Mobile banking with Pay-Pal

By

Nhlakanipho Ndlamlenze

A project report submitted in partial fulfillment
of the requirements for the degree of

B.SC. Computer Science (Hons)

University of the Western Cape

2012

Date: Nov 23

UNIVERSITY *of the*
WESTERN CAPE

University of the Western Cape

MOBILE BANKING WITH PAY-PAL

By

Nhlakanipho Ndlamlenze

Supervisor: Professor Isabel Venter

Department of Computer Science

The system will allow the mobile phone users including those without bank accounts to use and benefit from mobile banking.
If a Pay-Pal account is shared by multiple mobile phone users, each user can pay a third party by sending a Short Messages Service (SMS) message to a dedicated server. A server program will wait for incoming SMS messages from the mobile phone user, interpret the message and initiate third party payment service from the mobile phone users Pay-Pal account on the Pay-Pal server, if the user's account has the necessary funds for the payment.
All users will have to create their accounts to hold their balance amount on the system. The sum of all users’ balance is stored as a single amount in the multi-shared Pay-Pal account.

TABLE OF CONTENTS

Table of Contents

LIST OF TABLES

Index of Tables

ACKNOWLEDGMENTS

Professor I Venter, from Computer Science Department at UWC provided a great effort on the success criteria of this document compilation. She has been editing each and every sentence on this document and also providing the supervision of the whole process of requirements analysis.
Students from UWC have given us their valuable time to respond on our research questions during the research survey of the project user requirements.

GLOSSARY

SMSC: Short Message Service Center allows mobile phone users to send an SMS. It is provided from all Mobile Network Operators.
SMPP: Short Message Peer-to-Peer delivers messages from SMSC Agent like Mobile Network operator to a dedicated application program.
SPAM: Flooding of many copies of the same messages in attempt to force a message to multiple people. It is mostly used for commercial related tasks such as advertising and more.
Pay-Pal: A mobile banking System that provides a capability to share money on-line.
API: Application Programming Interface (API) allows programmers to write programs to command a System to perform certain functions.
NVP: The Name-Value Pair (NVP) API provides parameter-based association between request and response fields of a message and their values. The request message is sent from a website by the API, and a response message is returned by PayPal using a client-server model in which the site is a client of the PayPal server.
HTTPS: This is HTTP protocol that includes transport security layer (TSL) that promotes encrypted that exchange on the Network Transport Layer.
SMS: Is a service component from a mobile or computer device used to allow exchange of text messages.
BLUE-TOOTH: A wireless network that enables devices to share data at short range.

CHAPTER 1

THE USERS REQUIREMENTS DOCUMENT

Purpose

This document describes the user requirements for a project that explores the development of a system for mobile banking by means of SMS messaging. The system will be useful for persons without bank accounts when they want to pay a third party.

Requirements

A mobile phone user will send an SMS message containing a text such as: “send 5000 to 234567 (Pay-Pal account) from 0721655148 (Mobile number)”. A response SMS will require a user to provide necessary authentication details and etc. Each user cannot transfer an amount more than his/her account balance.

All of the users’ balance amounts are summed up and stored as a single Pay-Pal account balance. During the user transactions, Pay-Pal will transfer money from this shared account to a specified third party, on response to an SMS text that has been sent by a user. If the transactions are possible both the money sender and receiver will be notified by an SMS and the sender’s account balance will be reduced by the transferred amount.

Specific Requirements

- A user must be able to send money to a third party by means of SMS without the need of a bank account.
- All users should be able to send money from the combined account (Pay-Pal account or Bank account) to their mobile phone numbers' accounts.
- A user must be able to check their account balance statements and update their personal information using the SMS.
- Any user should be able to cancel his/her account.

User Requirements Survey Report

Closed Questions

Question	Description	Respondents	A	B	C	Mean	Analysis
1	Use cellphone banking (A)YES or (B)NO	9	5	4	-	0.55	55% of people uses cell-phone banking
2	Frequently performed Cell-phone banking tasks: (A) To pay the third pay party or (B) other	9	7	2	-	0.77	77% of mobile-phone banking users, use it for paying third party
3	Would not make a bank account if there was an alternative banking option: (A)Yes or	20	14	8	-	0.70	70% of people would not make a bank account if there was an alternative banking

	(B) No						option
4	Suffer from Internet connection due to their geographical area: Frequently(A), Sometimes(B), Never(C)	13	6	5	2	0.46	46% on average has Internet connection issues on their environmental locations
5	Usually receives SMS messages delivery failure messages: Yes(A) or No(B)	20	1	19	-	0.01	10% on average faces SMS message delivery failure
6	Familiar with SMS messaging application on their mobile-phone: Yes(A) or No(B)	21	18	3	-	0.85	85.07% on average people can use SMS messaging application
10	Think the system will	17	5	16	-	0.29	29.04% on average

	be beneficial only to the people without bank accounts and mobile web technology: Yes(A) or No(B)						thinks the system will not benefit everyone

Findings: Based on the open questions included on the appendix of this document, it is concluded that: People find it a complex task to open a bank account due to the number of documents to validate their personal details. This makes us an opportunity to develop a mechanism that which we can use to validate a user address and proof of student registration, or employment.

End User Requirements

The end user will only interact with the system by means of SMS.

Only, the administrative user who can access the system through the web services.

User can only send a specific text otherwise receive an error message that include help hints for options.

Paying a third party:
SEND {amount(either a character R can be include or not)}
TO {receipted(either a mobile or a pay-pal account number)}
FROM {only the mobile number}
Example: **SEND** 800 **TO** 0788700707 **FROM** 0791655148
Request the account balance:
BALANCE FOR {mobile number }
Example: **BALANCE FOR** 0791655148
Updating account details:
CHANGE {content}
TO {new value}
Example: **CHANGE** password **TO** mynewpassword22#
Authentication:
Example: mynewpassword22#

Conclusion

The system will solve the problem for a user who do not have a bank account when they need to pay a third party. That will benefit including a user that does not have a mobile web technology on their mobile-phone.

CHAPTER 2

THE REQUIREMENTS ANALYSIS DOCUMENT

Introduction

The system will be of use to all mobile phone users. They can use and benefit from Internet money transfer system if their mobile phones support SMS messaging technology.
The brief scope of the project: involves requirements analysis and implementation, testing, source code repository and the user manuals creation will come towards the end of this project.
The system objectives: is to allow people without bank accounts to pay the third party and the success criteria depends on the availability of time, sponsorship and availability of developer(s) during implementation process.

Current System

Mobile banking systems are currently advantageous when mobile phone users have mobile web technology on their cell-phones.

A statistical survey completed at the beginning of this project suggest that, most people cannot enjoy using the current INTERNET banking systems due to their poor or no Internet access on their geographical areas. It is also recognized that, the SMS banking services provided from the Banking Institutions does not solve the problem due to the shortage number of the banking ATM to withdraw their cash.

Besides the INTERNET problems, a lot of users cannot tolerate with the idea of supplying the hard copies of their personal details, such as residence address, pay-slip, proof of student registration or employment and etc, whenever they need to open the bank accounts.

Proposed System

Users will benefit from mobile banking when they have mobile-web technology or not. This solves a problem aroused from a number of users who find it necessary to include their mobile web non-supported devices.

Almost every individual who can need to do banking can send SMS message using their mobile-phone. That will be beneficial due to the case that, almost every geographical location has less issues accessing their base station of a mobile network. As a results, the network operator's SMSC can be accessed at almost anywhere without the mobile web browser on the mobile device.

Users can receive money to their accounts from others or the third parties. They will not need to withdraw cash from the ATM to pay third

party.

