ISSN 1512-1232

INSTANT PAYMENT SYSTEM BUSINESS MODEL AND TECHNICAL DESCRIPTION

Sofio Katamadze[1] , Nino Topuria[2]

[1]IS Department, Senior Software Developer, FINCA Bank Georgia
skatamadze88@gmail.com

[2]Associate Professor, Georgian Technical University
nino.topuria@gtu.ge

Abstract
This paper presents how fast transactions are processed by instant payment system. Processing is composed of many components. Each component has failover schema to keep system 24/7 alive. Communication between them is fast and secure. This system is managing its own centralized platform to provide an interface between different channels for payments or services and the corresponding recipients for processing. Instant payment system is a secure solution for processing payments by credit cards, virtual wallet or by cash for many types of providers. To secure the payment, i.e. make sure that the payments requests are really issued by the trusted terminal and are not modified in any unauthorized way, it requires to authenticate requests private and public signed certificates. Beyond the basic payment functionality, current system proposes a set of advanced functionalities providing more flexibility to control its transactions and offering to the end customer interesting value-added services.

Keywords*: Failover schema, Self-service terminal, Instant payment system, POS terminal, .Net program, Rfid reader.*

1. Introduction

The structure of business is not complicated. Company servers are connected:

- To provider's servers on the one hand
- To terminals on the other hand

Client gets to the terminal, makes request of payment, server connection confirms availability of payment, client makes payment and gets confirmation (illustrated in Figure 1).

History of Business Model

The first self-service terminal appeared in Russia in 2004. Soon it became very popular and with the help of flexible scheme it took only decade to become financial giant. The reason of the success was in the correct logic of business, all components – responsible for the success got in the winning position.

Instant payment system is one of the biggest generator of cash, which can get cash out of bank sector and take them back on to the bank accounts, therefore in big countries banks became main stakeholders of business, in some countries they started their own system.

It was not surprise, that new business idea spread into the new countries too. Only Russian giant is operating its' business in 22 countries with different terms - direct management, franchising or licensing rules Almost every retail business sectors were implemented in instant payment systems already in 2010:

ISSN 1512-1232

- Mobile phones operators
- Utility payments
- Internet shops
- Transportation companies
- Lottery tickets
- Automobile payments (parking, fines)
- Bank payments
- Governmental payments

Turkey

Hizli Tahsilat Sistemleri (HTS) is Turkish based company, It was founded in Istanbul, Full functioning started in August 2012. Company provides performance of different payments from self-service kiosks. The use of modern technologies enables system to make any type of payments in live continuous regime (24/7) without any interruption.

Payment chain includes several products:

Self Service Terminal. It works everywhere, only mobile phone service and electricity source is needed. It is equipped with bill and coin acceptors, printer for receipts and with credit card reader in some cases. Touch screen monitor and user-friendly software makes each payment easy.

Figure 2: - Self Service Terminal

It is more flexible and mobile device. It works on batteries. For performance of payments operator is needed. Every service from self-service terminal is installed in POS terminal too.

Figure 3: - POS Terminal

Company's web terminal allows agents and direct customers to make payments from every computer. It has no need of additional devices, only internet connection is enough.

Figure 4: - Web Terminal

Figure 5: - HTS Processing Logic Plan

2. Components

The picture shown above illustrates our company's servers prototype. On each component of the system it is indicated own technology.

The terminal is written in the .NET program. I am making any change or remote update of the pay box.

Now let's discuss the software structure - how the terminal works and what components are required for correctly functioning.

2.1 Terminal Software Components

During Terminal startup, all necessary controllers(jobs) are initiated, such as: configuration, session, server, transaction, etc. Above mentioned methods, enable us to control whole processes in software remotely.

Each terminal has own configuration. Configuration includes all properties required for pay box and providers.

Terminal has very good opportunity for offline providers. According to provider requirements, terminal accepts payments in desired way:

- Online
- Online + Available Offline
- Force Offline

Online is standard transaction.

Online with available offline functionality enables terminal to accept payment in case of provider timeout.

