1. Digital Bank Requirements

1.1. Introduction

This Digital Bank is all digital and gives the customer the opportunity to enjoy everyday bank assignments digitally.
The Bank has no branches and operates entirely digitally.
The Bank’s vision is to make the customer's life easier with daily bank assignments as much as possible.
The Bank’s objective is to convert every-day bank frontal assignments into easy handling digital assignments, while making profit out of it.
This objective will give the customer the ultimate digital experience while doing it remotely without the necessity to visit the branch.

1.2. Project Purpose

The business purpose of our Digital Bank is to make it easier for our customers to do bank assignments remotely instead of physical meetings.
We speculate that the problem this digital bank is going to solve is associated with time and comfort.
Today people use their time very precisely, hence a basic every-day assignment in the bank can be a struggle just because they need to visit the branch at specific times that not every-one can manage to come.
That's why our Digital Bank is saving up customer’s time with only a couple of buttons away instead of missing a day at work.

1.3. Project Scope

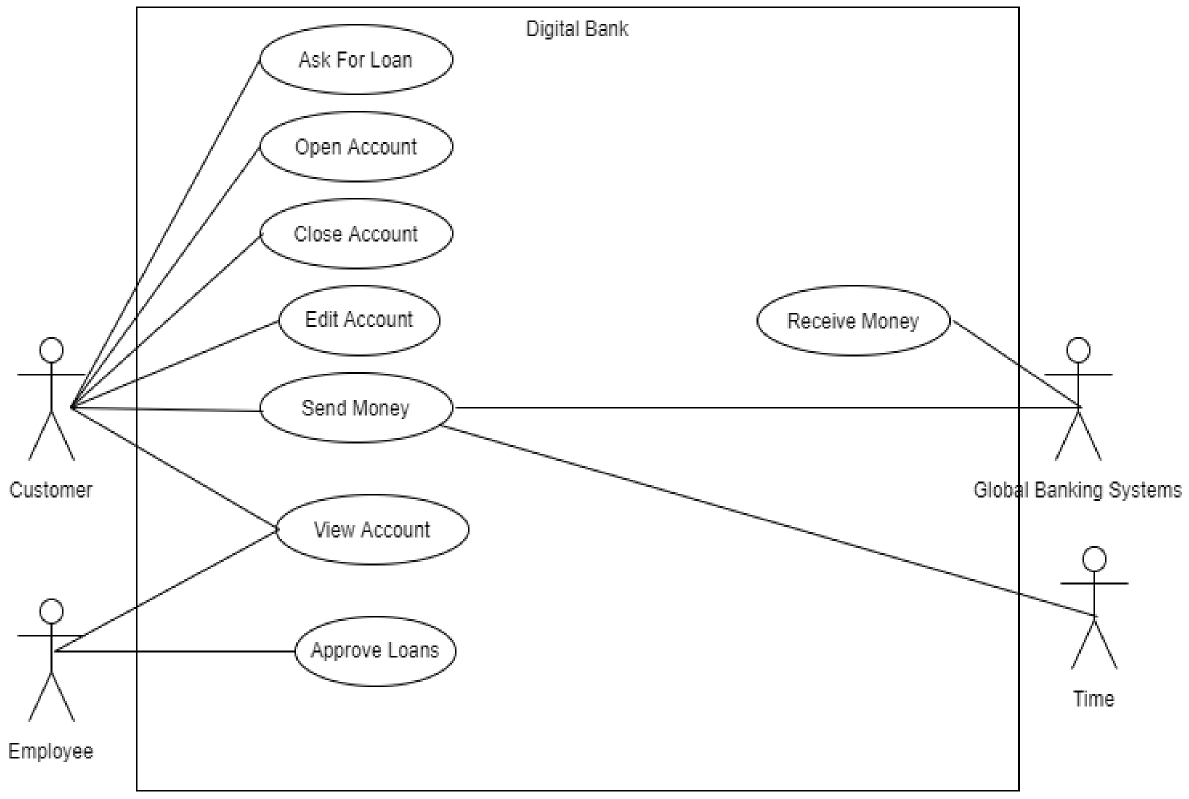
This Digital Bank is a basic digital bank.
The Bank includes customers and employees.
The customer can open/view/edit/close an account, transfer money and request a loan.
The employee can view the customer data and accept the customer’s loan requests.
This Bank’s scope does not include loan histogram, issuance of checks, issuance of credit cards, physical meeting arrangements with employees and stock investments...
We won’t support secure connections either (secured login, replay attacks, etc…)
The Send money function will not check if the destination account exists or if it got the money after successful transmission to the Global Banking System, but when we receive money to a non-existing account, we will return it to its source.

1.4. Actors and Goals

Actor Name	**Primary / Support**	**Description**	**Goals**
Customer	Primary	The Main audience. The Account belongs to the customer.	Manage the customer’s account (Transfer Money,Ask for loan etc.). By using this system, the customer can do everything he could have done with other physical banks, but digitally.
Employee	Support	Monitor customers accounts and Approve customer’s loans.	----
Global Banking Systems	Support	A central system that all banks are linked to, while communicating with each other about transactions between accounts.	----
Time	Support	The Time actor will initiate a functional operation every interval of time	----

1.5. Functional Requirements

1.5.1. Use Case Diagram



1.5.2. Use Case Details

1.5.2.1. <u>Open account</u>

1.5.2.1.1. **Goals:** Give the customer the option to open a new account.

1.5.2.1.2. **Participating actors:** Customer (Primary)

1.5.2.1.3. **Basic Flow:**

- **1.5.2.1.3.1.** The customer logs in to the system.
- **1.5.2.1.3.2.** The customer invokes “Open Account”
- **1.5.2.1.3.3.** The System shows the customer a registration form to fill.
- **1.5.2.1.3.4.** The customer fills his details accordingly.
- **1.5.2.1.3.5.** The System checks that all mandatory fields are filled and all fields are properly filled.
- **1.5.2.1.3.6.** The System transfers the customer 1000$ from the bank's account.
- **1.5.2.1.3.7.** The system creates a new customer account.
- **1.5.2.1.3.8.** The customer is redirected to the bank main menu (after login)

1.5.2.1.4. Alternate Workflow:

1a. The Customer didn't sign up yet.
1a1. The customer signs up.
1a2. The system asks for credentials details.
1a3. The customer enters credentials details.
1a4. The flow continues at step 2.

5a. The customer didn't enter data to a mandatory field.
5a1. The System alerts the customer which mandatory field needs correction.
5a2. The flow continues at step 3.

5b. The customer entered invalid data to one of the fields.
5b1. The System alerts the customer which mandatory field needs correction.
5b2. The flow continues at step 3.

