NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme 'HSE University and University of London Double Degree Programme in Data Science and Business Analytics'

UDC ______________

Research Project Report (Final)

on the topic ______Corporate settlements mechanism based on smart contracts_______

Fulfilled by the Student:
group #БПАД 193 ______[illegible]______ Jingtao Xu
Signature — Surname, First name, Patronymic, if any

30.05.2021
Date

Checked by the Project Supervisor:
Pavel Bolotov P.
Surname, First name, Patronymic (if any), Academic title (if any)
Senior SwA, Digital Innovations Department
Job
Raiffeisen Bank Russia
Place of Work (Company or HSE Department)

Date 29.05. 2021 | 10 | Болотов
Grade according to 10-point scale | Signature

Moscow 2021

The National Research University Higher School of Economics
Moscow
DSBA - Department of Computer science

REPORT

Subject «Coursework»
Topic «Project - Virtual Settlement»

Group	193
Student	Xu Jingtao
Supervisor	Bolotov Pavel P. [Raiffeisen Bank Russia]
Submission Date	2021

Moscow

Abstract

Data-management innovation is always on the top of the challenge list in every field. Over the last five years, the expansion of blockchain technology has enabled many industries to rethink data-management solutions that can find the equilibrium between high efficiency, decentralisation and security (that is "The Impossible Trinity of Blockchain".) Now, there are dozens of developed "Chains" that have different attributes and capabilities due to the change of consensus algorithms. In our project, we are going to develop and design a virtual settlement mechanism based on the R-Chain abstract business objects. In this report, the author will explicitly explain how Distributed Ledger Technologies can be utilised in applications of virtual settlement. We represent the final result of our approach that dynamically interacts with smart contracts (blockchain) and database.

Contents

1. Introduction

Over the past decade, several waves of new E-commercial technologies has emerged to meet societal and financial demands. "Coins, banknotes, cheques and credit cards were each innovation in their own day" — (Giannini (2011)). In recent years, there has been much buzz in the global market about Blockchain[1] (Distributed Ledger Technology). Since this technology field represents a significant opportunity for every ecosystem to establish leadership in areas of cross-border payments, trade finance and so on, a group of leading banks decided to step into the picture and contribute to the development of D-apps[11] and DLT[19].

Blockchains can be generally classified as: public chains, consortium chains, and private chains. These three classifications divide the blockchain technology into three types of applications, and the degree of openness of a decentralized application is actually very important to determine whether it is completely opened, semi-opened or completely closed, because the degree of openness is different, it's implementation will also be different. In our project, we are going to consider the completely closed networks, especially related to the banking system.

The social distancing measures caused by the Covid-19 pandemic have dramatically increased the speed of the shift toward digital payments, and it is carrying security issues along with it. Hence, one fundamental task of blockchain-based settlement is dealing with the settlement risk. The implementation of such a settlement has to set proper block sizes and computation times to ensure its ability to trace cash flow and make it easier to enforce financial regulations. This project works with the design and implementation of a D-Apps[11] that is able to meet requirements stated in section 3.

2. The objectives of Corporate settlements mechanism

2.1. Goals

This R&D project "Corporate settlements mechanism based on smart contracts" provides us an opportunity to understand the core principles of decentralized applications (D-Apps) and blockchain technology. This project is intended as a contribution to the improvement of the existing "abstract deals" concept on the R-Chain platform. Our goal is to develop and implement a mechanism for a group of companies based on R-Chain abstract business objects.

From an individual viewpoint, a feasible implementation of virtual settlements requires two type of instruments:

- First method is to implement central accounting smart contract.
- Second way is to develop a mechanism which deals with dedicated node (Oracle).

Moreover, it will be necessary to determine advantages and disadvantages of both approaches, as well as the possible boundaries of their applicability.

2.2. Roles of participants

This is a two-member team project. Another team member is Ranga Baminiwatte (2nd year student of Data Science & Business Analytics) who does the analysing and testing part. I am dedicated in practical programming tasks and development. However, at the final stage of our project, once we have done the development of our settlement mechanism and smart contracts, we will use Raiffeisen server as the testbed to apply and check the functionalities of our work. Finally, we will come up with conclusions.

3. Requirements of the mechanism

There are certain functional and non-functional requirement of the mechanism that we are going to design and develop.

3.1. Functional

- To make settlement within the platform using virtual accounts.
- Settlements are made when a certain deal status occurs.
- Settlements must be confirmed by payers.

3.2. Non-functional

- Performance: up to 100 deals by 1 hour, up to 500 status changes by 1 hour, up to 1000 executed payments by 1 hour
- Number of participants for 1 deal - up to 6
- Each participant should have an opportunity to work with multiple accounts

4. Terminologies and Definitions

[1] Blockchain – Blockchain technology is a decentralized virtual ledger that records the provenance and flow of a digital asset.

[2] Blocks - A block is a permanent container of logs and records.

[3] Smart Contract - a collection of agreements that exist in a digital format, including the various agreements followed by the parties in performing these agreements.

[4] Mining – The process of securing and verifying bitcoin transactions.

[5] Genesis block - First block of a Blockchain

[6] Gas - The amount of currency required to do a transaction in the network.

[7] Bitcoin – The most popular digital cryptocurrency introduced in January 2009.

[8] Ether – The cryptocurrency used in Ethereum platforms.

[9] R-Chain - A novel blockchain platform based on the formal framework of decentralized computation which leverages software development through producing massive scale concurrent blockchains.

[10] Geth - Go-ethereum is command-line interface (CLI) tool that interact with the Ethereum network and links your device and the rest of the Ethereum nodes.

[11] D-Apps - A computer application that runs on a distributed computing system.

[12] Ethereum wallet - API or mobile applications that allow you to interact with Ethereum accounts.

[13] Solidity - is an OOP (object-oriented programming) language for smart contracts development. It is used in implementing Ethereum based blockchain platforms as the main developing tool.

[14] Truffle - is an all-in-one development platform used to test frameworks and pipeline assets. It is based on Ethereum Blockchain and is used to develop flawless and efficiently functioning D-Apps.

[15] Proof-of-Work or Proof-of-Stake - a type of consensus protocol which use a certain amount of computational resources to prove other parties's work, or we can think it as an arbitrary mathematical puzzle.

[16] DLTs - Distributed Ledger Technologies are P2P networks that allow several participants to constantly update and share their ledgers. DLTs provide their members possibilities to securely deal with their transactions.

[17] Ethereum – An open-source blockchain which uses a decentralized method of software architecture and uses the cryptocurrency unit ether.

[18] Ethereum network - A separated Blockchain that is remotely functioning from the Main Ethereum network. These Private Networks are mainly developed and used by organizations to regulate the reading accessibility of the Blockchain.

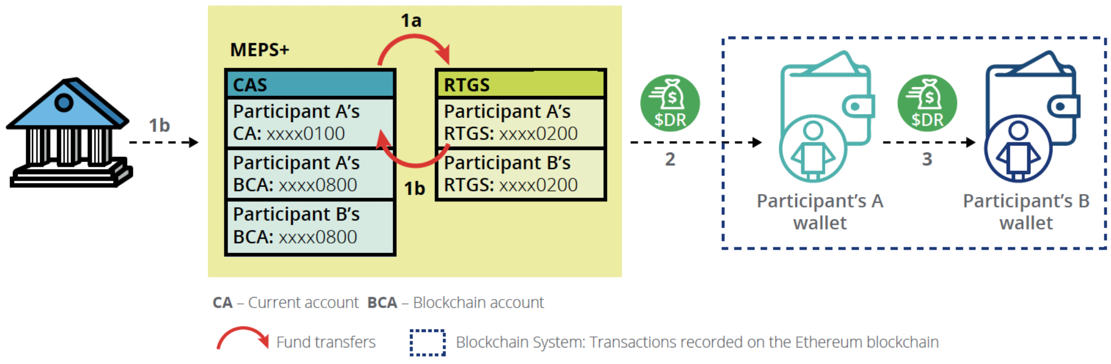
[19] SQL - Structured Query Language.(Used for database management)

[20] Swarm - A storage platform which allows participants to efficiently combine their storage capacity and network bandwidths in order to provide a base layer native service of the Ethereum web 3 stack. (From https://swarm-guide.readthedocs.io/en/stable/introduction.html)

[21] IPFS - a protocol and peer-to-peer network for storing and sharing data in a distributed file system. (From wikipedia)

[22] Quorum - Quorum Blockchain was built by JPMorgan using Ethereum network. The absence of gas cost is an important feature of Quorum.