More-over, a user that cannot open a bank account can still pay the third party without the need of opening one. The users can also get paid through the system when they supply their mobile numbers to their employees. Where case matters, a member from a certain bank can transfer money to the system a swell when they provide a specific mobile phone number.

Functional Requirements

The end users will send and receive messages through their current mobile network operator.

The system will have a mobile-phone that has a cellphone number which is registered with a specific mobile network operator, which will allow it to send and receive SMS messages. Such that, when our system mobile-phone receives an SMS message it will automatically forward it to the system server. The system will respond to users by sending a text messages, via blue-tooth, to the mobile-phone. Then, the mobile-phone will send them via mobile network operator's SMSC as SMS messages, to the end users.

Every user must have a few records, keeping their accounts information, including account balance on a database. A Pay-Pay account is created to store the total amount of every user's account balance. This account is used for third party payment options through Pay-Pal for the users. A user paid the third party will have his/her account balance reduced by the total amount used from Pay-Pal

account to pay the third party. Users will be notified by SMS on any matter taking place on their accounts, such as transactions failures and etc.

The system will access the Pay-Pal server through Name-Value Pair (NVP), an API, provided from Pay-Pal. Our client program will initiate third party payment transactions process, on Pay-Pal server program, when the system needs to pay a third party using the account that is shared by mobile-phone users.

A mobile number is used to read and write messages on a mobile network operator's SMSC to and from users from and to our server

Non-Functional Requirements

Usability
The enjoyable user experience will be ensured by the reliability of the SMS messages delivery. As results, the transactions will be guaranteed.

Reliability
SMS message delivery is reliable and guarantees uninterrupted communication between users and the system. The MySQL database and system backups' mechanism will be considered on implementation.

Performance

The efficiency of the system will be ensured through source code optimization. The aim is to produce a fast as possible system that will

improve enjoyable usability.

Implementation

The system will be hosted on an Apache Server. PHP programming language is used for a client program that initiates transactions from an account that is shared by mobile phone users, on Pay-Pal server.
Java Programming language will be used for a program which will receive and send SMS messages through a mobile-phone device. A mobile-phone device is used to send and receive SMS messages via the mobile network operator to and from the system users.
MySQL database is used for user accounts details processing.

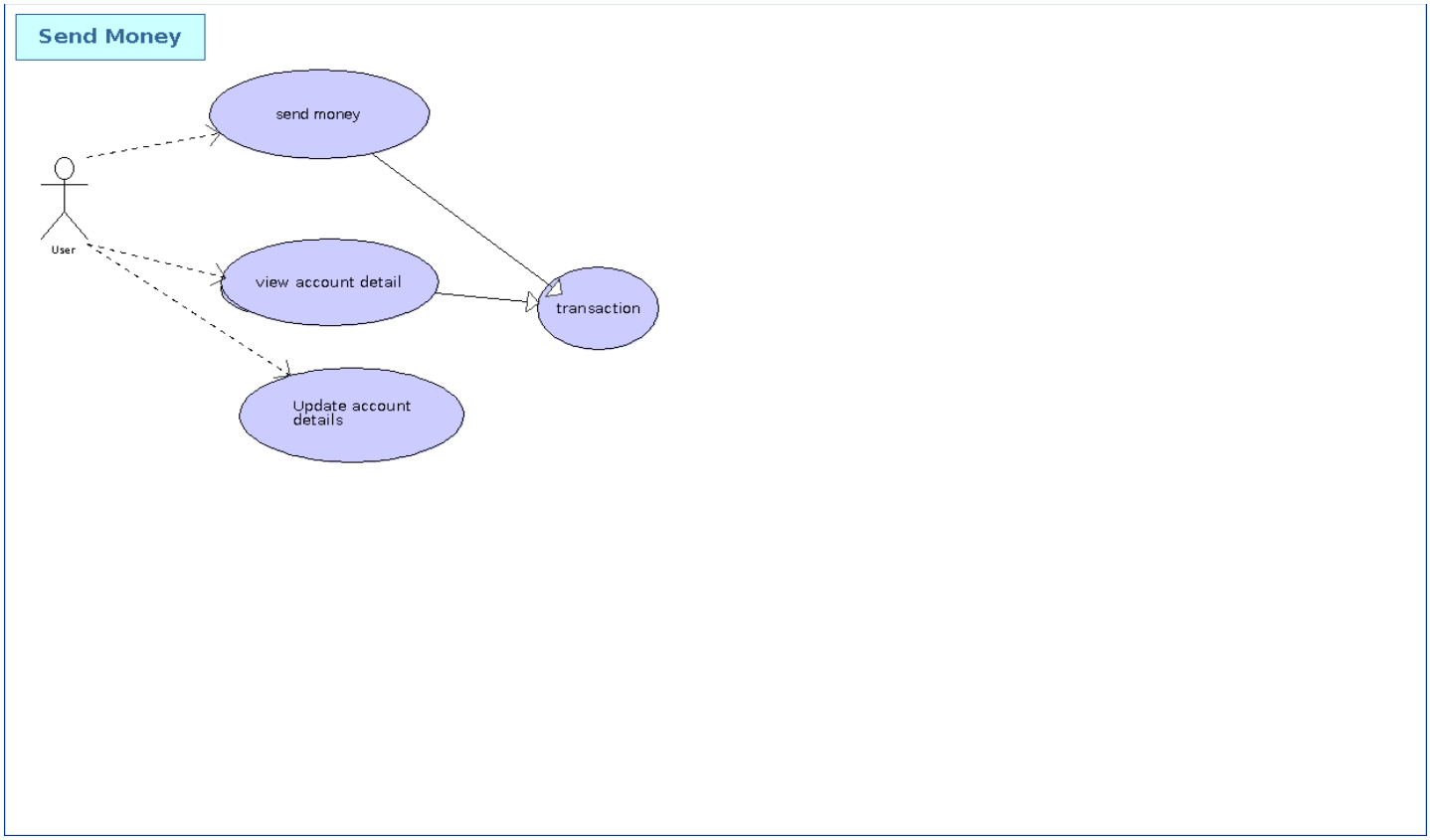
Interface
End users will only communicate to the system by means of SMS, thus, they will see the system outputs in SMS text format based on their specific mobile devices.

System models

A user wants to pay a third party

A girl sends an SMS message to 0791655148 (The system's mobile-phone number), containing this text: “send R300 to 0841234567 from 0791234567”. Then she receives an SMS message with the following text “Please send password for 0791234567”.
After she has sent her password, she will receive a text “Sent R300 to 0841234567, your remaining balance is: R4908”. Also an owner of the mobile number 0841234567 will receive the following text “Received R500 from 0791234567”, as an SMS message from the system mobile-phone number 0791655148.

User case diagram



Conclusion

The system will be beneficial to everyone. The users will open the bank accounts only when they want to not pushed by the need of paying the third party. There will be no need for a user that has received money by means of mobile banking to withdraw cash in order to pay a third party. Moreover, a number of users that has more to deal with will find a fast and reliable way to pay a third party.

CHAPTER 3

USER INTERFACE SPECIFICATION (UIS)

Introduction

The end users access the system by means of SMS. They will have to do every transactions by text messages.

Prototype Screen Shorts

The prototype demonstrates how a user will go about paying the third party by using this system.