1.5.2.2. <u>View Account</u>

1.5.2.2.1. Goals: Give the customer/employee the option to view the/a customer's account details: personal data, balance, latest actions and all actions.

1.5.2.2.2. Participating actors: Customer (Primary) or Employee (Primary)

1.5.2.2.3. Basic Flow:

1.5.2.2.3.1. The customer logins to the system.
1.5.2.2.3.2. The system shows the customer's accounts available.
1.5.2.2.3.3. The customer enters the account to use.
1.5.2.2.3.4. The customer invokes a view account option from the menu.
1.5.2.2.3.5. The system prints to the customer basic details and lists available options to view.
1.5.2.2.3.6. The customer chooses his wanted option.
1.5.2.2.3.7. The system shows the customer his chosen data.

1.5.2.2.4. Alternate Workflow:

1a**.** The employee logins to the system.
1a1. The system lists the employee all the customers account number.
1a2. The employee invokes the customer account number to view his account options.
1a3. The system shows the available options to view: customer account details, account balance, latest actions and all actions.
1a4. The employee chooses his wanted option.
1a5. The system shows the employee his chosen data

1a. The customer enters an invalid username or password.
1a1. The System sends the customer an error message
1a2. The flow is terminated.

6a. The customer enters an invalid input.
6a1. The system shows the customer an error message.
6a2. The flow stops.

1.5.2.3. <u>Edit Accounts</u>

1.5.2.3.1. Goals: Give the customer the option to edit his account details

1.5.2.3.2. Participating actors: Customer (Primary)

1.5.2.3.3. Basic Flow:

- **1.5.2.3.3.1.** The customer logins to the system.
- **1.5.2.3.3.2.** The system shows the customer's accounts available.
- **1.5.2.3.3.3.** The customer enters the account to use.
- **1.5.2.3.3.4.** The customer asks to edit his account details.
- **1.5.2.3.3.5.** The system shows the customer an edit menu with all available details to be edited.
- **1.5.2.3.3.6.** The customer chooses the details he wants to edit.
- **1.5.2.3.3.7.** The system shows the customer the field name to be changed.
- **1.5.2.3.3.8.** The customer enters new data to the current detail.
- **1.5.2.3.3.9.** The system checks that the data is valid.
- **1.5.2.3.3.10.** The system saves the new data.
- **1.5.2.3.3.11.** If there is another detail to edit the flow returns to step 3.4
- **1.5.2.3.3.12.** The system shows all the new customer's details.

1.5.2.3.4. Alternate Workflow:

1a. The customer enters an invalid username or password.
1a1. The System sends the customer an error message
1a2. The flow is terminated.

4a1. The system notifies the customer that he did not enter an option.
4a2. The flow is terminated.

4b. The customer enters an invalid input.
4b1. The system notifies the customer that he did not enter an option.
4b2. The flow continues at step 3.2.

6a. The customer entered an invalid input.
6a1. The system alerts the customer that the input is invalid.
6a2. The flow continues at step 3.4.

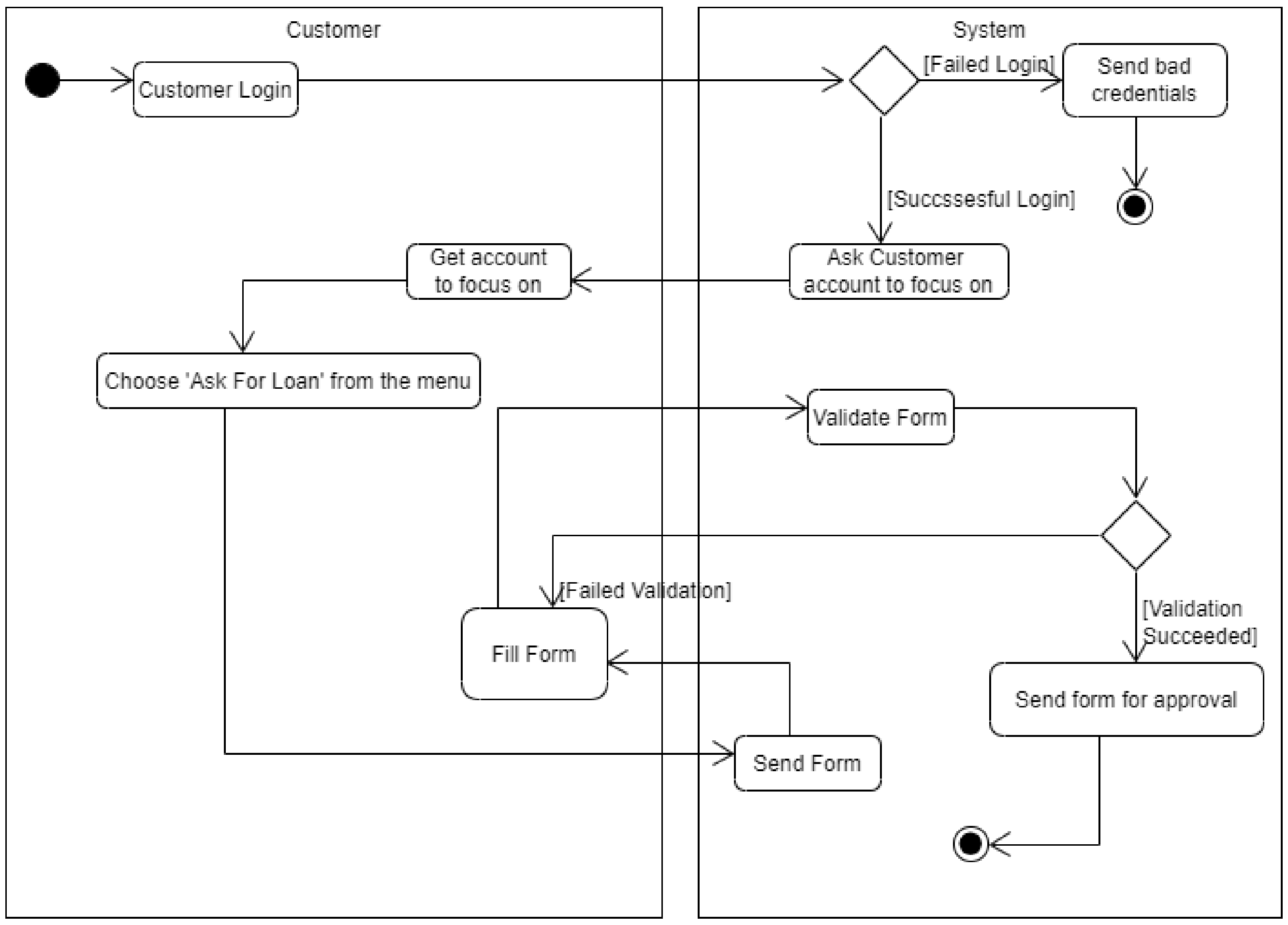
6b. The customer entered a blank input.
6b1. The system does not save new data to this detail.
6b2. The flow continues at step 3.8.