5. Overview and Comparative analysis of sources



MEPS+
1a
CAS
Participant A's CA: xxxx0100
Participant A's BCA: xxxx0800
Participant B's BCA: xxxx0800
RTGS
Participant A's RTGS: xxxx0200
Participant B's RTGS: xxxx0200
1b
1b
$DR
2
$DR
3
Participant's A wallet
Participant's B wallet
CA – Current account BCA – Blockchain account
Fund transfers
Blockchain System: Transactions recorded on the Ethereum blockchain

Figure 1. Sample structure[UBIN project]

The success of Bitcoin is testimony to the fact that a peer-to-peer decentralised digital currency is feasible. Besides, Ethereum demonstrated nicely the need for a decentralized application development platform. If we say Bitcoin and Ethereum are "The Blockchain v1.0 and v2.0", then R-Chain can be definitely considered as "The Blockchain v3.0" – such thing will establish an ecological blockchain environment.

From an individual viewpoint, the current implementation of Raiffeisen bank's R-Chain platform provides end-to-end deals management. However, settlements between the participants of the deals are carried out within ordinary bank settlement services outside the R-Chain, while only payment documents are transmitted through the platform as a justification for performing the corresponding operations. Currently, there are several decentralized platforms that are developing to implement p2p payments between participants (both banks and their clients). However, in such platforms, payments are implemented separately from the deal's management. In addition, the payment execution process is associated with a significant exchange of documents. To obtain a new service quality, it is desirable to combine in one platform both the management of the deals and the execution of payments, also the possibility to manage the efficiency of the participants' funds while ensuring their liquidity.

The sample structure of virtual settlement in UBIN project illustrates the basic idea of a sufficient flexibility and functionality in the usage of Business applications, including a proactive monitoring system(In figure `figure` 1). In this project, we may get the idea of what main challenges we are facing today and how can we tackle them. It has five phrases in total, but we are only interested in Phase 2 (see `table` 1) at current stage. In Phase 2 report, it says that they have implemented a real-time gross settlement system which has the following critical features:

- Digitalisation of Payments
- Decentralised Processing
- Payment Queue Handling
- Privacy of Transactions
- Settlement Finality

These features enable this system to safely execute a great amount of payments. As a result, it significantly reduces the cost of banking system. From a fully developed financial point of view, this innovation is definitely beneficial to the global economic-ecosystem.

Table 1. Ubin project, five phases

Phase	Date
Phase 1: Tokenised SGD on DLT[16]	16 November 2017 - 9 March 2017
Phase 2: Re-imagining RTGS	5 October 2017 - 14 November 2017
Phase 3: Delivery versus Payment (DvP)	24 August 2018 - 11 November 2018
Phase 4: Cross-border Payment versus Payment (PvP)	15 November 2018 - 2 May 2019
Phase 5: Enabling Broad Ecosystem Collaboration	11 November 2019 - 13 July 2020

6. Theoretical part & Description of mechanism

6.1. General structure of R-Chain[9] project

Having analyzed the possibilities and consequences of this approach, Raiffeisen bank's team of R-Chain project came to a conclusion that is necessary to introduce an adapter of unified business objects which is an additional layer between the backend of business and the technical components of distributed platform – Geth[10] and IPFS (Reserved option: Swarm[20]). (Adapter DLT in the `figure` 2).

Considering the commercial exploitation prospects of the R-chain[9] blockchain[1] platform, we must ensure it will be released into the market smoothly and without unexpected incidents. Looking at `figure` 2, the architecture of R-chain[9] blockchain platform provided by Raiffeisen Bank Russia has a core block called `Business Processes Mirror`, which contains a set of components. These components are responsible for dealing with data and operations of the Universal Process. `"Mirror"` conducts a two-way conversion of the information used in the Univeral Business Process and DLT-client[19].

In other words, there is a maintained synchronous Database relevant to each Node of business processes. Besides, this format is the most convenient to meet the need of the Business app. The business app interacts with this data management of Business processes (DataBase), receiving information about Processes state and sending operations on the state changes.

Universal Business Process in terms of DLTs.

Universal Business Process should have the following attributes: (Raiffeisen Bank)

- Process parameters (type of Process, status, contextual Process attributes)

- Role list of Process members
- List of electronic documents related to Process
- Process transitions map

At the same time, the Platform should provide the following conditions of the Process spread in the net on the level of blockchain-component:

- Completeness and wholeness of the information
- Confidence of the electronic documents outside of the pool of Process members
- Control over the adherence to the Process transitions map
- Storage of the revision history of the Process states

The attributes of the Universal Business Process platform provide maximum use of blockchain-platform advantages on the one hand and maximum flexibility and functionality in Business app usage. It does not make sense to formalize this diversity of alternatives into only one template of smart contracts. The development of the unique template for a concrete type of business process is a significantly more effective way. It provides high flexibility, a possibility of a "strict" insertion of both necessary elements and critical examinations into the "core" of a blockchain component, which eliminates any manipulations on each part of individual Members. Apart from this, the Platform combines a commonality of interfaces for Apps-Processes interaction and high modularity of their specific functionality.