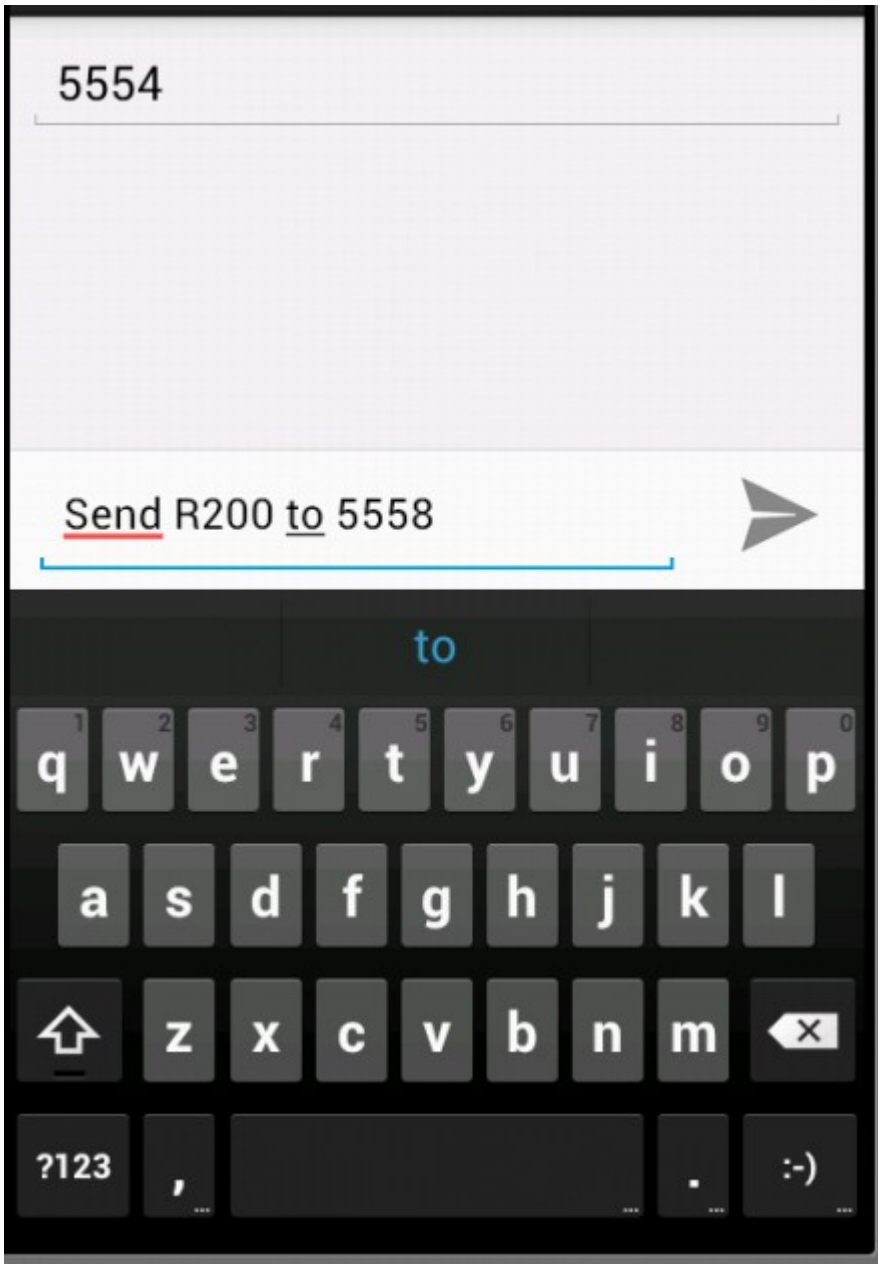


Illustration 1: User send SMS to our system (5554) to pay a third party (5558).

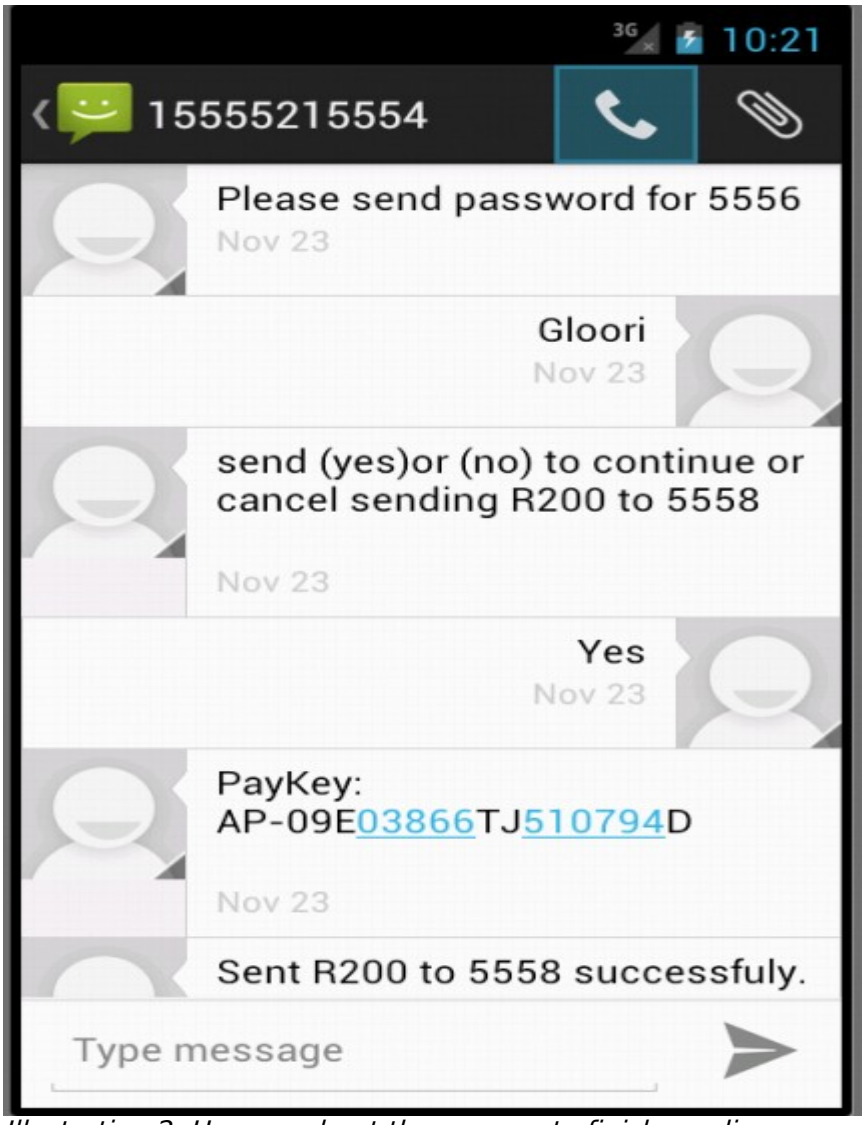


Illustration 2: User go about the process to finish sending money. Then he/she receives pay-Key to that must be given to the third party to approve the payment.