1.5.2.4. <u>Ask For Loan</u>

1.5.2.4.1. Goals: Give the Customer the option to ask for a loan.

1.5.2.4.2. Participating actors: Customer (Primary)

1.5.2.4.3. Flow:

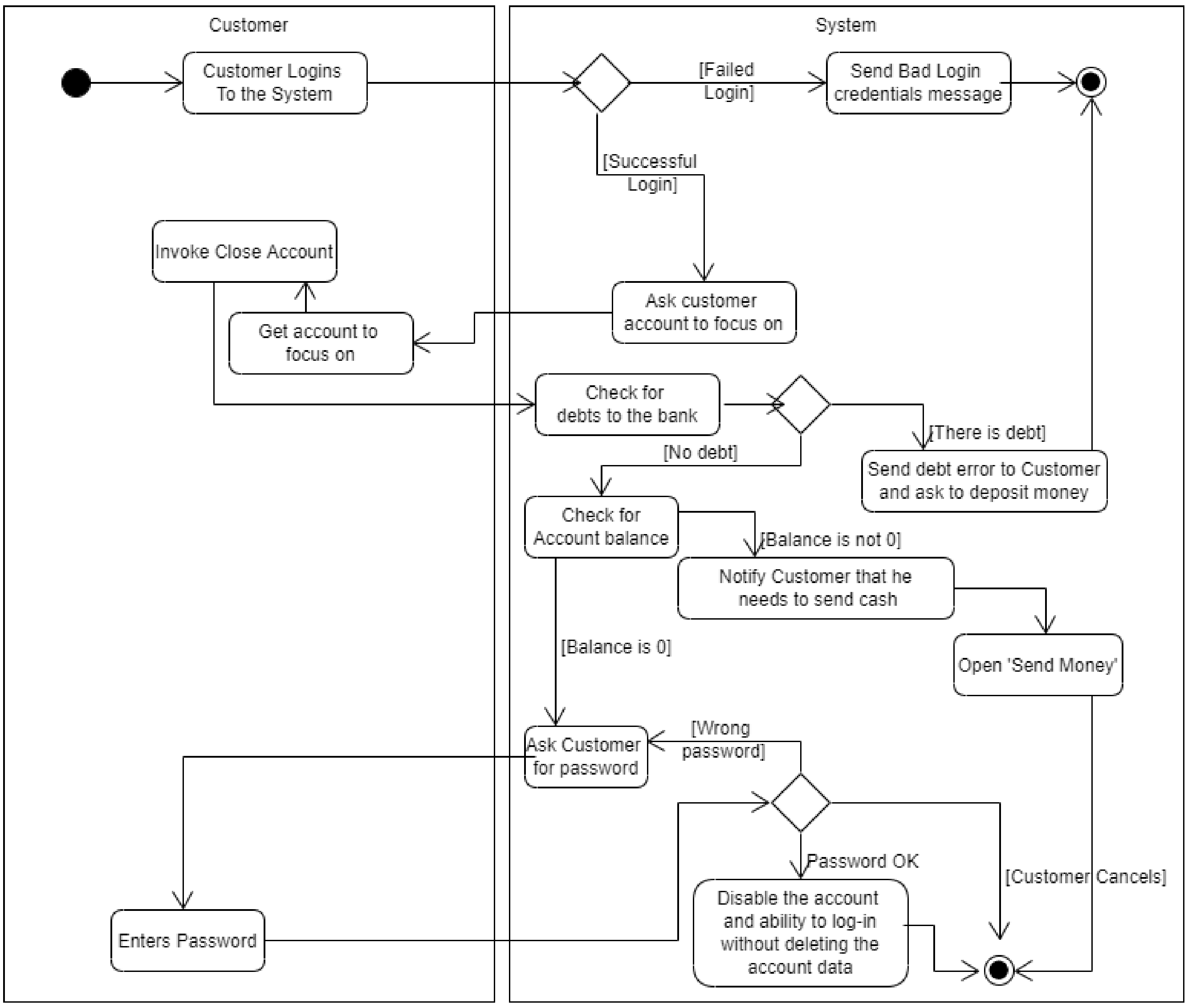


1.5.2.5. **Close Account**

1.5.2.5.1. Goals: Give the customer the option to close his account or quit the bank services.

1.5.2.5.2. Participating actors: Customer (Primary)

1.5.2.5.3. Flow:


Customer
System
Customer Logins To the System
[Failed Login]
Send Bad Login credentials message
[Successful Login]
Invoke Close Account
Ask customer account to focus on
Get account to focus on
Check for debts to the bank
[There is debt]
Send debt error to Customer and ask to deposit money
[No debt]
Check for Account balance
[Balance is not 0]
Notify Customer that he needs to send cash
Open 'Send Money'
[Balance is 0]
Ask Customer for password
[Wrong password]
Password OK
[Customer Cancels]
Disable the account and ability to log-in without deleting the account data
Enters Password

1.5.2.6. **Send Money**

1.5.2.6.1. Goals: Give the Customer/Time the option to Send money from his/a customer's account to another account in the same bank or in other banks.

1.5.2.6.2. Participating actors: Time (Primary) or Customer (Primary), Global Banking System

1.5.2.6.3. Basic Flow:

1.5.2.6.3.1. The customer logins to the system.

1.5.2.6.3.2. The system shows the customer's accounts available.

1.5.2.6.3.3. The customer enters the account to use.
1.5.2.6.3.4. The customer asks to send money.
1.5.2.6.3.5. The system asks the customer for the receiver's bank account details.
1.5.2.6.3.6. The customer enters details.
1.5.2.6.3.7. The system asks the customer the amount of money he wants to send.
1.5.2.6.3.8. The customer enters an amount.
1.5.2.6.3.9. System verifies the customer can transfer the amount of money he entered.
1.5.2.6.3.10. System issues the transfer via the Global Banking System.
1.5.2.6.3.11. System withdraws the amount of money from the customer's balance.
1.5.2.6.3.12. System saves a transfer confirmation to source customer data.
1.5.2.6.3.13. System sends an approval message to the customer.

1.5.2.6.4. Alternate Workflow:

1a. Time invokes send money.
1a1. Time sends the source and destination accounts to the system.
1a2. The flow continues at step 9, without step 12.