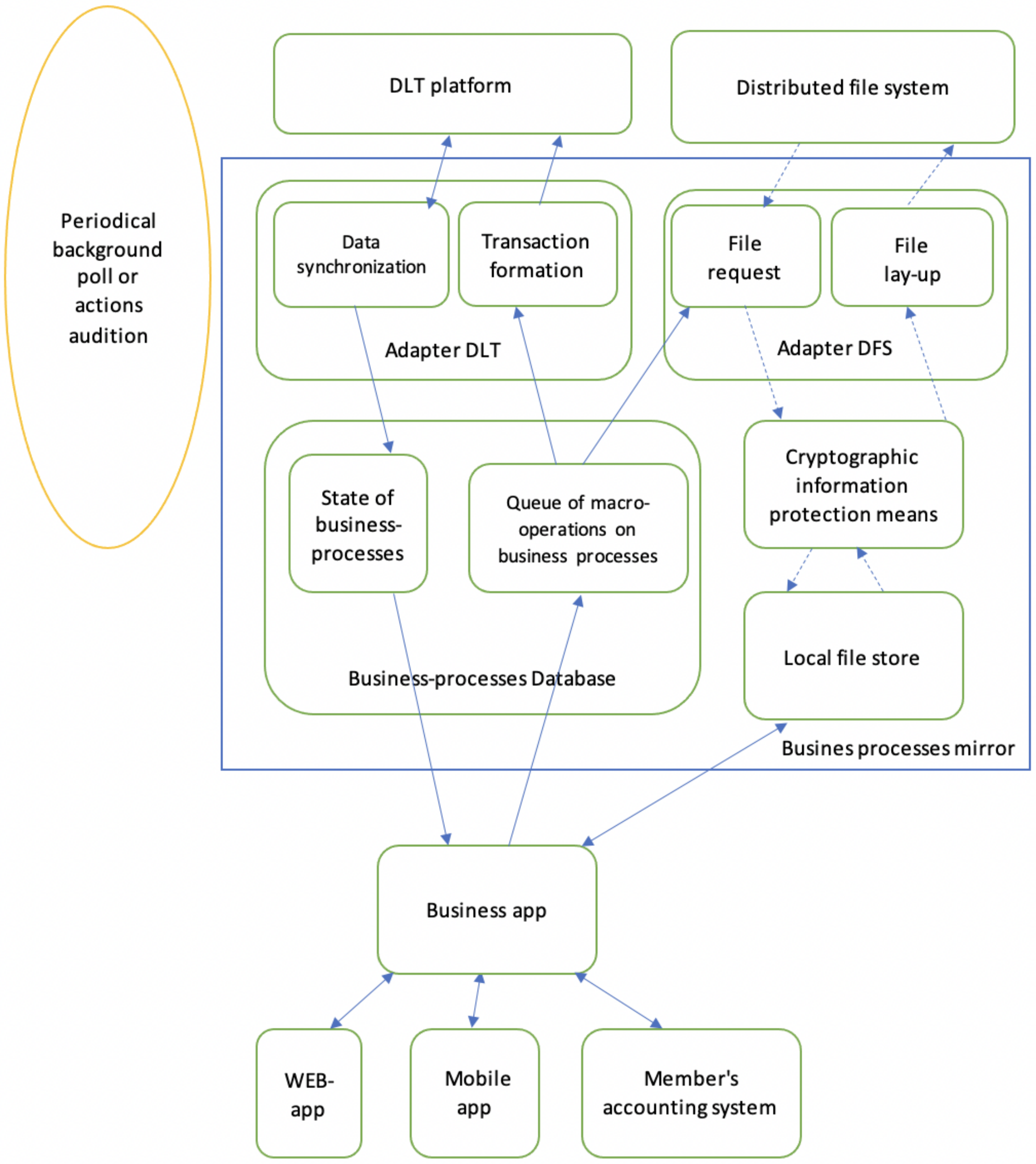
Debates of the universal business process platform

Possibility of the hidden modifications of the blockchain basis ——————

Business App does not directly depend on the concrete implementation of blockchain basis, and it allows to change this basis without affecting the Business App. Labour costs for a Business App (especially with a developed dialogue interface) can comprise 75-95% of the total labour costs for a program complexity. There are some possible changes, for instance, switching to another blockchain basis, using two parallel blockchains Ethereum for the increase of capacity. On the other hand, there is a well-known debate - "Is it possible that a selected basis will maintain interoperability after the consequent updates? Will it still function in 2 years?"

Possibility of standardization & gateway exchange on process state level ——

This point seems to be fantastic on the current level of corporate D-Apps. Abstract objects creation and formalization is the essential step in standardization and a necessary condition of a wide integration of blockchain nets. However It is clear that only a Universal Business Process is not sufficient – complicated relations, and physical world processes, and requirements of business apps will certainly lead to the establishment of other abstract objects with their own features and possibilities.



Periodical background poll or actions audition
DLT platform
Distributed file system
Data synchronization
Transaction formation
Adapter DLT
File request
File lay-up
Adapter DFS
State of business-processes
Queue of macro-operations on business processes
Business-processes Database
Cryptographic information protection means
Local file store
Busines processes mirror
Business app
WEB-app
Mobile app
Member's accounting system

Figure 2. The general design of R-chain[9] blockchain platform

6.2. BlockChain[1] & Ethereum[17]

The development of BlockChain technology opened new possibilities in many fields and industries. Firstly, we have to understand what blockchain is and why it stands out as a bright spot. In a strict sense, a blockchain can be envisioned as a linked list (data structure) that sequentially combines data blocks in a chronological order. Moreover, the distributed node consensus algorithm and cryptography are used for data protection. In other words, we can say that Blockchain was built on the basis of the perfect combination of the advantages of cryptography, hash algorithm and game theory.

There are three important concepts about BlockChain: (see `figure` 3)

- Transaction: It is basically an operation on the blockchain that triggers a block state to change. Such operation can be a transfer of digital asset or a piece of information (notification of events).

- Block: The accumulated records of transactions and status information that occur over a period of time are packaged and it forms a block by using hash algorithm and consensus algorithm. In an enterprise-level blockchain platform, the consensus time can be set dynamically.

- Chain: Blocks are linked to the blockchain in a chronological order, and each block records the hash value association of the previous block, which is a record of the entire state-change history.

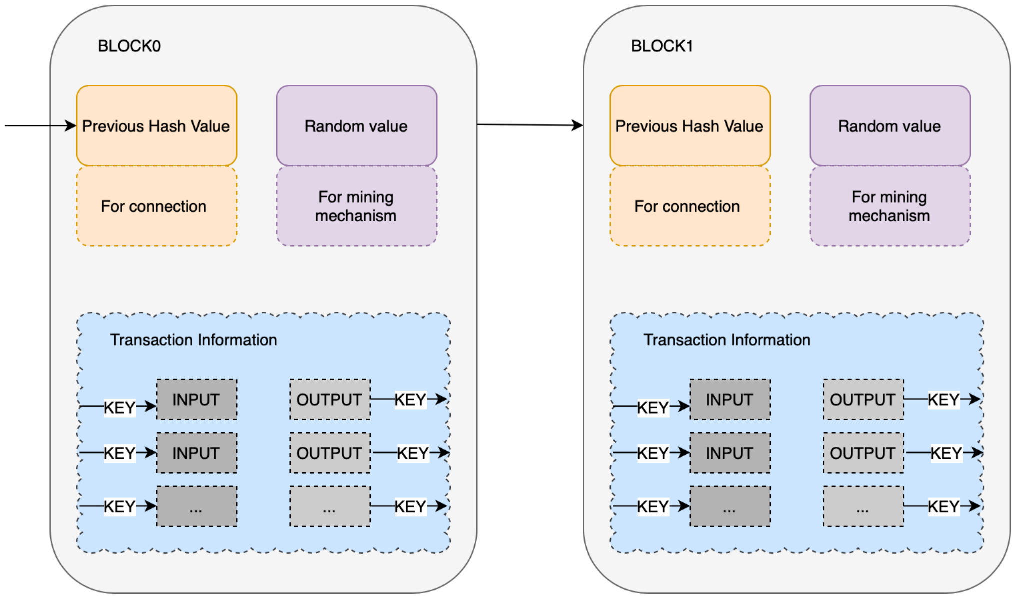


Figure 3. BlockChain System

The classification of BlockChains

According to the different ways of node participation, blockchain technology can be divided into: Public Blockchain, Consortium Blockchain and Private Blockchain.

- `Public Blockchain` is fully public to everybody, therefore everyone can be a node in the network and participates in "bookkeeping". In the process of the consensus formation of algorithms, each node can contribute to the consensus process, which is also what we generally call "mining", to obtain economic rewards proportional to the contribution, that is, the digital tokens issued in the system.

Advantages	Disadvantages
Transparent transaction data.	Low TPS (Transactions Per Second)
Cannot be tempered with	Slow and Expensive

- `Private Blockchain` is entirely controlled by one organization who has write and read privileges. Therefore, the application scenario of the private chain is generally the management aspect of a single company's internal headquarters to the branch, such as database management and auditing. Compared with other types, the value of the private blockchain is much higher due to the fact that they can provide a secure, traceable and tamper-resistant platform, and can prevent both internal and external security attacks.

Advantages	Disadvantages
Fast transaction speed	Contradict with decentralisation
Cannot be easily attacked	
Protects privacy and data	

- `Consortium Blockchain` is a multi-centralized or partially decentralized blockchain. When the blockchain system is running, its consensus process may be controlled by some designated nodes. For instance, in a blockchain system with 15 financial institutions connected to it, each institution acts as a node on the chain. Every confirmation of a transaction requires at least ten nodes to confirm (2/3 confirmation). In this chain, only consortium member nodes can access it, and operations such as read and write permissions on the chain and participation in accounting rules need to be jointly decided by the consortium member nodes.