Force offline feature, forces all transactions to fall in offline folder. They are synchronized periodically, in case of answer delivery – file is deleted, otherwise it’s processed again.

The terminal has network and hardware control functions. These functions are constantly running and checking whether the terminal components are enabled. In case of problem, program tries to eliminate defect identified by the error code. This maximizes control mechanism without the intervention of technical personnel.

ISSN 1512-1232

```
        private void InitJobsQueue()
        {
            //Add Functions which should be done when application initialized.
            //If something is not done then it remains in the queue and application will try to do it in offline mode
            mInits.Enqueue(this.InitControllerMngr);
            if (!Program.testFrontEnd)
                mInits.Enqueue(this.InitServerController);
            mInits.Enqueue(this.InitTransController);
            mInits.Enqueue(this.InitConfig);
            mInits.Enqueue(this.InitDenominations);
            mInits.Enqueue(this.InitPlayLists);
            mInits.Enqueue(this.InitRespCodes);
            mInits.Enqueue(this.InitResource);
            mInits.Enqueue(this.InitSession);
#if BV
            mInits.Enqueue(this.InitBillValidator);
            mInits.Enqueue(this.InitCoinValidator);
#endif
            mInits.Enqueue(this.InitOtherControllerS);
            mInits.Enqueue(this.StartVideoRecord);
            mInits.Enqueue(this.StartControllerMngr);
            mInits.Enqueue(this.InitSupervisor);
            mInits.Enqueue(this.SendBV_ID);
            mInits.Enqueue(this.InitBuses);
            mInits.Enqueue(this.InitBusesDirections);
            mInits.Enqueue(this.InitArrivalDirections);
            mInits.Enqueue(this.InitDepartureDirections);
            mInits.Enqueue(this.InitCardDispenser);

            ////////////////////////////////////////////////////////////////////////////////////////////////////////////
        }
```

Figure 6: - Jobs Initialization

2.1.1 Terminal Modes

Terminal has different modes, such as:

- Idle
- Supervisor
- Offline
- Out of Service
- Transaction
- Booting
- Closing
- Addin
- Unknown

Above mentioned modes are displaying terminal status in different cases. Now let's clear out each of them.

When terminal starts up – it is in booting mode. While pay box is trying to reach the processing server - it is in out of service mode. In case terminal couldn't make successful connection, it stays in out of service mode. We should keep in mind, that terminal has all kind of controllers continuously running. They always check if any component is working correctly. So, whenever server responses successful message, terminal mode changes to transaction mode.

Terminal has specified timeout for transaction mode. If there is no interaction with terminal, after timeout it switches to idle mode.

Terminal requires encasement periodically. At that time, technical staff reaches to terminal with special equipments and enters own user name and password. After successful login, system enables to make encasement. Firstly, encasement button is pressed. It sends denominations file to the system (count of coins and cash inserted since last encasement). After successful response, technician can empty coin and bill acceptor cassettes. During encasement process terminal is in supervisor mode.

ISSN 1512-1232

Terminal has supplementary mode called addin mode. We know that even in fully tested systems and programs, there can appear exceptional cases. In live system, remote debugging is not really good tool. Now, best practice is to log locally main methods and events. When problem occurs, or is in suspicious case, we send command from admin panel and receive logs on server. While logs are uploading in the server, terminal is in addin mode.

As we mentioned in previous sectors, terminal can accept offline payments. When terminal has lost local internet connection – it goes in offline mode. For providers which allow offline payments, transactions are still received from customers. After successful internet reconnection, all transactions are processed.