2b. The customer enters an invalid username or password.
2b1. The System sends the customer an error message
2b2. The flow is terminated.

4a. The customer didn't enter details into a mandatory field.
4a1. System asks the customer to enter valid details.
4a2. The flow continues at step 3.2.

7a. The customer can't afford the transfer.
7a1. The system Shows an error message to the customer indicating that he cannot make the transfer.
7a2. The flow continues at step 3.2.

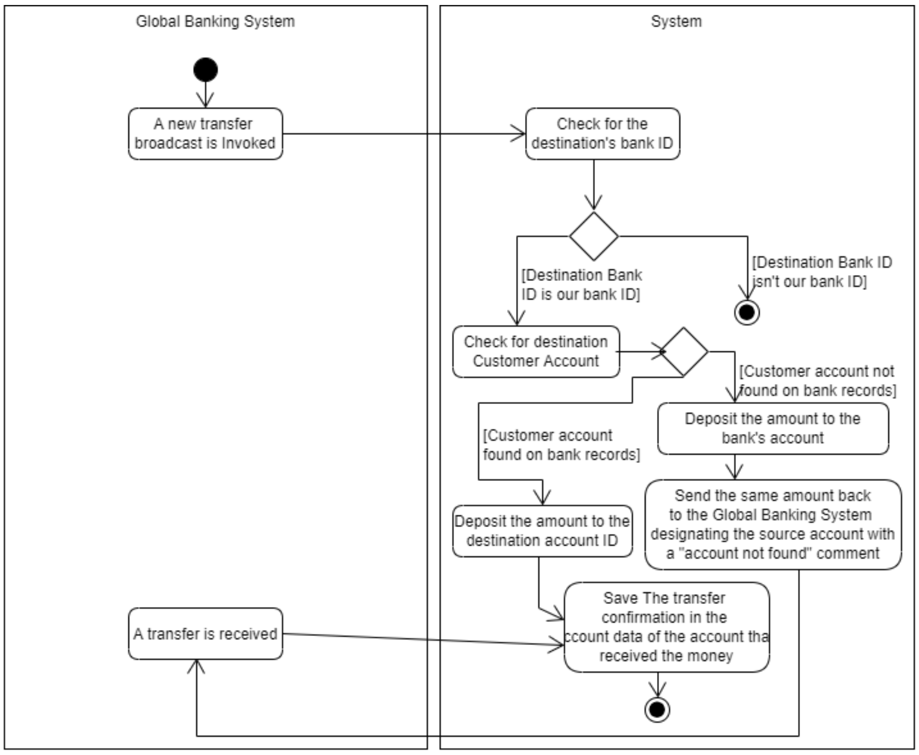
8a. The Global Banking System is not connected.
8a1. The system notifies the customer that the transfer failed.
8a2. The flow continues at step 3.2.

1.5.2.7. <u>Receive Money</u>

1.5.2.7.1. Goals: Stand by and handle Global Banking System's broadcasts of money transfers to a bank's customer according to the broadcast transfer request.

1.5.2.7.2. Participating actors: Global Banking System(Primary)

1.5.2.7.3. Flow:

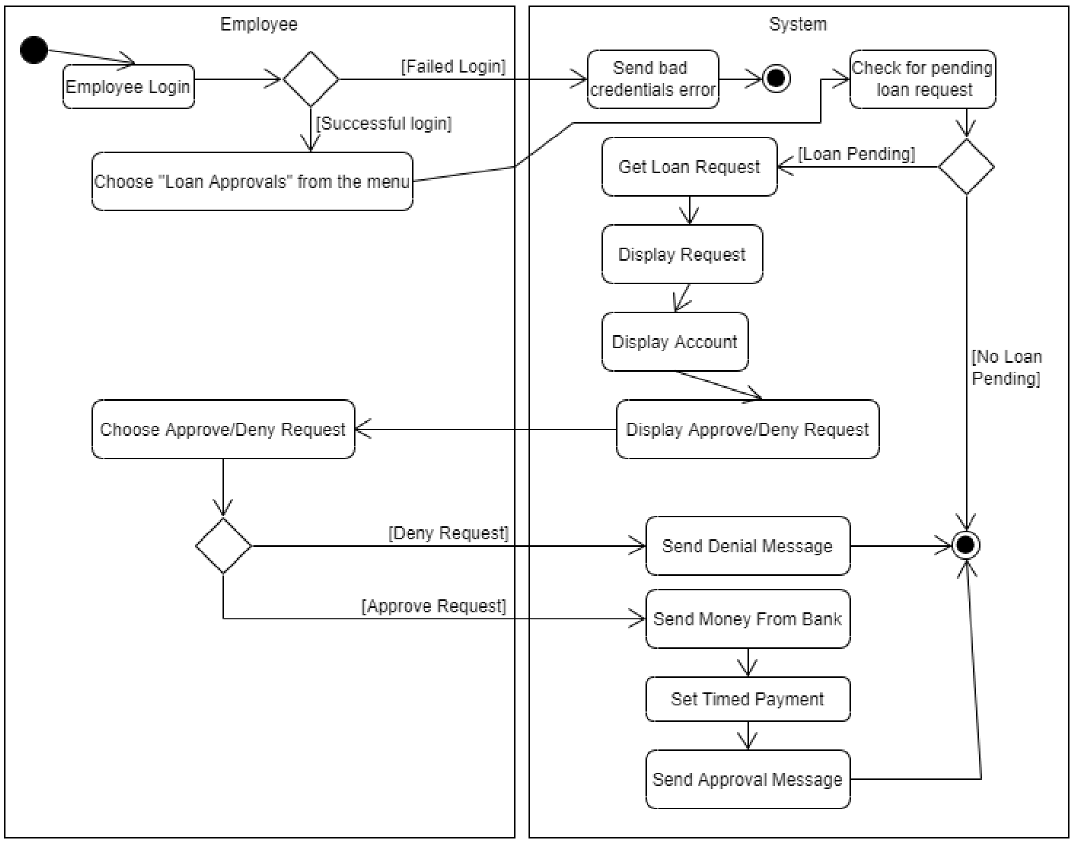

Global Banking System
System
A new transfer broadcast is Invoked
Check for the destination's bank ID
[Destination Bank ID is our bank ID]
[Destination Bank ID isn't our bank ID]
Check for destination Customer Account
[Customer account not found on bank records]
[Customer account found on bank records]
Deposit the amount to the bank's account
Send the same amount back to the Global Banking System designating the source account with a "account not found" comment
Deposit the amount to the destination account ID
Save The transfer confirmation in the ccount data of the account tha received the money
A transfer is received