Advantages	Disadvantages
Fast transaction speed	restricted in the field of usage
Cannot be easily attacked	
Protects privacy and data	
More credible than private chain	

Ethereum is the representative platform of BlockChain 2.0. It is a blockchain development platform that provides a Turing Complete smart contract[3] system. Through Ethereum, users can write smart contracts and build D-APPs by themselves. Based on the Turing Complete nature of the Ethereum smart contract, developers can develop any decentralized application, such as voting, financial transactions, intellectual property, smart-property, and so on. There are currently many decentralized applications running on the Ethereum platform. According to its white paper, they can be divided into three types of applications. The first type is financial applications, including "digital currency", financial derivatives, hedging contracts, and savings wallets, which involve financial transactions and value transfer applications. The second type is semi-financial applications. They involve the participation of money, but a large part of it is non-monetary. The third type is non-financial applications, such as online voting and decentralized autonomous organizations that do not involve money at all.

Raiffeisen bank uses Quorum[22] due to the feature of Raft consensus mechanism. Raft consensus mechanism has two special attributes: 1. faster blocktime; 2. the absence of forking. When using Ethereum in a private network, gas is paid, but it is distributed free of charge to participants through administrative accounts. In addition, Quorum also provides the transactions of digital tokens on a DL (Distributed ledger) without revealing data and sensitive information.

Through the study of fundamental knowledge about blockchain, we can start to develop our smart contract. In the next section, there are details of the realisation of this project.

7. Realisation of the Project

In the following subsections, I will firstly introduce the tools and instruments that I have chosen to use, then focus on the project itself with a detailed report on the tasks that I have completed during this academic year.

This is our GitHub repository.

7.1. Tools and instruments

- `Solidity`[13] - an object-oriented programming language for writing smart contracts. It is used for implementing smart contracts on various blockchain platforms, most notably, Ethereum.
- `C++` - a powerful general-purpose programming language.
- `Github` - the largest web service for hosting IT projects and their joint development.
- `pgAdmin 4` - a free and open source graphical user interface administration tool for PostgreSQL(database management system [DBMS]).
- `Putty` - a terminal simulator with secure remote shell access to a UNIX or Linux system.

7.2. Description of Project Implementation

First of all, it is seen that the starting point of this workflow process (see `figure` 4) is a set of Smart Contracts which contain the required functions and data structures, such as `Account` and `Payment`. After being deployed, Smart Contracts are transformed into hexadecimal codes. Then, hexadecimal codes are added to BlockChain network by transactions. Secondly, R-chain[9] Adapter is a solid base of our D-app[11] (It is described in

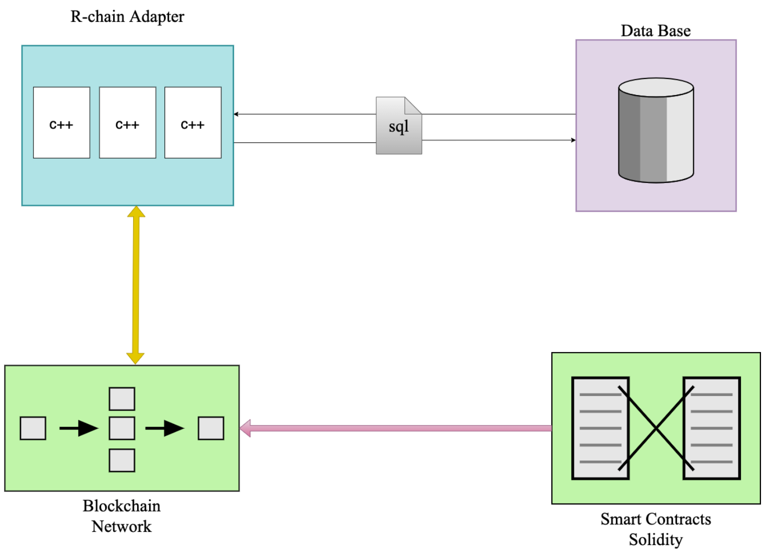


Figure 4. Connection between each functional elements

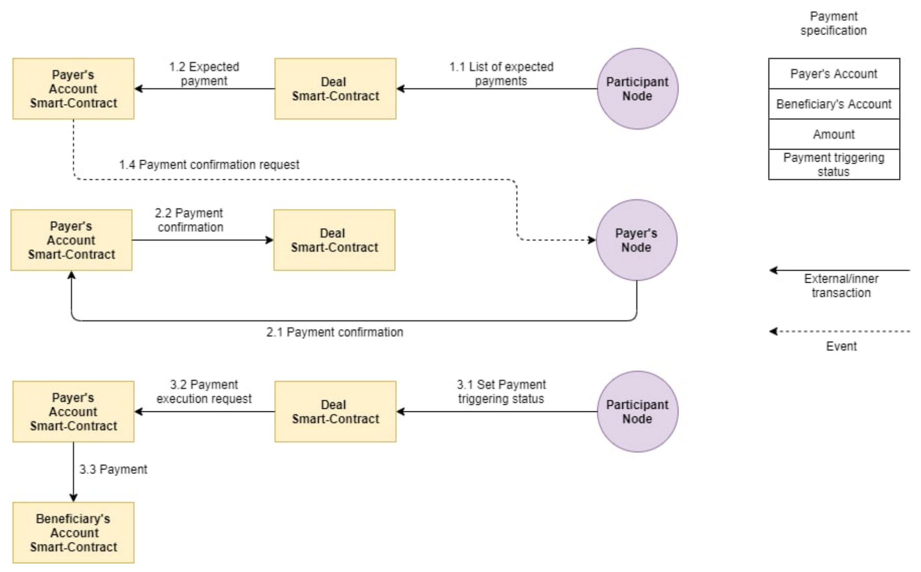
subsection 2.2.2). Briefly, it plays a role of correspondent to record and send the changes on the BlockChain network and interact with database through SQL[19] scripts.

7.2.1 Smart Contracts[3]

The exchange protocol is implemented within an Ethereum[17] smart contract[3], because it is publicly accessible, and moreover, no additional costs are imposed on users beyond standard gas costs due to the feature of Quorum[22]. This part is absolutely central to the whole realisation of this project.