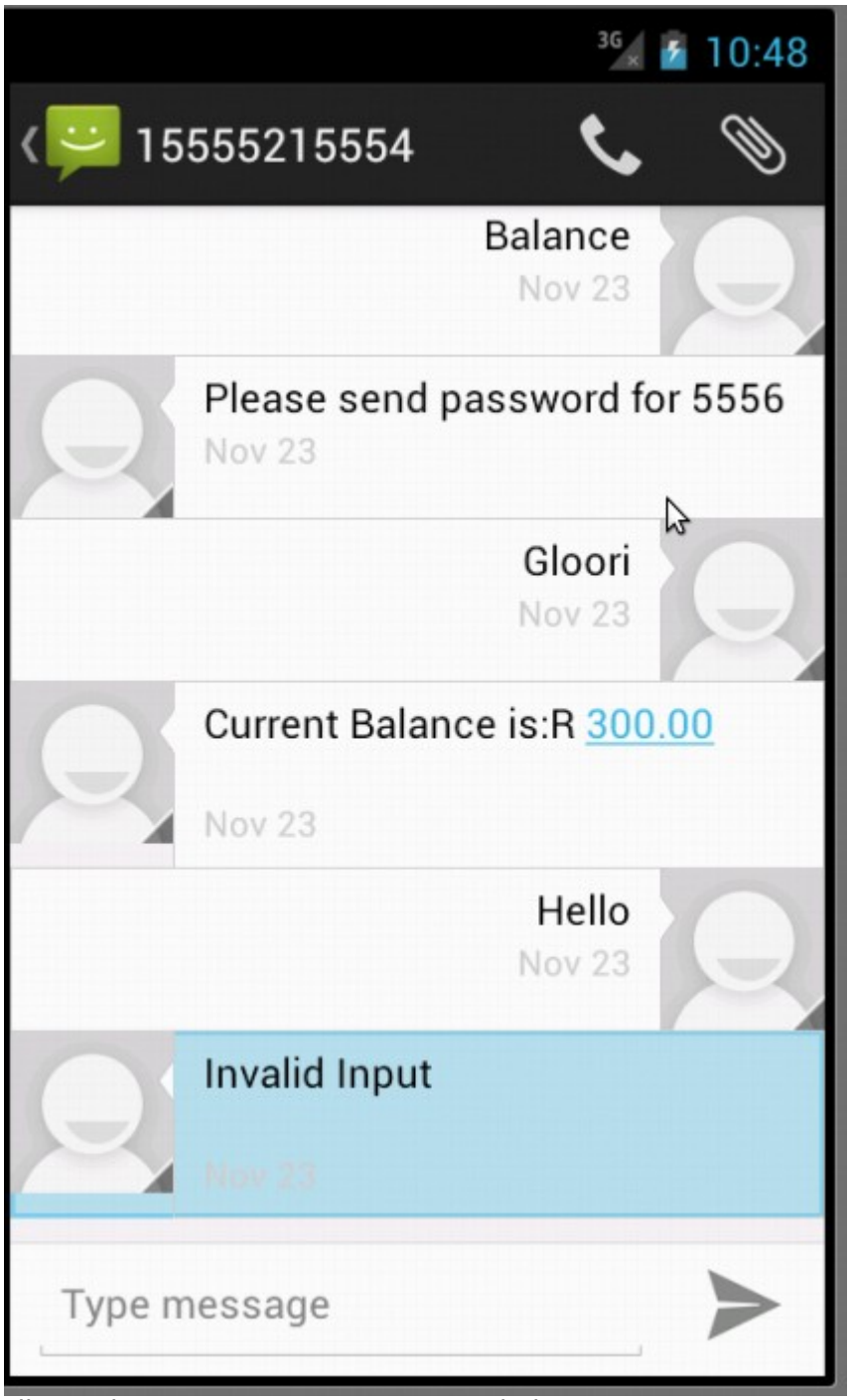


Illustration 3: User requests account balance.

Conclusion

The user interacts with the system by means of SMS. The specific text required from a user to complete a transaction is known, the remaining text received will be considered as the input exception.

CHAPTER 4

OBJECT ORIENTED ANALYSIS (OOA)

Introduction

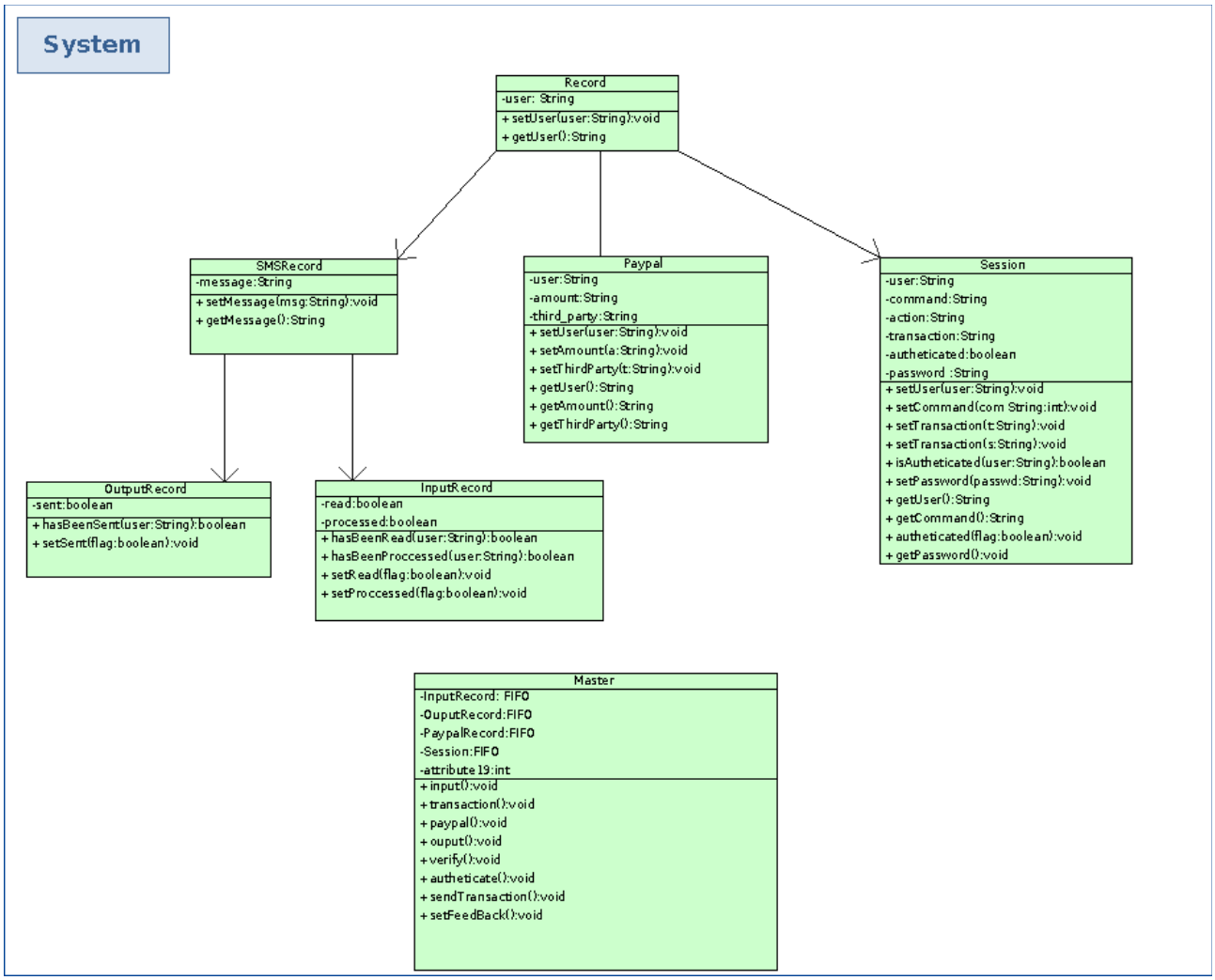
The Object Oriented Analysis (OAA) design stage will be discussed in this chapter. It is a process that precedes the object orientated design stage. And, involves the concepts of object and the activity sequencing.

Business Rules

Business rules defines the ground rules concerning a system and its database. This is to ensure a minimum control on the end users of the system and the organization as a whole. They are usually related to the rules governing the whole company. For this system, Pay-pal business rules are used to define the database blueprint. These rules can be found from the website, named PayPal integration center.