1.5.2.8. Approve Loans

1.5.2.8.1. Goals: Gives the employee the option to approve or deny the loan request.

1.5.2.8.2. Participating actors: Employee (Primary)

1.5.2.8.3. Flow:

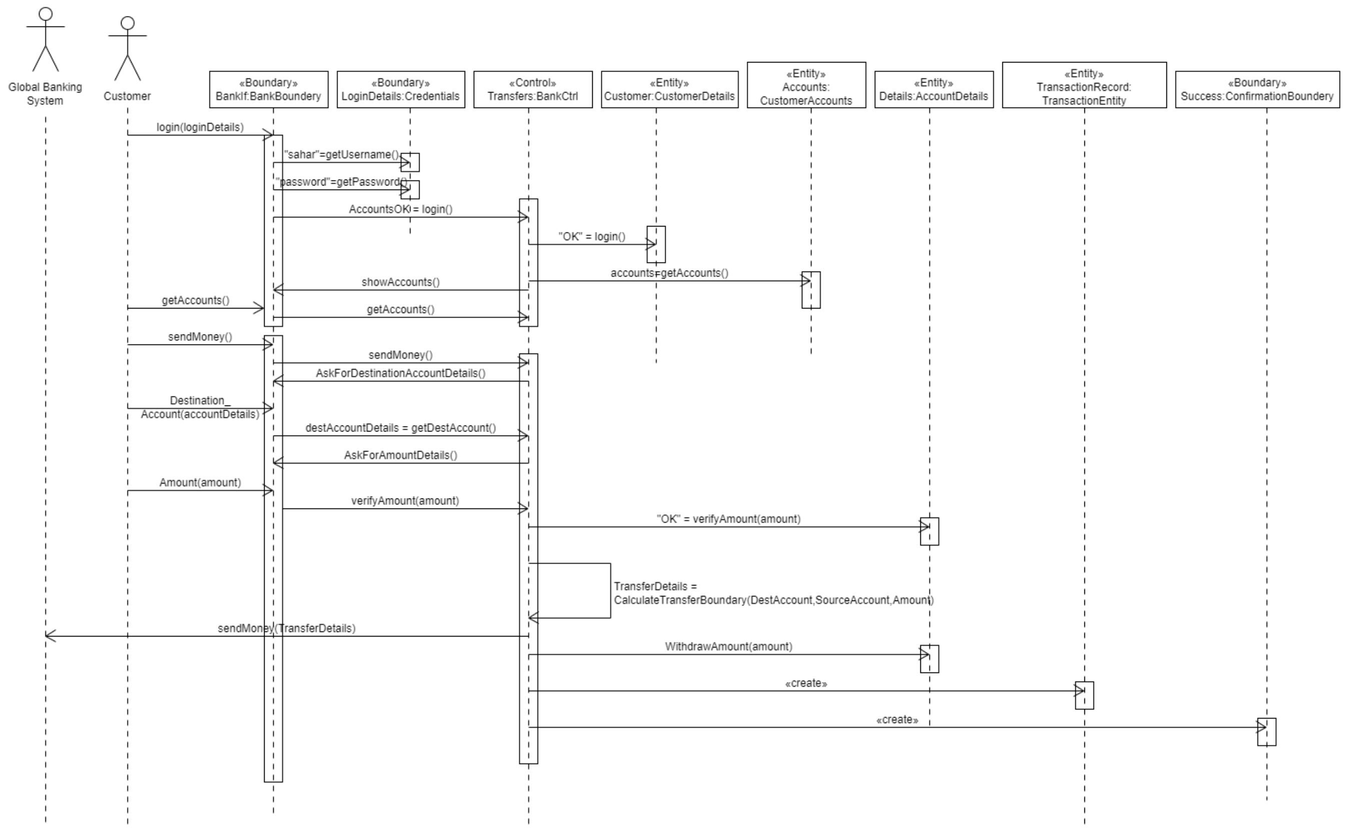


2. Digital Bank System Analysis

2.1. Dynamic Model

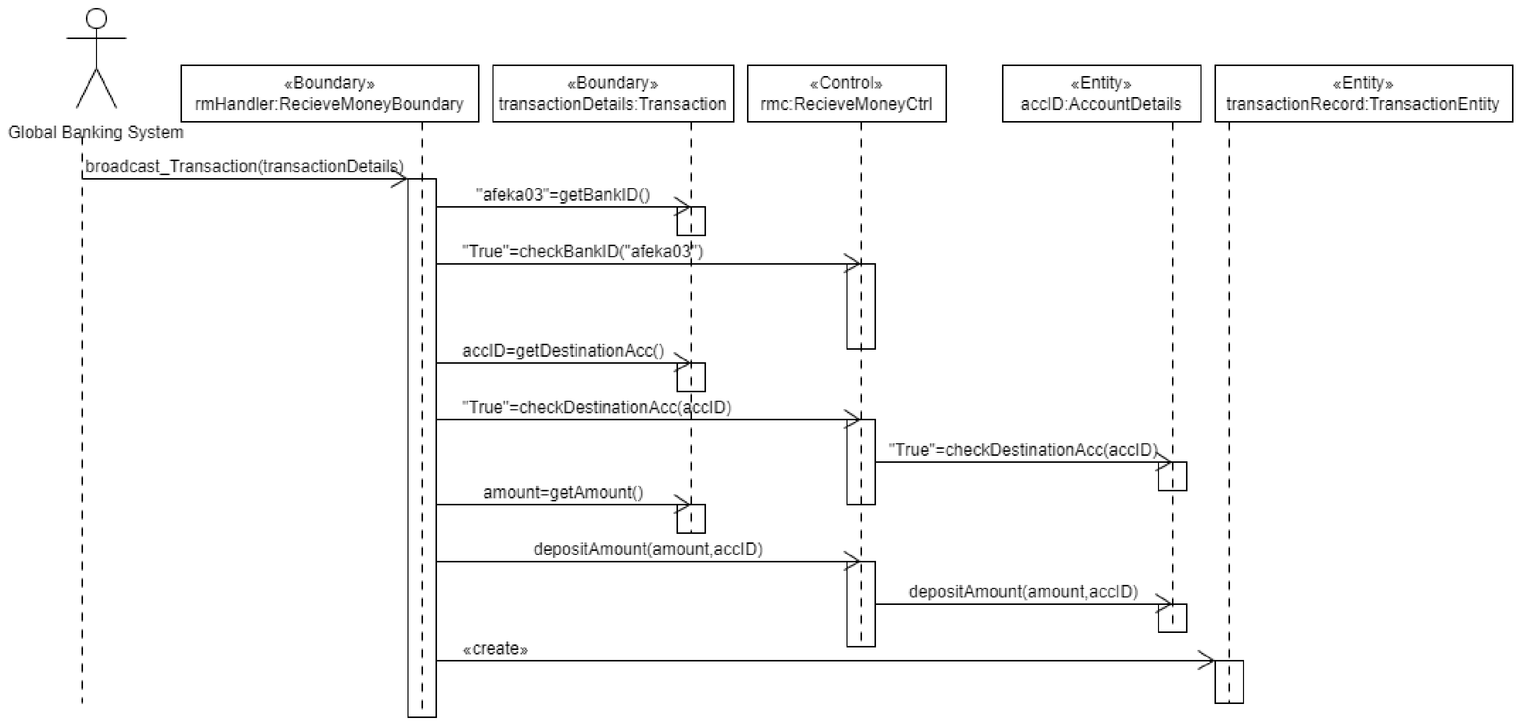
2.1.1. Sequences

2.1.1.1. Basic flow of Send money Use Case by **Sahar Hezkia**

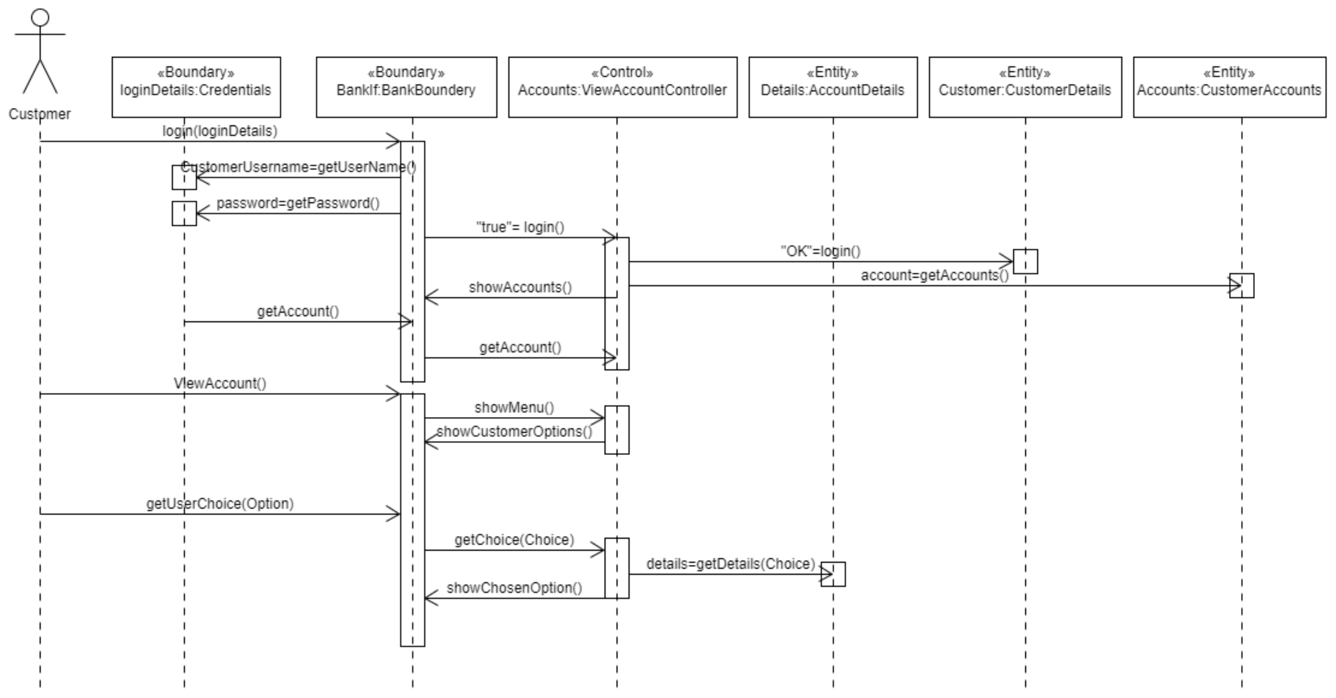


Global Banking System
Customer
«Boundary» BankIf:BankBoundery
«Boundary» LoginDetails:Credentials
«Control» Transfers:BankCtrl
«Entity» Customer:CustomerDetails
«Entity» Accounts: CustomerAccounts
«Entity» Details:AccountDetails
«Entity» TransactionRecord: TransactionEntity
«Boundary» Success:ConfirmationBoundery
login(loginDetails)
"sahar"=getUsername()
"password"=getPassword()
AccountsOK = login()
"OK" = login()
accounts=getAccounts()
showAccounts()
getAccounts()
getAccounts()
sendMoney()
sendMoney()
AskForDestinationAccountDetails()
Destination_ Account(accountDetails)
destAccountDetails = getDestAccount()
AskForAmountDetails()
Amount(amount)
verifyAmount(amount)
"OK" = verifyAmount(amount)
TransferDetails = CalculateTransferBoundary(DestAccount,SourceAccount,Amount)
sendMoney(TransferDetails)
WithdrawAmount(amount)
«create»
«create»