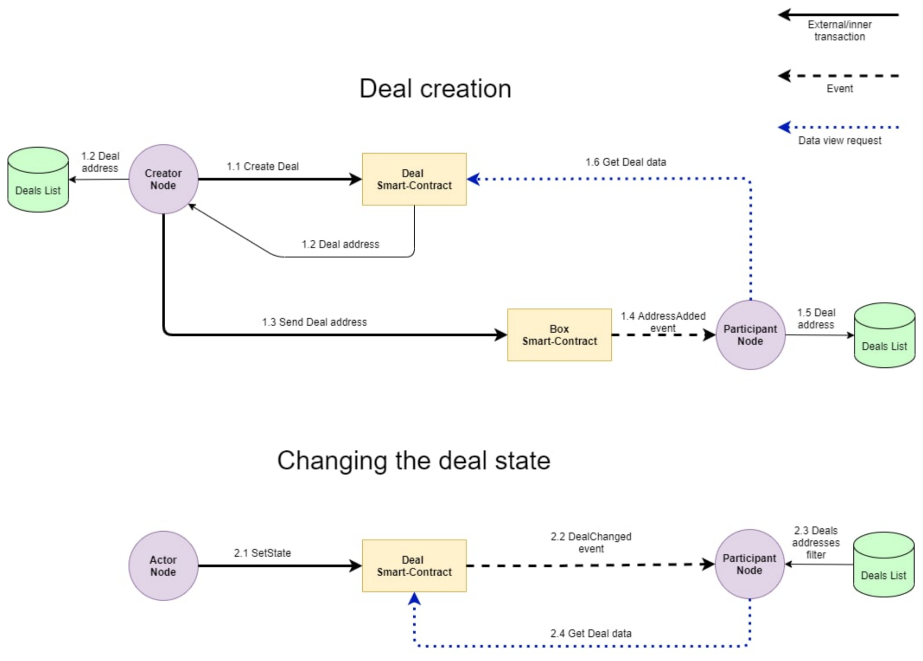
At the very initial stage, we wrote two Smart Contracts[3] (*Deal_Free.sol and Account.sol*). Deal_Free was created for handling with Payments. In the diagram `figure` 5a, it is a complete procedure of deals' settlements. From arrow 1.1 to 1.2, deal smart contract requires a function which can efficiently store numerous payments and their details. A function `LoadPayments(bytes32[] memory Payments_)` - loads a list of payments to our smart contract Deal_Free by storing data in a struct called `Payment` that has five fields: `address account_from`; `address account_to`; `uint256 amount`; `bytes32 state`; `bytes32 confirmed`. After the expected payments have been sent to their payer's account smart contract, payer's node (in 1.4) can decided whether this payment is confirmed by using function `PaymentAccept()`. Finally, if payers confirm to set deals, the information will be sent back to Deal smart contract. In the Deal smart contract, the state of corresponding payments is changed to "Y" (means that payments are ready to be executed).

One noteworthy feature is the payment triggering statue, which is stored in the mapping data structure (mapping from states to payment addresses), can avoid to check the state of each payment. The approach could greatly optimize the time complexity of payment execution. Diagram `figure` 5b illustrate the internal actions of Deal creation and Changing the deal state which are implemented in these two functions.



Payment specification
Payer's Account
Beneficiary's Account
Amount
Payment triggering status
Payer's Account Smart-Contract
1.2 Expected payment
Deal Smart-Contract
1.1 List of expected payments
Participant Node
1.4 Payment confirmation request
Payer's Account Smart-Contract
2.2 Payment confirmation
Deal Smart-Contract
Payer's Node
2.1 Payment confirmation
External/inner transaction
Event
Payer's Account Smart-Contract
3.2 Payment execution request
Deal Smart-Contract
3.1 Set Payment triggering status
Participant Node
3.3 Payment
Beneficiary's Account Smart-Contract

(a) pic1. Basic structure of two mentioned smart contracts



(b) pic2. The workflow of Deal.sol

Figure 5. Mechanism of the virtual settlement(transaction)

`Description of Functions & Operations`.
The functionality of all operations are shown in the following two tables *Account.sol* and *Deal_free.sol*:

Table 2. account.sol

Operations	Descriptions
PutPayment()	It forms a payment and emits *AcceptPayment* event.
PaymentAccept()	It is called for setting status = 'Y' (`ONLY` by owner).
GetBalance()	It returns the current balance of the caller.
GetPayment()	It gives full information of payment by passing ID and ADDRESS.
Income()	It increases balance by the given AMOUNT.
ExecutePayment()	After checking all necessary conditions, it does the transaction and emits *SetBalance* and *PaymentExecuted* events.

Table 3. deal_free.sol

Operations	Descriptions
SetStatus()	It's used to set the status of deal(New_, Foamed_, Accepted_, Rejected_,Paid).
PutPayment()	It inputs a single payment and push into a new array PaymentsTriggers if state of payment is execute_state_.
LoadPayments()	It's used to iterate through an array of Payments and Calls PutPayment function on each array element.
ConfirmPayment()	It's used to set Payments[_id].confirmed="Y".

7.2.2 R-Chain Adapter & DataBase

Moving to the core of our DApp - R-Chain Adapter, it is a back-end server which handles every function calling and event emitting. On the server side of this application, we need to design the corresponding functions which carry out the role of interaction between Smart Contracts and Data Base. In other words, each function in our smart contracts has its corresponding fucntion in the R-chain adapter, e.g. `GetBalance()` <—> `EMIRi_dl_GetBalance()`. C++ functions establish a "bridge" between BlockChain Network and our DataBase. Whenever there is any data modification, R-Chain Adapter will detect the change and draw a quick respond, that is to update such change to a new block with a new timestamp on the BlockChain.

```
int   EMIR_dl_action_AddDeal      (Dl_action *action, SQL_link *db, char *error) ;
int   EMIR_dl_action_SetStatus    (Dl_action *action, SQL_link *db, char *error) ;
int   EMIR_dl_action_AddPayments  (Dl_action *action, SQL_link *db, char *error) ;
int   EMIR_dl_action_AddAccount   (Dl_action *action, SQL_link *db, char *error) ;
int   EMIR_dl_action_UpdateBalance(Dl_action *action, SQL_link *db, char *error) ;
int   EMIR_dl_action_GetPayment   (Dl_action *action, SQL_link *db, char *error) ;
long long  EMIRi_dl_GetBalance    (char *contract, char *error) ;
int   EMIRi_dl_GetDealId          (Dl_action *action) ;
```

Figure 6. Additional functions created

All added functions are listed in `figure` 6. Taking function `EMIR_dl_action_UpdateBalance()` as an example, it requires the following steps to update a new balance to the target account.

- Initialisation of SQL pointer.

```
/*-------------------------------- initialisation */
    *error = 0 ;
  if(code == NULL)  code = (char *)calloc(1, _CODE_SIZE) ;
  status = EMIR_db_nodepars(db, error) ;
  if(status)  return(-1) ;
```

- To obtain account and password.

```
/*---------------------------- account & password */
    memset(account, 0, sizeof(account)) ;

if(action->executor[0] != 0)
   strncpy(account,  action->executor, sizeof(account)-1) ;
else
   strncpy(account, _member_executor, sizeof(account)-1) ;

 password = strchr(account, ':') ;
if(password == NULL) {
                      password = "";
                 }else{
                     *password = 0 ;
                      password++ ;
                 }
```

- To parse `action->data` into its constituents (64-hexadecimal symbols size).

```
/*----------------------- Parse data into fields */
    do {
               memset(json, 0, sizeof(json)) ;
              strncpy(json, action->data, sizeof(json)-1) ;
     r_id     = strstr(json, "\"Id\":\"") ;
     r_amount = strstr(json, "\"Amount\":\"") ;
  if(r_id    == NULL ||
     r_amount== NULL   ) {  r_id=NULL ;  break ;  }

     r_id    = strstr(r_id,      "\":\"")+3 ;
     r_amount= strstr(r_amount,  "\":\"")+3 ;

     end = strchr(r_id, '"') ;
  if(end == NULL) {  r_id=NULL ;  break ;  }
    *end = 0 ;

     end = strchr(r_amount, '"') ;
  if(end == NULL) {  r_id=NULL ;  break ;  }
    *end = 0 ;
               amount = strtoull(r_amount, &end, 10) ;
   } while(0) ;
```

- To check the existence of account.

```
/*----------------------- Check Account existance */
  sprintf(text, "select \"BlockChainId\" "
                "from   %s "
                "where  \"Id\"='%s'",
                __db_table_deals_accounts, r_id) ;
      status = db->SelectOpen(Cursor, text, NULL, 0) ;
   if(status)
   {
                  strcpy(error, db->error_text) ;
                                db->error_text[0]=0 ;
                              __db_errors_cnt++ ;
                                        break ;
     }
      status = db->SelectFetch(Cursor) ;
  if(status)
  {
   if(status == _SQL_NO_DATA)  sprintf(error, "Get data: No record") ;
   else                        sprintf(error, "Get data: %s", db->error_text) ;
                db->error_text[0] = 0 ;
                    __db_errors_cnt++ ;
                                break ;
     }
```

```
        memset(contract, 0, sizeof(contract)) ;
        strncpy(contract, (char *)Cursor->columns[0].value, sizeof(contract)-1) ;
        db->SelectClose(Cursor) ;
```