- A CLIENT can have only one ACCOUNT. (1-to-1 relationship)
- An ACCOUNT is defined by a unique mobile-phone number. (Primary key)
- A CLIENT must have a unique identity number. (Primary Key)
- A CLIENT has a unique mobile-phone number. (Foreign Key, links to ACCOUNT table)
- A USER will have a mobile-phone number as the user-name. (Foreign key to ACCOUNT)
- Every TRANSACTION taking place should be stored according to the involved mobile-phone number. (Foreign key, links to ACCOUNT)
- The TRANSACTION is unique determined by the time of occurrence, timestamps(Primary Key)

Class Diagrams



CHAPTER 5

OBJECT ORIENTED DESIGN (OOD)

Introduction

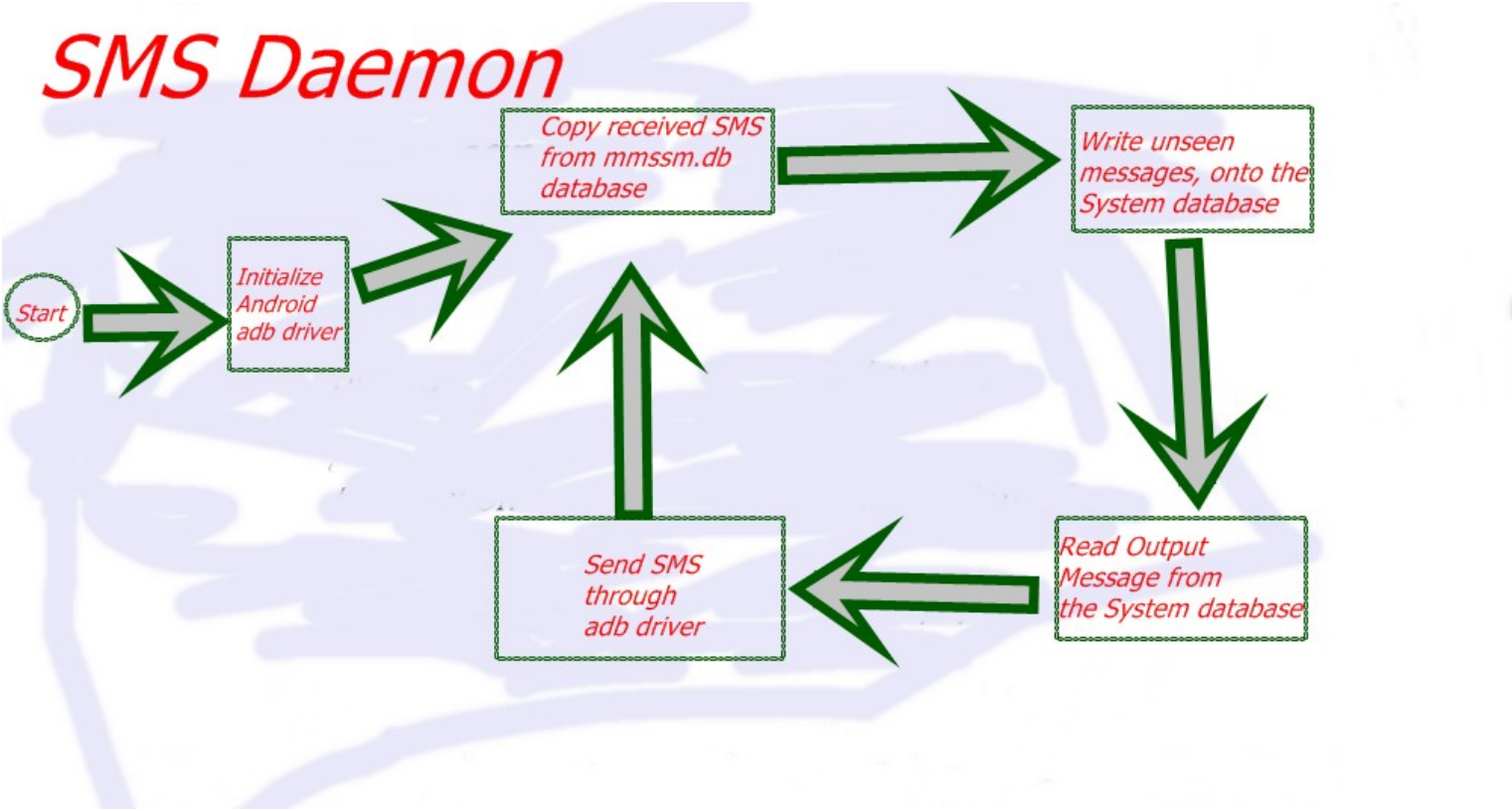
This chapter discuses the system programs design details, where the objects described from the object oriented analysis stage are used.

Development Technologies

- Java, PHP and Perl programming languages
- Linux Ubuntu O/S
- Android phone and SDK
- Net-beans IDE
- Apache and MySQL Database servers

The system program classes are defined clearly on the rest of this chapter.

SMSDaemon



The system will access GSM through a Mobile Station attached to the Server computer by means of USB connection. Android Mobile Phone is a chosen platform for this system.

Algorithm

Initialize the android ADB driver , that enables communication between the mobile phone and computer through the USB connection.

DO:

Send adb command to the mobile phone to copy received messages from the SMS table of the cell phone's messages database.

For each copied message; count how many times does it appear on the on the copied list (phone-count).

Also if such a messages is already on the system database, count how many times does it also appear in it (system-count).

If phone-count > system-count OR the message is not already on the system; write it on the system as MSISDN, BODY, timeStamp, system-count, and SEEN(set to -1).

Send SQL query to the system database to copy a record, where its attribute; DILIVERED is set to -1 (false).

Send the message to the mobile phone by means of ADB command to send SMS to the end user. When message, delivered; update it's

record on the system database as DILIVERED = 0 (true).

REPEAT.

User Input Program

The system will read messages from its databe, those where written by the SMSDaemon program.

Algorithm

Read message record where its SEEN value is set to -1 (false).

