-banks-radically-more-efficient/

29. Ethereum. Ethereum: Now Going Public. In: Ethereum Blog [Internet]. 23 Jan 2014 [cited 4 Jul 2018]. Available: https://web.archive.org/web/20140302035654/http://blog.ethereum.org/2014/01/23/ethereum-now-going-public/

30. Ethereum wiki [Internet]. Github; Available: https://github.com/ethereum/wiki

31. ethereum. ethereum/wiki. In: GitHub [Internet]. [cited 5 Aug 2018]. Available: https://github.com/ethereum/wiki/Programming-languages-intro

32. Smith K. Ethereum's move to PoS — First version of Casper released. In: BraveNewCoin [Internet]. 12 May 2018 [cited 25 Jul 2018]. Available: https://bravenewcoin.com/news/ethereums-move-to-pos-first-version-of-casper-released/

33. Ethereum github wiki [Internet]. Github; Available: https://github.com/ethereum/wiki

34. Kelly J. Nine of world's biggest banks join to form blockchain partnership. In: Reuters [Internet]. 15 Sep 2015 [cited 4 Jul 2018]. Available: https://www.reuters.com/article/us-banks-blockchain/nine-of-worlds-biggest-banks-join-to-form-blockchain-partnership-idUSKCN0RF24M20150915

35. Brown RG. R3 Corda: What Makes It Different - corda.net. In: corda.net [Internet]. 25 Oct 2016 [cited 5 Jul 2018]. Available: https://www.corda.net/2016/10/r3-corda-makes-different/

36. Hearn M. Corda: A distributed ledger. In: Corda documentation [Internet]. 29 Nov 2016 [cited 5 Jul 2018]. Available: https://docs.corda.net/_static/corda-technical-whitepaper.pdf

37. Flows — R3 Corda V3.1 documentation [Internet]. [cited 5 Jul 2018]. Available: https://docs.corda.net/key-concepts-flows.html

38. Consensus — R3 Corda V3.1 documentation [Internet]. [cited 5 Jul 2018]. Available: https://docs.corda.net/key-concepts-consensus.html

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Germo Hünerson ,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Comparison of Ethereum and Corda

supervised by Fredrik Payman Milani and Luciano García-Bañuelos,

2. I am aware of the fact that the author retains these rights.
3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **05.08.2018**