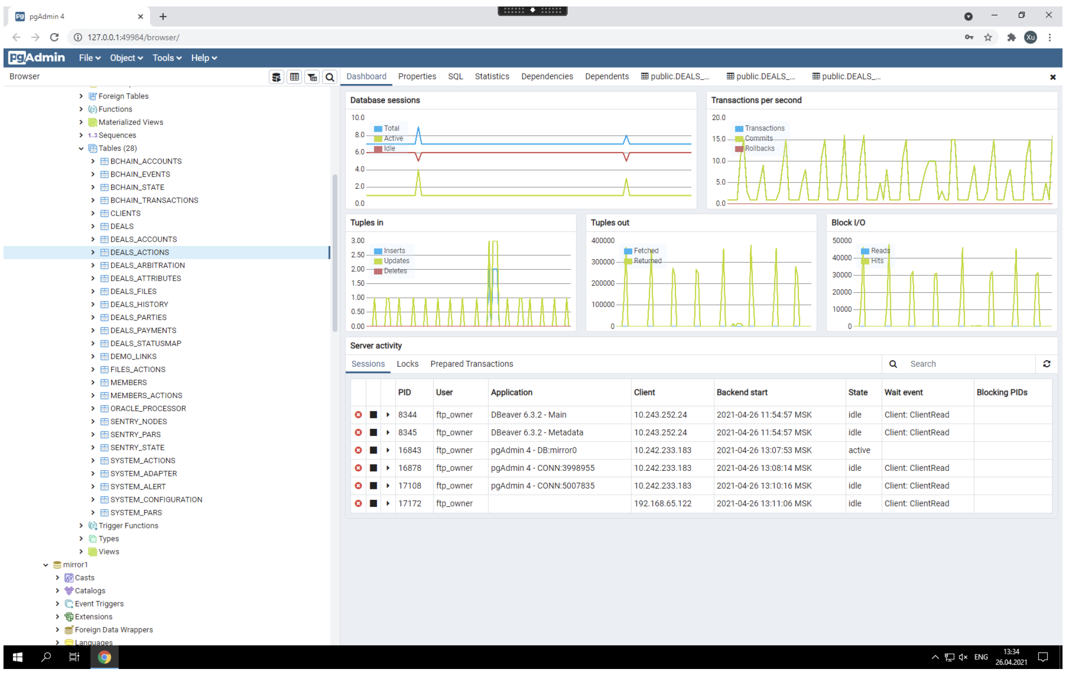
- To send the new amount of balance.

```
/*-----------------------  Form Income transaction */
  sprintf(code, "4e26002d"          //   Income(int256 amount) manifest
                "%064llx",          //   Amount in hex
                amount
         ) ;
/*-----------------------  Send transaction */
  do {
      status=EMIR_node_checkgas(account,       /* Calculate transaction gas */
                                contract, code, gas, error) ;
      if(status)  break ;

      status=EMIR_node_unlockaccount(account,  /* Unlock account */
                                    password, reply, sizeof(reply)-1, error) ;
      status=EMIR_node_sendcontract(account,   /* Send transaction */
                                   contract, code, gas, txn, error) ;
     } while(0) ;
  if(status)  break ;
/*----------------------- Entering answer to the DB */
  db->SelectClose(Cursor) ;

  if(status==0)  sprintf(text, "update %s "
                               "set    \"Status\"='WAIT', \"Reply\"='%s,%lld'"
                               "where  \"Id\"=%s",
                   __db_table_deals_actions, txn, old_balance, action->id) ;
  else  sprintf(text, "update %s "
                      "set    \"Error\"='%s'"
                      "where  \"Id\"=%s",
                __db_table_deals_actions, error, action->id) ;
  status=db->SqlExecute(Cursor, text, NULL, 0) ;
```

- There are two possible ways to check whether the balance has changed:
  - Comparison of old and new balances.
  - Event `IncomeBalance check`.

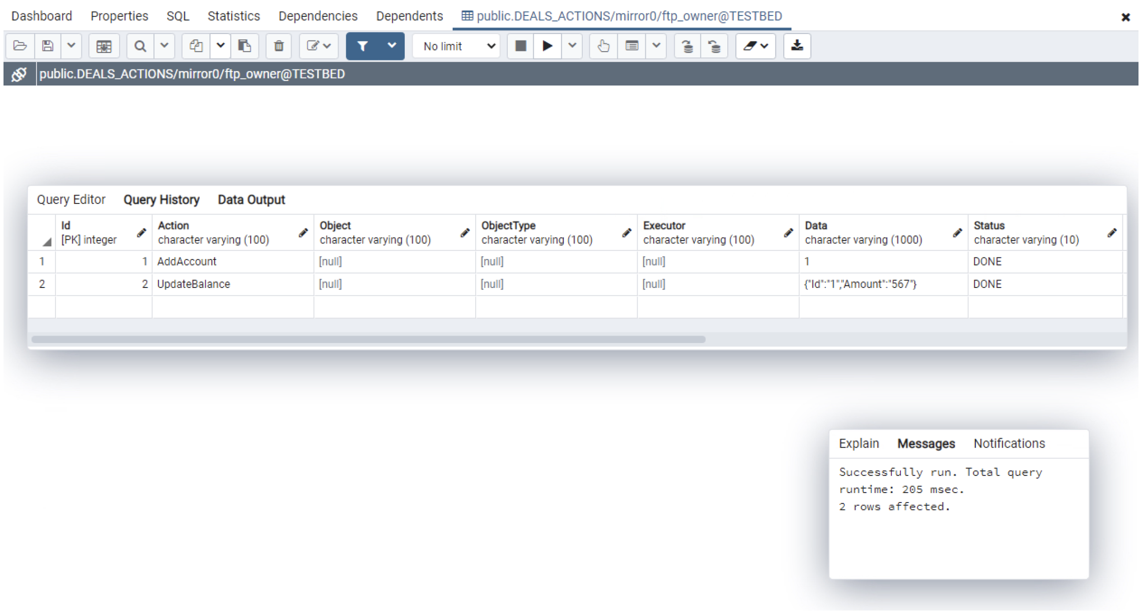
The pieces of code shown above is protected by NDA.

Moreover, SQL scripts are used for interaction between R-Chain Adapter and DataBase. We use pgAdmin4 to manage our database (as in `figure` 7) In particular, there are two actions `AddAccount` and `UpdateBalance` in the Deals_Actions table. First, after calling the function EMIR_dl_action_AddAccount (see in `figure` 8), a new account is added to the table Deals_Account. `UpdateBalance` is used for changing the account balance which is described in detail above.