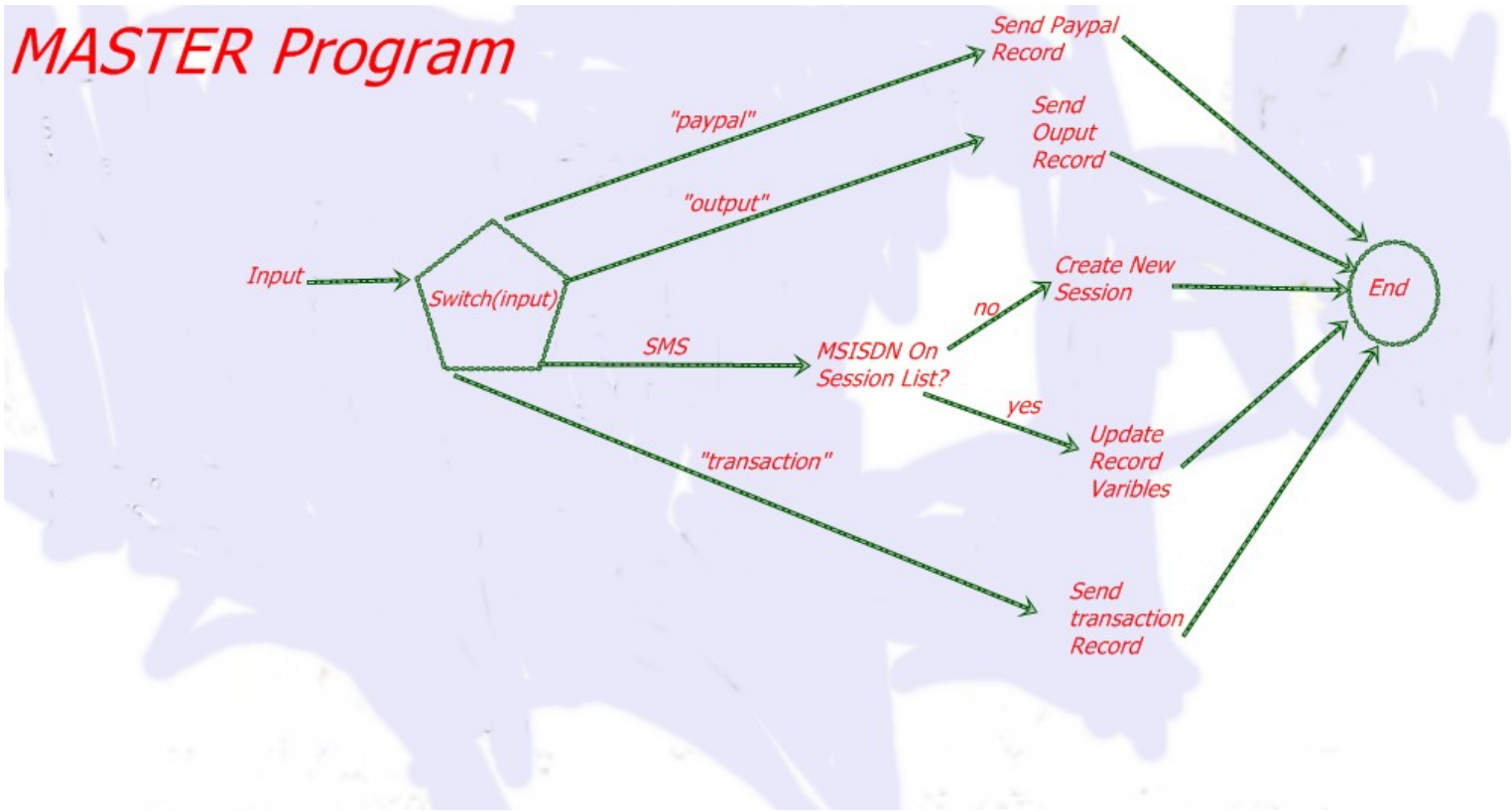
Create a client socket to the Master program (TCP Port 12) .

Froward the message, only MSISDN along with BODY to the server. Read response from the socket.

IF success THEN:

Update the read message record as SEEN = 0 (true).

Server Program



This program is a TCP Server that awaits for clients sockets at port 12. It controls the whole system processing include the SESSION records of Linear lists. It acts upon input from the client programs.

Processing Data

Output Linear List: This stores a record that consists of a mobile number as well as the message that which must be sent to the mobile number.

PayPal Linear List: Stores the records those that consist of a user's mobile phone number, the third party's account number and an amount should be pay the third party.

Session Linear List: Store the record of the users' sessions. Each record consists of a flag, authenticated, that determines whether a user has been authenticated or not.

A string variable, command, tells what the next transaction to take place for this record (e.g.“authenticate” to verify a user's password.).
A string variable, action, determines the type of a session such as money transfer session or update details one, and or etc.

Finally, a string variable named transaction contains what is passed as input to the program that performs the transactions to the SQL server.

Inputs

[message MSISDN along with BODY] , “paypal”, “output”, “transaction” and [“success” | “failure”].

Outputs

[“success” | “failure”] , [message MSISDN along with BODY], [transaction command] and [pay-pal service call details].

Algorithm

Create and bind the server socket to the local-host:12.

DO:

Accept client socket and read input.

Switch (Input);

Case “MSISDN along with BODY”

Search the SESSION list for any record id equal to MSISDN.

IF none found; Create a new SESSION.

Set session id to MSISDN.

Split BODY to array of strings (body[]).

Switch (body[0]);

Case “Send” : Set record's action = “transfer”.

Execute moneyTransfer() function.

Case “Balance”: Set record's action = “balance”.

Execute balanceRequest() function.

Case “Change”: Set record's action = “update”.

Execute updatePassword() function.

Default : DELETE session.

Execute invalidInput().

ELSE (session found);

IF the SESSION record's authenticated attribute is set to false;

Set its password value to the read message BODY.

ELSE;

IF input is “yes”; Set SESSION command to “transaction”.

ELSE IF input is “no”; Terminate SESSION.

Create output object with MSISDN and BODY.

Set BODY = “Good bye!”.

ELSE; Create output object with MSISDN and BODY

Set BODY = “Please send (yes) to continue or (no)”.

Case “paypal”

Send the top record from the PAYPAL list.

Read response from the socket.

Create output object/record with MSISDN and BODY.

IF “failure”; Set BODY = “Transaction failor. Please try again!”.

ELSE;

Set BODY = “PayKey: “+ response.

Set command = “transaction” of SESSION id MSISDN.

ENDIF.

Append created record to OUTPUT records list.

REMOVE Record from PAPAL records list.

Case “output”;

Send the top record's MSISDN along with BODY out the socket.

REMOVE Record from OUTPUT records list.

Case “transaction”;

Access the top record of the SESSION list.

IF its command variable is set to “authenticate”;

Send “authenticate” + session's password value + MSISDN.

Get response from the socket.

IF “failure”;

Create output object/record with MSISDN and BODY.

Set BODY = “Incorrect Password!”.

ELSE;

Set session's authenticated = 0 (true).

Set session's command = “transaction”.

ENDIF.

ELSE IF command == “verify”;

Verify user balance.

Send “verify” + session's amount + MSISDN.

Get response from the socket.

IF “failure”;

Create output object/record with MSISDN and BODY.

Set BODY = “Insufficient funds!”.

ENDIF.

ELSE IF command == “transaction”;

Switch(session's action);

Case “transfer”;

Send “transfer ” + sessions' amount + “ from ”+ session's MSISDN through the socket.

Get response from socket.

IF not “failure”;

Create OUPUT record with session's MSISDN.

Set BODY=”Sent R”+session's amount + “ to ” + third-party + “ successfully.”. response from socket.

ENDIF

Case “balance”;

Send “show balance “ + session's MSISDN.

Read response from the socket.

Create OUPUT record with session's MSISDN.

Set BODY=”balance :R”+ response from socket.

Case “update”;

Send “change password to ” + session's new_password
Read response from socket.

Create OUPUT record with session's MSISDN.

Set BODY=”changed password successfully.”.

User Output Program

The system will write output messages on to the system. This program allow query the output from the master program.

Query : “output”

Response: [MISISDN along with the BODY]

Algorithm

Create client socket to connect to the master. Send a string “output”. Read response from the Master.

If response is not “null” then; write the message on the database as delivered set to -1 (false).

Transaction Class

This program execute SQL commands to support the transactions for the whole system. The program will send a string “transaction” to scan for transactions to perform on the SQL server.

Algorithm

Connect to the master and forward a string “transaction'.

Read the server response, as a line of string: Line .

Strip the line into an array of strings: msg[]

Switch(msg[0])

Case "authenticate" // The input looks as follows:
authenticate +2771234567 'password'

Verify from the SQL server, if the user +2771234567 has a password similar to the one supplied as “password” from the server.

Set $result to “success” if it matches or fail or. otherwise

Case “verify” //The input looks as follows: verify +27712345672000

Simply check if the user has balance above R2000 from the user +2771234567 on the database.

Set $result to “success” where the balance is above R2000 or else fail where the funds are insufficient .

Case “transfer” //The input looks as follows:
transfer 2000 +2771234567 +27838989835.
On the SQL server, decrement by R2000 the balance for the user +2771234567.
If the msg[3] is the mobile number that is registered on the system then increment its balance by R2000 as well.

Finally, set the variable $result to “success”.

Case "change" //Input: change "Table" "Cell" "new information" +27791234567 .

Simply update the specified table name and column where user = +27791234567 with the “new formation”.
Finally, set $result to “success”.