Terminal can have exceptional case when none of the modes are assigned. In that case terminal is in unknown mode.

```
int iRandomID = 0;
//1 - check up Master Server
//Create Responce Object
CServerControlRequest oServerControlRequest = new CServerControlRequest(base.mApplication.TerminalID, Program.myMasterServerURL);
CResult<CCheckServer> oServerControl = new CResult<CCheckServer>();
oServerControl.result = new CCheckServer();
//Get Random id
iRandomID = oServerControl.id;
try
{
    mApplication.JsonConnection.Request<CCheckServer>(oServerControlRequest, ref oServerControl);
}
catch (Exception ex) { CUtils.ErrorHandler(null, ex); }
String myServerStatus = "";
if (iRandomID == oServerControl.id)
    myServerStatus = oServerControl.result.serverStatus;
//if master doesn't work check slave server
if (myServerStatus != "00")
{
    string tempSlaveServerURL = "";
    if (Program.myMasterServerURL == masterServerURL)
        tempSlaveServerURL = slaveServerURL;
    else if (Program.myMasterServerURL == slaveServerURL)
        tempSlaveServerURL = masterServerURL;
    //1 - check up Master Server
    //Create Responce Object
    oServerControlRequest = new CServerControlRequest(base.mApplication.TerminalID, tempSlaveServerURL);
    //\\
    //
    oServerControl = new CResult<CCheckServer>();
    oServerControl.result = new CCheckServer();
    //Get Random id
    iRandomID = oServerControl.id;
```

Figure 7: - Server Control

2.1.2. Server Control

Above figure demonstrates small part of the server controller code. From experiences, most of big organizations have backup systems of each component of the processing. Especially in live systems, where payments are processed 24/7, each minute is very important. It's like a live organism, it feels pain when any part of it is damaged and needs urgent recovery. To keep all data and system always protected and healthy, we should always recheck it continuously. For frontend server, we have master slave principal. In predefined timeout, server controller thread sends special request to the system. Depending on the response, controller leaves terminal in master server or switches it to the slave. Even terminal is switched to slave server, controller still sends requests to the system. Whenever master server goes alive – terminal is back to master again.

As we observe on code snippet, some technologies used for connection gets clear. JSON requests are processed by REST service.

ISSN 1512-1232

2.2 Hardware Components

The terminal consists of a lot of devices. I have programmed them and integrated with the existing projects. Below is a list of pay box devices:

- Bill Acceptors (ICT P77, ICT L77, ICT P85, Cashcode MVU, Mei)
- Coin Acceptors (ICT SCA1, WHB EMP890)
- Bill / Coin Dispenser +Acceptor (MEI BNR4-21, MEI CF7900)
- Pin Pads
- Barcode Scanner (LV1000R)
- Printers (Hwasung HMK-054, SANEI)
- Modems
- Card Dipsenser (CRT-541, TTCE-D2000)
- Rfid Reader (CN613-U)

Well it might look a bit suspicious that one person can handle them all together. We can clear out some few steps to start hardware programming. All we need is good hardware documentation, SDK of the device, and good programming skills.

```
/// <param name="iComPort">Port number</param>
/// <param name="bitRate">Bitrate</param>
/// <param name="parity">Parity</param>
/// <param name="dataBits">Databits</param>
/// <param name="stopBits">Stopbits</param>
/// <returns>True if succeeded otherwise false</returns>
bool BVConnect(int iComPort, int bitRate = 9600, Parity parity = Parity.Even, int dataBits = 8, StopBits stopBits = StopBits.One
/// <summary>
/// Disconnect from device
/// </summary>
void BVDisconnect();
/// <summary>
/// Resets device
/// </summary>
/// <returns>True if succeeded otherwise false</returns>
bool Reset();
/// <summary>
/// Enables device for bill acceptance
/// </summary>
/// <returns>True if succeeded otherwise false</returns>
bool Enable();
/// <summary>
/// Disables device for bill acceptance
/// </summary>
/// <returns>True if succeeded otherwise false</returns>
bool Disable();
/// <summary>
/// Informs device to hold bill for further instructions
/// </summary>
/// <returns>True if succeeded otherwise false</returns>
bool Hold();
/// <summary>
/// Polls status of device
/// </summary>
/// <returns>True if succeeded otherwise false</returns>
ID003Command Poll();
```

Figure 8: - Cash Acceptor Main Functions

2.2.1 **Bill Acceptor**

The cash acceptor machine has a lot of functions such as: switching on, turning off, switching to listening mode – when machine is ready to accept cash; resetting, etc.