PID	User	Application	Client	Backend start	State	Wait event	Blocking PIDs
8344	ftp_owner	DBeaver 6.3.2 - Main	10.243.252.24	2021-04-26 11:54:57 MSK	idle	Client: ClientRead	
8345	ftp_owner	DBeaver 6.3.2 - Metadata	10.243.252.24	2021-04-26 11:54:57 MSK	idle	Client: ClientRead	
16843	ftp_owner	pgAdmin 4 - DB:mirror0	10.242.233.183	2021-04-26 13:07:53 MSK	active		
16878	ftp_owner	pgAdmin 4 - CONN:3998955	10.242.233.183	2021-04-26 13:08:14 MSK	idle	Client: ClientRead	
17108	ftp_owner	pgAdmin 4 - CONN:5007835	10.242.233.183	2021-04-26 13:10:16 MSK	idle	Client: ClientRead	
17172	ftp_owner		192.168.65.122	2021-04-26 13:11:06 MSK	idle	Client: ClientRead	

Figure 7. pgAdmin4 interface & Tables



	Id [PK] integer	Action character varying (100)	Object character varying (100)	ObjectType character varying (100)	Executor character varying (100)	Data character varying (1000)	Status character varying (10)
1	1	AddAccount	[null]	[null]	[null]	1	DONE
2	2	UpdateBalance	[null]	[null]	[null]	{"Id":"1","Amount":"567"}	DONE

```
Successfully run. Total query
runtime: 205 msec.
2 rows affected.
```

Figure 8. Table - Deals_Actions

8. Analysis & Evaluation of results

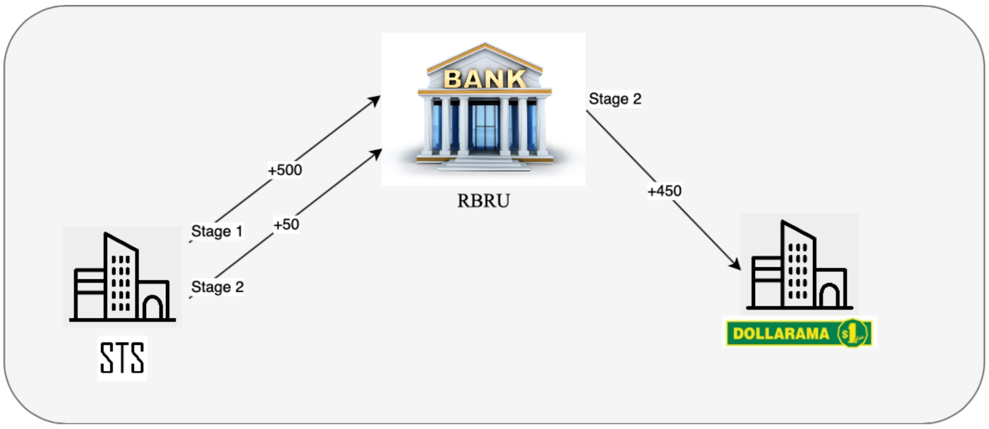


Figure 9. Demo Scenario 1

After the entire development of our virtual settlement mechanism, we are going to assess our approach through one testing scenario. It is a basic typical financial transaction settlement that consists of three payments. The participants of this deal are RBRU, STS and DOLARAMA. The whole procedure can be divided into two stages:

- On stage 1, STS initiates one payment of *500* to RBRU.

- On stage 2, RBRU is going to move *450* to account of DOLARAMA. At the same time, STS has to pay bank an extra fee of *50*.

First, we have to deploy our smart contract *Account.sol* by using RPC API(Raiffeisen Bank). We form the transaction:

```
{"jsonrpc":"2.0", "method":"eth_sendTransaction",
 "params":[{"from":"0x...",
            "gas":"0x400000",
            "data":"0x(hexcode of Account.sol)"}],"id":1}
```

After that, we need to do the following steps:

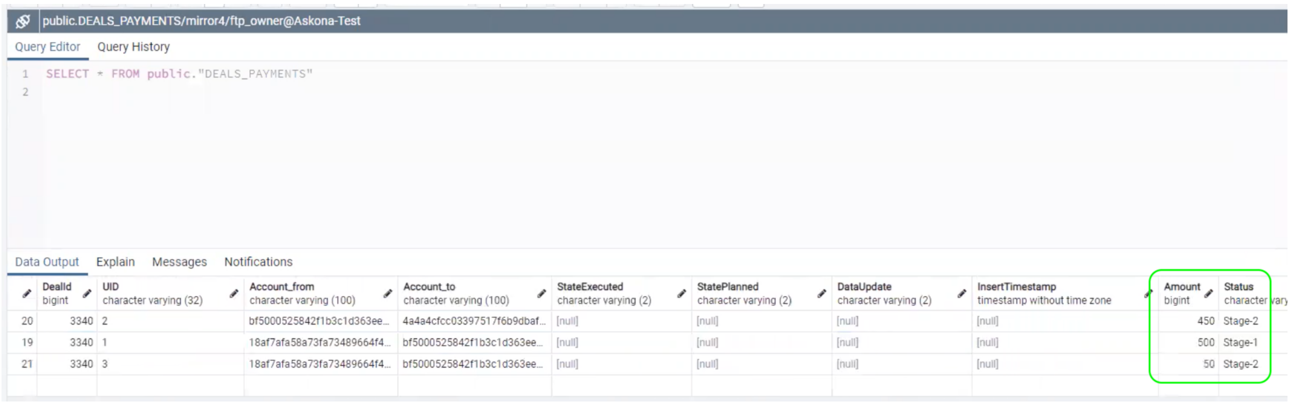
- Create accounts at nodes RBRU, STS, DOLARAMA. (RBRU - Bank; STS, DOLARAMA - clients)

- Update balance in accounts of STS and DOLARAMA. Let's say +1000 (initialisation of accounts). RBRU remains ZERO.

- Create the deal of RBRU, STS and DOLARAMA on node STS

- Load these payments to deal:
```
      - STS  -> RBRU     +500 at status "Stage-1"
      - RBRU -> DOLARAMA +450 at status "Stage-2"
      - STS  -> RBRU      +50 at status "Stage-2"
```

- Set state of deal to "Stage-1" (from "New") by STS
- Set state of deal to "Stage-2" (from "Stage-1") by DOLARAMA
- Check accounts.'balances' feild in table DEALS_ACCOUNTS at each nodes RBRU, STS, DOLARAMA.


public.DEALS_PAYMENTS/mirror4/ftp_owner@Askona-Test

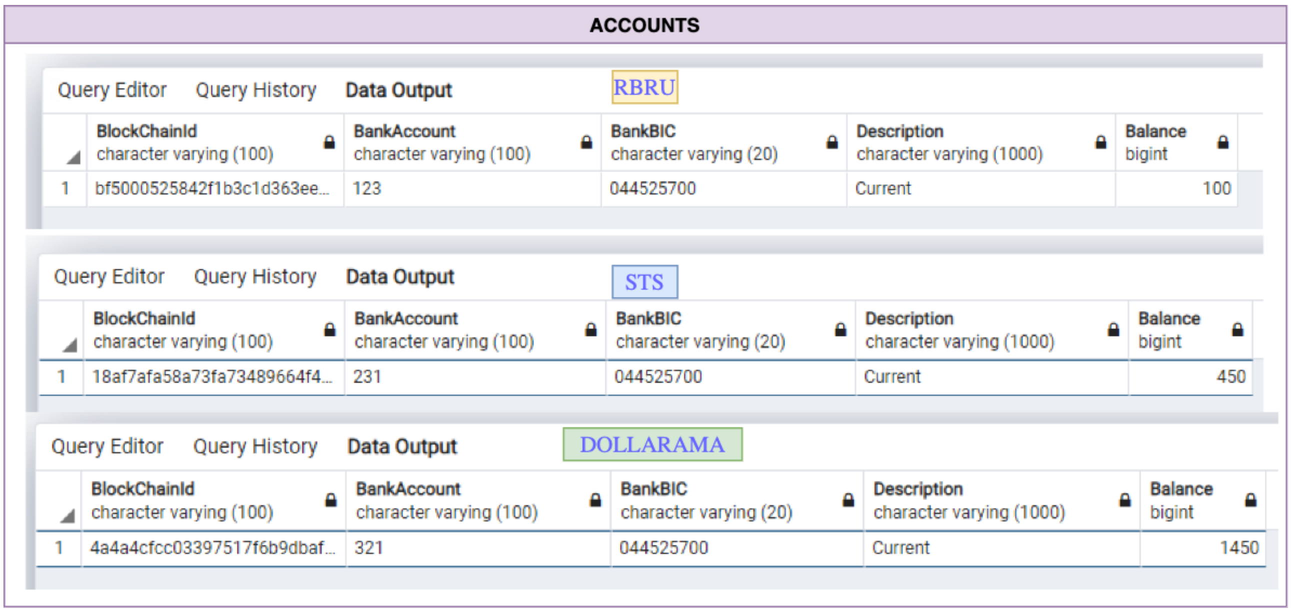
Query Editor | Query History

```
SELECT * FROM public."DEALS_PAYMENTS"
```