Case “show” //Input: show "Table" "Column" +2789876547 .
Simply select the specified Column from the Table where the mobile number is +2789876547 .
Set $result to the acquired results from the select statement on the database.

Finally, send the $result value out to the server and close the socket.

Outputs

The program outputs the transactions' results back to the server.

CHAPTER 6

Code Documentation

Master Program

```
/*
 * @program Master
 * @author Ndlamlenze NS, MR (2856212)
 *
 * @Description
 * This program is the master of the system. It execute functions on response
 * to slave program's input. The are for slaves, viz Input, Output, PayPal
 * and Transaction.
 *
 * @Inputs:
 *
 * The Input program; enters the end user messages. The remaining
 * three programs enter the string that is its file name.
 *
 * @Outputs:
```

```
 *
 * This program outputs messages to be delivered to the end users, transactions
 * to execute on the database and details for PayPal payment caller service.
 *
*/

package master;

public interface MasterInterface {



/*Global variables */
    private ServerSocket servSock; //server socket
    private Socket clientSock; //A client connected to a serve
    private int port; //Server port
    private DataInputStream fromSock; //socket read
    private PrintStream toSock; //socket writer
    private master.FIFO PayPalRecords; //paypal records linear list
    private master.FIFO OutputRecords; //output records linear list
    private master.FIFO TransactionRecords;
    private master.PayPalRecord pr_pointer; //paypal record pointer
```

```
    private master.OutputRecord or_pointer; //output record pointer
    private master.TransactionRecord tr_pointer; //transaction
record_pointer
    private String data; //input variable
    private String[] msg; //input processing array



/*Send a string 'autheticate' followed by tr_pointer's user and
password values.
    out the client socket.
    Read response from the client socket.
    Switch(Using the response)
    Case 'success'
    //password matches the one kept on the database
    Set the tr_pointer's autheticated value to TRUE
    IF tr_pointer record's action is set to 'transfer' THEN
    Set pointer's command variable to 'verify' to verify the suffient
    funds for a user to transfer to a third party.
    ELSE
    Set the user verification message; run setVerification()
    Default Case
    //password doesnt match the one kept on the database
    Create a new output record to send SMS out to the end user
```

```
    Set the newly created record user to tr_pointer's user
    Set also its message to 'incorrect password, please try again'
    Append the created output record onto the output records list.
     */
    void authenticate();


    /*Create a balance request session and append it on the transaction
list*/
    void balance();


    /*Create a user details updating session or transaction and append
it to the
    transaction records list.
     */
    void changeDetails();


    /*Using the method of regular expressions, we determine if a
message has either send or receive
    and or statement as SMS content to create a new transaction or
session*/
```

```
    /*@return: a function, whether it's transfer, or update, or
account_statement*/
    String getFunction(String data);



    /*Three possible input to initiate a transaction
    A method of regular expression is used tho extrac patterns
    if input has a string 'send'
    run transfer()
    else if it has 'change'
    run changeDetails
    else if 'balance'
    run balance()
    else
    run validateInput()
     */
    void input();



    /*Generates an invalid SMS instance to send it to the end user*/
    void invalidInput();
```

```
    boolean isFound(String T, String p);



    /*Check if a third party is either a paypal account or a mobile
number of a user that
    is registered on this system*/
    boolean isPaypalAccount(String data);

    boolean isStatement(String T);

    boolean isTransfer(String T);

    boolean isUpdate(String T);

    /*Get the top node from the outputrecords linear list and call it;
or_pointer.
    IF the top node or or_pointer is null THEN
    Write a string 'null' out the socket.
    ELSE
    Write both the user and message values of the or_pointer record
out the
    socket.
```

```
    Remove the top output record from the linear list.
     */
    void output();



    /*Searches the linear list; PayPalRecords, for an instance.
    IF no record entity is found on the list; PaypalRecords THEN
    Send a string 'null' out the client socket.
    ELSE
    Get the top node of the paypalrecords linear list, and name  it
pr_record.
    Initialize the a variable; data, with th amount from the record
pr_pointer.
    Append th data with the third party's paypal account from th
pr_pointer record.
    Append data variable with th user's mobile number from th
pr_pointer record.
    Write data variable's value through the client socket.
    Read the PayPal Transactions' result from the client socket
    switch(using the data that was read from socket as results)
    case 'success'
    Set the transaction record pr_pointer's command variable
    to 'transaction'.
```

Remove the paypal record that we have dealt with from the paypal records list.
case 'failure':
Create a new output record; or_pointer, to send SMS out
to the user.
Set the or_pointer's user variable with the tr_pointer's user
value.
Set the or_pointer's message with an unsuccessful message value.
Append the op_pointer record on the output records list
 */
void paypal();

/*A TCP server process is started
loop for ever
accept client and read input from an accepted socket
ELSE a client inputs a string 'paypal' THEN
run paypal()
ELSE IF it's a string 'transaction' THEN
run transaction()
ELSE IF a string is 'output' THEN
run output()

```
    ELSE
    input is an SMS message that has been sent from the end user
    run input()
     */
    void run();



    /* Sends the transaction details found on the transaction record; tr_pointer
    IF a tr_pointer's command variable is set to 'authenticate' THEN
    run authenticate()
    ELSE IF command is 'verify' THEN
    Verify sufficient funds to pay a third party for a tr_pointer's user
    Write a string 'verify' followed by both th tr_pointer's user and amount values
    out the client socket.
    Read the response from the client socket.
    Switch( Using the read response)
    Case 'insufficient funds'
    Create a new output record to send an SMS out to the end user.
    Set the newly created instance's user to the tr_pointer's user value.
    Set also its message variable to 'insufficient funds'.
    Delete the transaction, tr_pointer from the transactions list.
```

```
    Default case
    Set user verification message; run setVerification.
    ELSE
    It is a general transaction, simply write pointer's transaction
variable's value
    out the socket.
    Read response from the client socket.
    IF the response is 'success' THEN
    What ever transactions were successful, generate a user feedback
    message; run setUserFeedBack.
     */
    void sendTransaction();


    /*Generates a user feedback message and also terminate a
transaction*/
    void setFeedBackMessage();


    /*Generates an SMS to get user response about whether he still
wants to continue or cancel the
    transactions*/
    void setVerification();
```

/*Gets the head node from the TransactionRecords list and name it; tr_pointer.

IF the top node or tr_pointer is null THEN

Write a string 'null' out the socket.

ELSE

DECLARE found boolean and initialize it to FALSE

WHILE (NO TRANSACTION HAS FOUND), LOOP through the transactions linear list.

IF a transaction record on the list has command set to 'paypal' THEN

Skip, move to the next node.

ELSE IF a command set to 'sms' THEN

Skip, move to the next node.

ELSE

Set found boolean to TRUE,...has found a transaction.

END WHILE

IF found boolean is still set to FALSE THEN

Write a string 'null' out the client socket.

ELSE

RUN sendTransction()

*/

void transaction();

```
    /*Creates a money transfer session or transaction record and
append
    it to the transaction records linear list
     */
    void transfer();

    /*Ensures that, a user has entered a correct input*/
    void validInput();

    /*If an input SMS has any string that is not meant to
     * create a new transaction
    get a an instance from transaction records list that matches the
user
    if no such user exits on the list
    run invalidInput()
    else
    if the input is the password
    set the instance's passwd = input SMS
    also command = 'verify'
    else a user had to send yes or no, to verify transaction
    if input == 'yes'
    set instances command =  'transaction'
```

```
    else if is 'no'
    remove the instance from transactions list
    else
     *  run invalidInput()
     */
    void validateInput();

}
```

CHAPTER 7

Testing Document

Testing Techniques

The Agile software process was used as therefore, unit testing was applied such that, a lot of changes have taken place. There were three stages of the development framework where the testing processes took place which are; Design, Implementation, and Functional Testing).