ISSN 1512-1232

To receive money, the device must pass several stages. The poll function in the terminal is constantly receiving various orders from the machine. Terminal accepts already identified codes from the device and acts correspondingly.

Now let's make it clear how it works? Well, each device has its brand name and each brand has its working protocol. We have two kinds of devices – ICT and CASHCODE. ICT works on ID003 protocol and CASHCODE works on CCNET protocol.

ID-003 interface is a bi-directional serial interface, which enables CONTROLLER to control the status and action of ACCEPTOR and confirm the function settings by sending the polling ([STATUS REQUEST]) and the commands ([OPERATION COMMAND] and [SETTING COMMAND]).

Transmission Specification [1]

- Transmission method - Full-duplex transmission
- Transmission speed - 9600 bps/19200 bps
  (Selectable with dipswitch, depending on the machine model)
- Synchronous system - Asynchronous method
- Connection control method - Polling method
- Data format - Start bit 1
  Data bit 8
  Parity bit EVEN
  Stop bit 1
  X parameter Not used

Message format

SYNC 1 byte: Start code of sending message [FCH] fixed
LNG 1 byte: Data length (total number of bytes from SYNC through CRC)
CMD 1 byte: Command, status
DATA 0-250 byte: Data required for a command (may be omitted, depending on the CMD)
CRC 2 byte: Check code of CRC method
The object is the interval from SYNC through the end of DATA.
CRC *(L)* CRC *(H)* (Default value = 0)

Error control, system Error detection, CRC method
CRC-CCITT
P (X) = X16+X12+X5+1

Sending/Receiving Message Format

Formats of sending/receiving messages are classified into five types as shown below.

(1) Polling format (CONTROLLER □ ACCEPTOR)
SYNC: [FCH]
LNG: [05H]
CMD: [11H] (STATUS REQUEST)
CRC *(L):* [27H]
CRC *(H):* [56H]

(2) ACK format (CONTROLLER □ ACCEPTOR / ACCEPTOR □ CONTROLLER)
SYNC: [FCH]
LNG: [05H]
ACK: [50H]
CRC *(L)*: [AAH]
CRC *(H)*: [05H]

ISSN 1512-1232

Command format (CONTROLLER □ ACCEPTOR)

SYNC: [FCH]
LNG: Data length
CMD: Command
DATA: Data required for a command (may be omitted, depending on the CMD)
CRC: Check code of CRC method (2 byte)

(4) Response format I (ACCEPTOR □ CONTROLLER)

SYNC: [FCH]
LNG: Data length
SST: Status code
DATA: Data required for a status (may be omitted, depending on the status)
CRC: Check code of CRC method (2 byte)

(5) Response format □ (ACCEPTOR □ CONTROLLER)

SYNC: [FCH]
LNG: Data length
CMD: Response
DATA: Data required for a command (may be omitted, depending on the CMD)
CRC: Check code of CRC method (2 byte)

Communication Flow

(1) Sending STATUS REQUEST

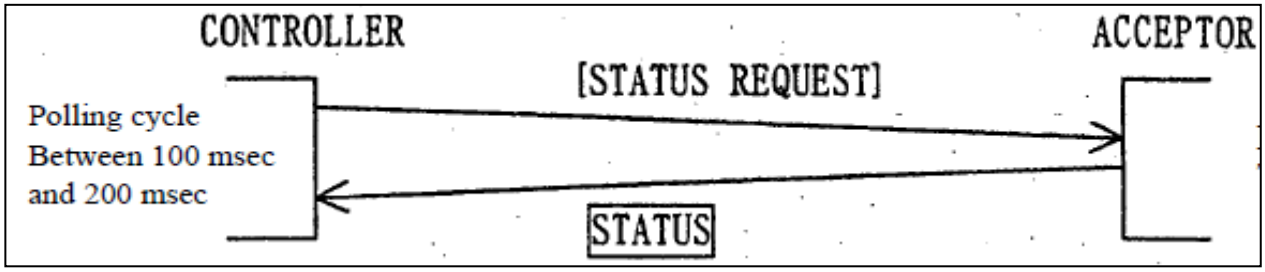

Figure 9: - Status Request Flow

When sending STATUS REQUEST after sending command to ACCEPTOR, transmission interval should be left for polling cycle interval.