Data Output | Explain | Messages | Notifications

	DealId bigint	UID character varying (32)	Account_from character varying (100)	Account_to character varying (100)	StateExecuted character varying (2)	StatePlanned character varying (2)	DataUpdate character varying (2)	InsertTimestamp timestamp without time zone	Amount bigint	Status character vary
20	3340	2	bf5000525842f1b3c1d363ee...	4a4a4cfcc03397517f6b9dbaf...	[null]	[null]	[null]	[null]	450	Stage-2
19	3340	1	18af7afa58a73fa73489664f4...	bf5000525842f1b3c1d363ee...	[null]	[null]	[null]	[null]	500	Stage-1
21	3340	3	18af7afa58a73fa73489664f4...	bf5000525842f1b3c1d363ee...	[null]	[null]	[null]	[null]	50	Stage-2

Figure 10. Payments


ACCOUNTS

Query Editor | Query History | Data Output — RBRU

	BlockChainId character varying (100)	BankAccount character varying (100)	BankBIC character varying (20)	Description character varying (1000)	Balance bigint
1	bf5000525842f1b3c1d363ee...	123	044525700	Current	100

Query Editor | Query History | Data Output — STS

	BlockChainId character varying (100)	BankAccount character varying (100)	BankBIC character varying (20)	Description character varying (1000)	Balance bigint
1	18af7afa58a73fa73489664f4...	231	044525700	Current	450

Query Editor | Query History | Data Output — DOLLARAMA

	BlockChainId character varying (100)	BankAccount character varying (100)	BankBIC character varying (20)	Description character varying (1000)	Balance bigint
1	4a4a4cfcc03397517f6b9dbaf...	321	044525700	Current	1450

Figure 11. Results

This is SQL script for all steps above.

This demo scenario gives us the result `figure` 11 that we expected before. It is a signal of success in this project. Payment queue handling is illustrated in this demo testing. The status of payment have functions of prioritisation, holding and cancellation tools. Let's recap what our requirement of this mechanism were. We certainly proved that our virtual settlement mechanism meets the functional requirements as we stated in section 3.1. The non-functional are also met since the platform now supports these characteristics, moreover the new operations have not introduced a visible delay in the work.

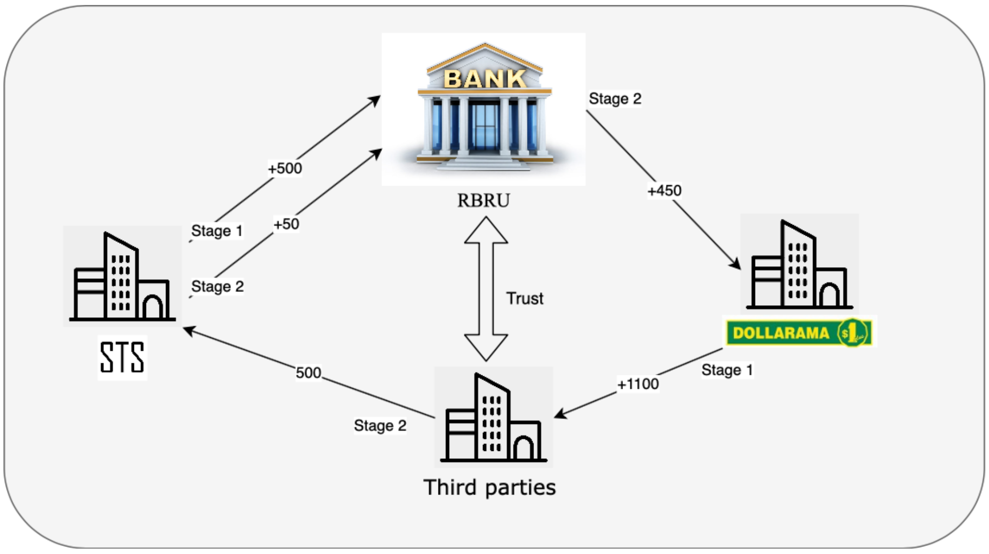


Figure 12. Demo Scenario 2

So now, if we add a new trusted participant "Third parties" into the previous scenario, there will be four nodes in our network. (see `diagram` 12) We have new list of payments in our deal:

```
- STS            -> RBRU           +500 at status "Stage-1"
- DOLARAMA       -> Third parties +1100 at status "Stage-1"
- RBRU           -> DOLARAMA       +450 at status "Stage-2"
- STS            -> RBRU            +50 at status "Stage-2"
- Third parties  -> STS            +500 at status "Stage-2"
```

It is noteworthy that the second payment will cause a deficit in DOLARAMA's balance. Since our system accept negative balance, such a gridlock scenario won't be a problem.

Table 4. Transactions

Participants:	RBRU	STS	DOLARAMA	Third parties
Starting Balance	0	1000	1000	1000
T1	+500	-500		
T2			-1100	+1100
T3	-450		+450	
T4	+50	-50		
T5		+500		-500

9. Conslusion

Overall, in this project, we dealt with the possible application of blockchain technology to business process platform development. We went through everything that we initially planned.

- The implementation of Account smart-contract and Deal smart-contract.
- Design and Creation of Account table and Payments table in R-Chain Data Base.
- Develop these operations and events in R-Chain adapter.
  - `LoadPayments()`
  - `AddAccount()`
  - `BalanceAccount()`
  - `ApprovePayment()`
- Deploy R-Chain adapter with LoadPayment, AddAccount, BalanceAccount to test network
- Create and run test scenario for Payments - SQL scripts:
  - Create Deals
  - Load Payments for Deal
  - Confirm Payments
  - Change Deal State one or more times -> Payments was triggered and executed
  - Check Balance of participants' accounts.

Without a doubt, the pratical application of D-apps and blockchain technology is still in the early stage of exploration. Since it is not widely used and accepted, there is no example or a general approach about this corporate settlement mechanism. However, the blockchain technology has brought us a huge room for imagination and innovation. During the study of this field, we realised the potential possibilities of it, thus we made a prediction for the future trend of this virtual settlement usages. Traditional centralised applications use blockchain technology to reduce costs and imporve efficiency. For example, the use of blockchain technology for bookkeeping between banks enhance the efficiency of settlement. Such a blockchain system is operated and maintained by a traditional centralized unit. That is, applications based on consortium chains and private chains, from the perspective of user use, may not be much different from centralized apps. Therefore, the uses of D-apps and blockchain technologies can completely diversified for different business models.

Links of Reference materials

[1] Nrayin Prusty Building Blockchain Projects — © 2017 Packt Publishing

[2] Project Ubin: Central Bank Digital Money using Distributed Ledger Technology — https://www.mas.gov.sg/schemes-and-initiatives/project-ubin https://github.com/project-ubin

[3] Information about R-chians — https://holdex.io/x/rchain/about https://github.com/rchain https://github.com/rchain-community

[4] Remix documentation — https://remix-ide.readthedocs.io/en/latest/

[5] Solidity documentation — https://docs.soliditylang.org/en/v0.7.4/

[6] C++ documentation — https://en.cppreference.com/w/

[7] SQL documentation — https://dev.mysql.com/doc/

[8] pgAdmin4 documentation — https://www.pgadmin.org/docs/

[9] Blockchain — https://www.ituring.com.cn/book/tupubarticle/29698