Functional Testing

Black Box Testing method was used. The testing case is where the tester does not know anything about the source code or program implementation.
Initially, the requirements were analyzed as follows:

1. The valid inputs were chosen.
2. The, we have tested the invalid inputs.
3. Decided on all possible outputs.
4. Created and executed the test cases with the selected inputs.
5. Debugging and re-testing the fixed defect.

Valid Inputs

Transfer money:
[Send] [amount] [to] [third-party account]

Update password:
[Change password] [new password]

Request balance:
[balance]

User Transaction Verification or Confirmation:
[yes] or [no]

Invalid Inputs

Transfer money :
- [Send] (if a user has misspelled; “send”.)
- Line with less than four strings (a user has forgotten to write; “to”.)
- (forgot to write the amount to be transferred.)
- (forgot to write the third party's account.)

Update password:
- [Update password] (any misspelled word)
- (the user has not included a new password string)

Request balance:
[balance] (misspelling error)

User Transaction Verification or Confirmation:

[yes] or [no] (misspelling error)

Empty string input.

Possible Outputs

User Authentication:

Please send password.

Please send password to change {old password} to {new password}.

The password {supplied password} is incorrect, please send the correct password.

User Transaction Verification or Confirmation:

Please send (yes) or (no) to continue or cancel {specified transaction}.

User feedback:

Sent {amount} to {third-party account}, successfully.

Could not send {amount} to {third-party account}, {reason for transaction failure}.

You have insufficient funds to send {amount} to {third-party account}.

Balance: {amount}.

Invalid input {supplied string}.

Please specify the amount to send to {third party}.

Implementation Testing

White-box-testing technique was used. This test focuses on the internal structures of the application. Code coverage; scripts that make all statements in the program to be executed at least once, control flow testing, data flow testing and decision coverage.

The system consists of the following three units which were was tested on this stage:

1. Master Program – Data Structures and Records implementation functions.
2. PayPal Useragent – Paypal Communication through HTTP with NVP, functionality.
3. User I/O Programs – SMS I/O through a GSM Mobile Station functionality.

Test Tools

The Master Program test is by means of Netbeans developer tool. This involves code Source Code Profilation (Memory and CPU Performance).

Tool	**Function**
Netbeans	CPU Performance

Netbeans	Memory Profilation
AppPerfect	Java Unit Code Performance Testing

Test Scripts

Name	**Average Time (second)**	**Function**
SMSDaemeon-TEST	0.0012	Copy a received message from the Mobile Station to the System database. And also from the System database and send it out the mobile station as SMS.
UserInput-TEST	0.0010	Read unseen message records, from the System database to the Master program.
PayPalPayer-TEST	0.0125	CREATE and EXECUTE remotely, a PayPal Implicit Adaptive Payment.
UserOut-TEST	0.0020	Scan output from the Master program to the System database.

User Testing

Task	No of Users	Average Time\ (minutes)	Variance (seconds)	Level of Difficulty
Add Account	9	1	0.05	Very Easy
Paying a Third Party	9	3	0.04	Average
Change password	10	2	0.05	Easy
Request balance	9	2	0.06	Very Easy

CHAPTER 8

User's Guard

Introduction

The system allows the administrative user as well as end users to manipulate data.

Administrative System

The administrative user adds the users accounts on the system. They user the web based administrative tool.

Request parameters:

Variable	Description
Name	User first names.
Surname	User surname
Identity Number	User identity/passport number
Mobile Number	User Account Registration mobile number.
Initial Balance	Cash handed on by a user when registering account.

Response:

Variable	Description
Username	User's mobile number.
Password	Default user password.

The administrative user can request user details from the system to edit.

Input Parameters:

Variable	Description
Mobile Number	A user's mobile number that his/her details are requested.

Response Parameters:

Variable	Description
Name(s)	User both names and surname.
Mobile-phone number	User account mobile number.
Id Number	User identity number.
Balance	User account balance.
Button	Submit button after editing values.

End User System

The end users get default password that needs to be updated immediately when the user has registered on the system. The End User do all the transactions by means of SMS.

Change password by SMS:

Input   : change password to [new_password].
Output : please send old password.
Input   : [old password].
Output : changed password to [new_password].

Pay a third party by SMS:

Input   : send [amount] to [third party mobile number]
Output : please send password.
Input   : [user_password]
Output : please send (yes) or (no) to continue or cancel [transaction]

EITHER
Input   : Yes
Output : Send [amount] to [third-party mobile number], successfully
OR
Input   : No
Output : Cancel led sending [amount] to [third-party mobile number].

Request balance by SMS:

Input : Balance
Output : please send password.
Input : [your password]
Output : Balance: [amount]

REFERENCES

- Pay-Pal Integration Center URL:https://cms.paypal.com/us/cgi-bin/?cmd=_render-content&content_ID=developer/library_download_sdks
- JblueZ, a Java package which interfaces with the BlueZ Bluetooth protocol stack for Linux. URL: http://jbluez.sourceforge.net/
- BlueZ, an Official Linux and Android Bluetooth protocol stack. URL:http://www.bluez.org/

Android Developer SDK, URL: http://developer.android.com/sdk/index.html

- Android.bluetooth,URL:http://developer.android.com/reference/android/bluetooth/pAckage-summary.html
- How to Handle SMS on Android. Written by: Denys Podymskyy on Friday, 29 April 2011 09:00. URL: http://www.apriorit.com/our-company/dev-blog/227-handle-sms-on-android
- Computer World- How SMS Works, last edited on 2011/11/06 URL:http://computer.howstuffworks.com/e-mail-messaging/sms.html

APPENDICES

Statistical Demography Answers for Open Questions

Why would you need a bank account if there was a system that kept your money at confident , let you pay your bills by sending a couple of SMS and updated you about account statements via the SMS messages.

Respondent
I would need a bank account so that I can have a credit/debit card to pay at shops, garages and restaurants.
To borrow loans, such as study loans. I would not need a bank account, guess.

Respondent
I do not need a bank account.

Respondent
I am a woman of style; prefer not to carry cash around but my ATM card, with me. Paying with your card is up-to-date.

Respondent
To buy air-time from the ATM, use my debit card for payments and a lot, you know.

Respondent
I do not know…
Findings: It seems as people does not use their bank accounts only for banking but for purchases as well. This makes it necessary to

adjust a system so that they can be able to benefit from e-commerce payments systems using their mobile numbers.

What else do you do with your bank account except saving your money, paying the ATM charges and credit/debit card purchasing at shops?

Respondent
Nothing at all

Respondent
I do not have much to do with banking

Respondent
I prefer to buy in cash, do not borrow loans from my bank either.

Respondent
I am paying for my loan I have used to buy my car.

Respondent
Nothing

Respondent
I am not sure

Findings: Some people still depends on their banks for loans; because of the increase in the number of loan providers this should not affect the user's view of this system's quality
Is it too easy to create a bank account such that everyone can open a bank account irrespective of the location they may be situated at presence and their status of life?

Respondent
No, you cannot create a bank account at a different city of you hometown.

Respondent
To create a bank account, you need a valid proof of residence which is hard to find due to the shortage of paper based letters from post-offices.

Respondent
Blacklisted people find it too hard or imposable, to create bank accounts. Besides, they do not allow you to create a business account if your business is still making too less profit.

Respondent
Students cannot create student bank accounts at times where they cannot reach their educational institutions to ask for the proof of registration.

Respondent
Creating a bank account will always be a complex challenge because of the bunch documents needed to create a simple bank account.