(2) Sending a command to ACCEPTOR

Command transmission must not overlap with a response to polling.

(3) Communication error □

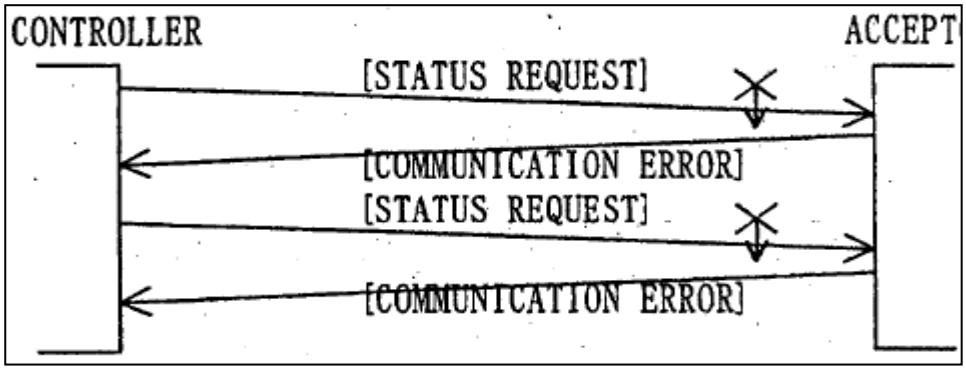

Figure 10: - Communication Error

If RESET occurs in ACCEPTOR, recovery of communication may take a few seconds. Therefore, STATUS REQUEST must be sent continuously, and status of ACCEPTOR must be monitored even if a communication error is detected.

ISSN 1512-1232

ACCEPTOR comes into [STACKING] status upon sending [ACK] response.

A response of [INVALID COMMAND] status is sent back when receiving a [STACK-1] command (OPERATION COMMAND) resent from CONTROLLER. The response of [INVALID COMMAND] status against [STACK-1] command shows that ACCEPTOR has normally received the [STACK-1] command and also been in a status other than [ESCROW] status.

Therefore, in this case, the status of ACCEPTOR is to be verified by sending [STATUS REQUEST] to ACCEPTOR from CONTROLLER.

A request from CONTROLLER for a response on the status of ACCEPTOR.

CONTROLLER monitors the action status, return from the error status, etc. of ACCEPTOR by using [STATUS REQUEST].