2.1.1.2. Basic Flow of Receive Money Use Case By **Guy Rinsky**

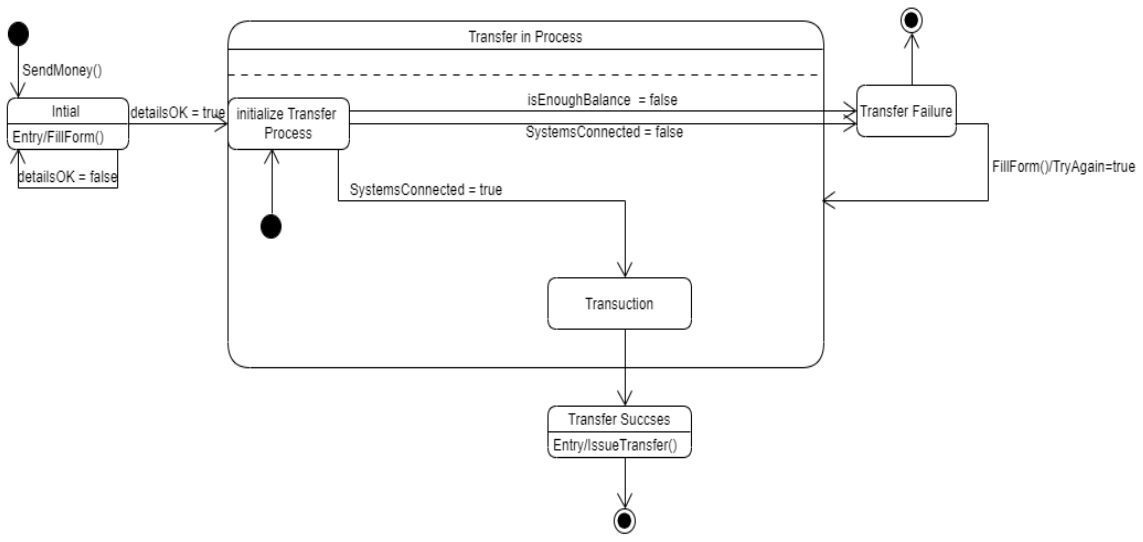

Global Banking System
«Boundary» rmHandler:RecieveMoneyBoundary
«Boundary» transactionDetails:Transaction
«Control» rmc:RecieveMoneyCtrl
«Entity» accID:AccountDetails
«Entity» transactionRecord:TransactionEntity
broadcast_Transaction(transactionDetails)
"afeka03"=getBankID()
"True"=checkBankID("afeka03")
accID=getDestinationAcc()
"True"=checkDestinationAcc(accID)
"True"=checkDestinationAcc(accID)
amount=getAmount()
depositAmount(amount,accID)
depositAmount(amount,accID)
«create»

2.1.1.3. Basic flow of View account Use Case by **Linor Ben-Yosef**

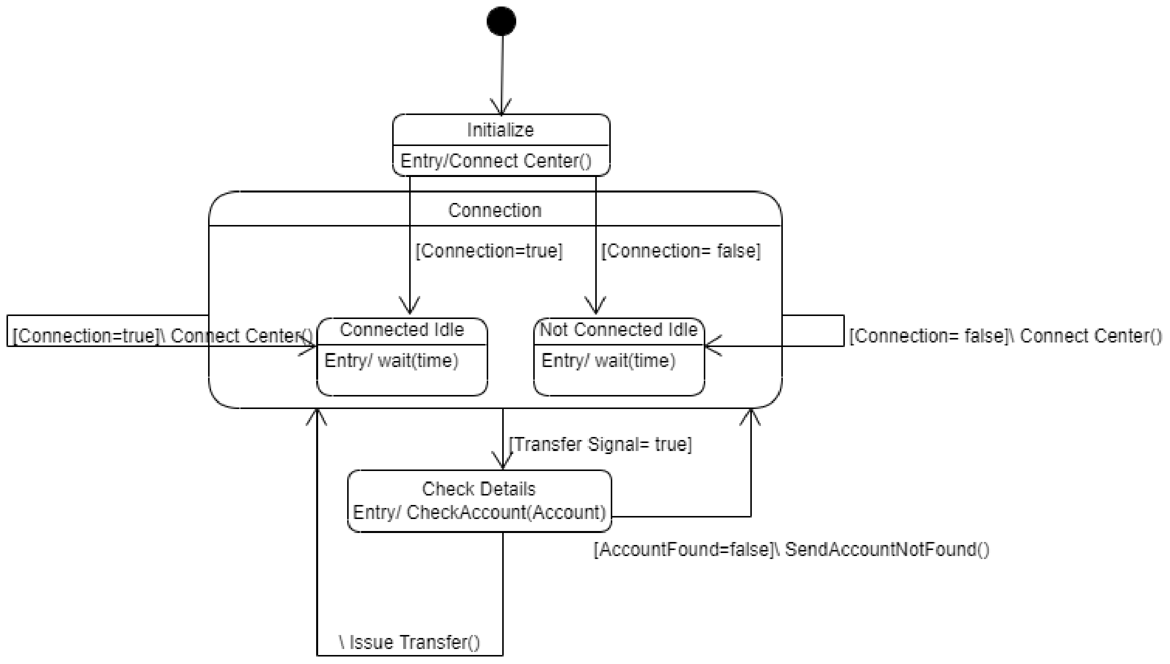

Customer
«Boundary» loginDetails:Credentials
«Boundary» BankIf:BankBoundery
«Control» Accounts:ViewAccountController
«Entity» Details:AccountDetails
«Entity» Customer:CustomerDetails
«Entity» Accounts:CustomerAccounts
login(loginDetails)
CustomerUsername=getUserName()
password=getPassword()
"true"= login()
"OK"=login()
account=getAccounts()
showAccounts()
getAccount()
getAccount()
ViewAccount()
showMenu()
showCustomerOptions()
getUserChoice(Option)
getChoice(Choice)
details=getDetails(Choice)
showChosenOption()