Response: Status answer

- Polling cycle is to be between 100 msec and 200 msec.
- A response from ACCEPTOR is to be made within 50 msec.
- CONTROLLER is to resend a message when receiving a response of communication error
- and/or receiving no response within 200 msec.

```
case ccnPollResults.ST_REJECTING:
    CUtils.AdvancedLogger(CCNBV.PrintPoolResult());
    Program.BillValidatorReady = false;
    break;
case ccnPollResults.ST_ST_FULL:
    if (mbCasseteFull != true)
    {
        mbCasseteFull = true;
        if (OnCasseteFull != null)
            OnCasseteFull(CCNBV, new EventArgs());
    }
    Program.BillValidatorReady = false;
    break;
case ccnPollResults.ST_BOX:
    if (mbCasseteRemoved != true)
    {
        mbCasseteRemoved = true;
        if (OnCasseteRemoved != null)
            OnCasseteRemoved(CCNBV, new EventArgs());
    }
    Program.BillValidatorReady = false;
    break;
case ccnPollResults.ST_BV_JAMMED:
case ccnPollResults.ST_ST_JAMMED:
case ccnPollResults.ST_FAILURE://sofi edit
    mbCasseteJammed = true;

    var eFailure = new CCNETBillValidatorEventArgs();
    eFailure.CMD = pollStatus;
    eFailure.DATA = pollData;
    if (this.ControllerState == eControllerState.eConnected)
    {
        this.ControllerState = eControllerState.eDisconnected;
        this.RaizeOnFailure(CCNBV, eFailure);
    }
```

```
case ccnPollResults.ESCROW:
    var eEscrow = new CCNETBillValidatorEventArgs();
    eEscrow.CMD = pollStatus;
    eEscrow.DATA = pollData;
    eEscrow.BillType = this.GetBillIndex((int)(pollData) + _denominationShift
    eEscrow.Bill = this.GetBill((int)(pollData) + _denominationShift, eEscrow
    eEscrow.BillName = this.PrintMoney((int)(pollData) + _denominationShift,
    if (this.RaizeOnAccepting(CCNBV, eEscrow))
        if (CCNBV.Pack())
        {
            escrowBillType = eEscrow.BillType;
            escrowBill = eEscrow.Bill;
            escrowBillName = eEscrow.BillName;
            vendValid = true;
        }
        else
        {
            CCNBV.Return();
        }
    break;

case ccnPollResults.ST_RETURNING:
    CUtils.AdvancedLogger(CCNBV.PrintPoolResult());
    Program.BillValidatorReady = false;
    break;
case ccnPollResults.ST_CHEATED:
// case ccnPollResults.COMMUNICATION_ERROR:
case ccnPollResults.ST_PAUSED:
    this.ControllerState = eControllerState.eDisconnected;
    Program.ba_status = 6;
    Program.BillValidatorReady = false;
    break;
default:
    Program.BillValidatorReady = false;
    break;
```

Figure 11: - ID003 Protocol C# Code Snippet

3. Proxy Server

Proxy Server is a transmitter of payments to providers.

The payment proxy server is written in Java programming language. Any new provider is involved with the respective protocol. We have lot of providers working in the system.

Proxy server is java applet. It is always in listening mode. It accepts special commands such as balance, check and pay.

Actually, processing transactions consist of three parts:

- Specification of a specific abortion provider (Balance Call)
- Check

ISSN 1512-1232

- Pay

Each provider service is mapped to our protocol. All providers in our proxy server work in the same way after dual integration. Figure 11 below shows code snippet of one sample provider:

```
if (command.equals("info")) {
    date = new Date();
    GetProvisionRequest prReq = new GetProvisionRequest();
    prReq.setPassword(password);
    prReq.setTransactionDateTime(asXMLGregorianCalendar(date));
    prReq.setUsername(username);
    prReq.setUtilityCompanyCode(555);
    prRes = getProvision(prReq);
    if (!prRes.getResponseCode().equals("0000")) {
        ProviderLogger.log(prvId, "RESPONSE(" + txnId + "," + command + ")\t" + prRes.getResponseCode());
        return prRes.getResponseCode();
    }
    PackageInquiryRequest piReq = new PackageInquiryRequest();
    piReq.setCustomerAccessType(1);
    piReq.setProvisionId(prRes.getProvisionId());
    piReq.setCustomerCode(account);
    piReq.setLanguageCode("tr-TR");
    piReq.setPassword(password);
    piReq.setTransactionDateTime(asXMLGregorianCalendar(date));
    piReq.setUsername(username);
    piRes = packageInquiry(piReq);
    result = piRes.getResponseCode();
}

if (command.equals("check")) {
    date = format.parse(txnDate);

    GetProvisionRequest prReq = new GetProvisionRequest();
```

Figure 12: - Proxy Server Code Sample

4. Conclusion

This was a brief overview of the HTS system, where I have made a great contribution to software development. Our entire team will raise payments in this direction and continue to develop the company with all the forces.

References

[1] ID-003 Communication Specification, The 5th edition, JAPAN CASHMACHINE CO., LTD.
[2] MEI CF7000 USB .NET API Programmers Manual, July 15, 2011, Bryan Fishell
[3] CF7000 USB Harness Users Manual, May 1, 2008, WCR

Article received: 2017-12-25