2.1.2. Banking Transaction Statechart

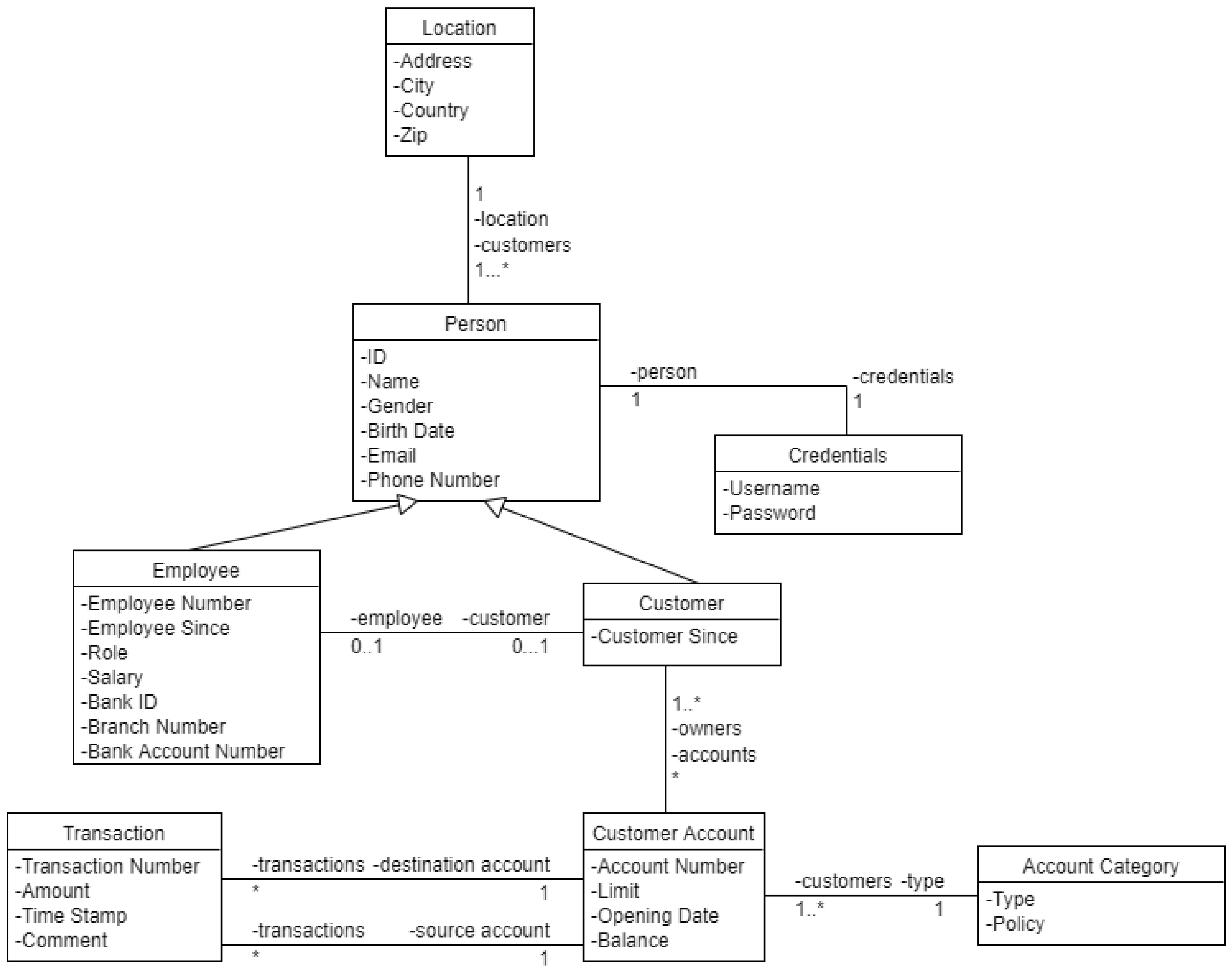
2.1.2.1. Send Money Statechart



2.1.2.2. Receive money statechart



2.2. Domain Object Model



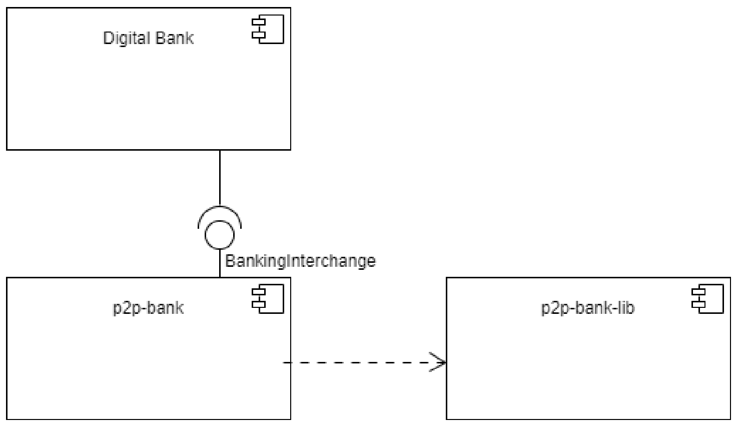
<u>NOTES:</u>**

1) Employee has bank account details because we want the bank to transfer his salary automatically, while the employee's bank account is not necessarily in our bank.
2) Loans are a private case of the transfer entity, implemented via timed transfers and recorded in the comment that it is a loan(which part is paid of the loan).
3) Customer logs in to the system with his personal ID, while employees login to the system with their employee number.
   A customer can have a couple of accounts while an employee can have only 1 employee account (this is because we want to separate between the accounts in case of an employee that has an employee account and a customer account in the bank).

3. Digital Bank System Design

3.1. Subsystem Decomposition

3.1.1. Component Diagram



3.1.2. Components Description

Inter-banking communication channel - a mediating infrastructure that enables the bank to communicate with other banks and invoke inter-banking transactions, for funds transfers between accounts in different banking projects.

3.2. System Architecture

Peer-to-Peer - the Digital Banking projects communicate via the Inter-banking communication channel that enables a Peer-to-Peer architectural pattern implementation. Each banking system works independently of the other teams' implementations, and it can communicate asynchronously whenever it requires to transfer funds to accounts in other systems or when it receives funds from other systems.

3.3. Persistent Architecture

As this is a prototype, the storage and retrieval of the entities in our system will be handled by local files on the host computer. Optimally, We would prefer to store the data with SQL DB.

3.4. System Installation

1. **Kafka-** is a P2P infrastructure that enables applications to communicate through it asynchronously.
2. **Java- JDK 1.8.2**
   JDK- includes a complete JRE plus tools for developing, debugging, and monitoring Java applications

Instructions

- First make sure you have JDK version 8 or higher installed.
- Download the KAFKA installation file and unzip it to a folder, such as D: \ kafka

    WINDOWS users: Please note that the PATH of the folder **does not** contain spaces or letters in Hebrew. In particular, do not retrieve the folder directly to DESKTOP.

    - ❖ Recommended to unpack the installation files using WINZIP, WINRAR or 7ZIP software.
    - ❖ Note after the first extraction, may appear a file with a TAR extension - in this case, extract this file also to complete the installation.
- After installation, **MUST** run the following programs according to the following instructions:

Open a command prompt and go to the KAFKA installation folder that you shared - make sure it contains the bin folder.

Check which JAVA version is installed on your pc.

- ■ Run the following command in command prompt:

```
java -Version
```

○ **Running ZOOKEEPER**

WINDOWS users

- ■ Run the following command:

```
bin\windows\zookeeper-server-start.bat config\zookeeper.properties
```

LINUX or MAC users

- ■ Run the following command:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

○ **Running KAFKA**

- ■ Open another command prompt window and go to the KAFKA installer folder you shared - make sure it contains the bin folder.

WINDOWS users

- ■ Run the following command:

```
bin\windows\kafka-server-start.bat config\server.properties
```

LINUX/MAC users

- ■ Run the following command:

```
bin/kafka-server-start.sh config/server.properties
```

- **Now that the KAFKA infrastructure is running, open the eclipse framework and import the zip project file.**
- **download the “p2p-bank-components-3.0.0” dependencies file from moodle and import them to the eclipse project:**
  - **Right click on the project**
  - **Select “Build Path”**
  - **Click on “Configure Build Path…”**
  - **Select the “Libraries” tab.**
  - **Select “Classpath”.**
  - **Click on the “Add external JARs...” button.**
  - **select the Jar files from “p2p-bank-components-3.0.0”**
  - **Click Apply and close the Build Path configuration window.**
- **Run the code**

1.6. Non Functional Requirements

Requirement Number	Requirement Type (U / R / P / S)	Description	Suggested tests
1	Usability	Support English language	
2	Supportability	Must have JDK installed on the host computer.	
3	Supportability	Must have Kafka communication system to make transfer money transactions	