CORE
Metadata, citation and similar papers at core.ac.uk
Provided by ePrints.FRI

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Šeme

Nakup z magnetnim zapisom z MasterCard obročno shemo na Verifone Vx520 POS terminalu

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Luka Šajn

Ljubljana, 2019

COPYRIGHT. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi preučite programske vmesnike za delovanje POS terminalov za plačevanje s plačilnimi karticami. Osredotočite se na vrsto kartic MasterCard in možnost odloženega plačevanja preko obrokov. Razvijte in validirajte potrebno programsko kodo. Ob implementaciji, naj bo nova funkcionalnost delujoča v takšni meri, da sovpada s trenutno aplikacijo in ne ogrozi katerekoli že delujoče funkcionalnost. Najprej je potrebno razumevanje obstoječega aplikacijskega ogrodja ter njegovega delovanja, nato sledi sama implementacija MasterCard obročne sheme. Implementacija naj bo testirana do take mere, da uspešno opravi Bankart-ov integralni test.

Za izdelavo diplomske naloge, bi se najprej rad zahvalil staršem, ki so mi omogočili študij, ter družini za podporo. Prav tako bi se rad zahvalil za pomoč sodelavcev, ki so mi temo omogočili, kot tudi Bankartu, ki je bil pripravljen pomagati. Seveda je potrebna zahvala tudi mentorju, ki mi je ob izdelavi diplomske naloge pomagal, da sem prišel do samega cilja.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
POS	Point-of Sale	Plačilni oz. transakcijski terminal
CT	Contact transaction	Kontaktna transakcija
CTLS	Contactless transaction	Brezstična transakcija
Mag	Magstripe transaction	Transakcija z magnetnim zapisom
Man	Manual transaction	Ročna transakcija
ECR	Electronic cash register	Blagajna
MC	MasterCard	MasterCard
flowchart	Flowchart.xml	Flowchart.xml
SPDH	Standard POS device handler	Standardni upravljalnik POS naprav
TMS	Terminal management system	Sistem za upravljanje POS terminalov
Online	Online transaction	Transakcija s povezavo
Offline	Offline transaction	Transakcija brez povezave
BIN	Bank identification number	Identifikacijska številka banke
PAN	Permanent account number	Stalna številka računa
NLB	New Ljubljana bank	Nova Ljubljanska banka
Token	Token	Žeton
FRMT-CDE	Format code	Vrsta kode
INSTL-PLAN-TYP	Instalment plan type	Način plačila obrokov
PMNT-OPT	Payment option	Možnost plačila MC obrokov
INSTL-OPT	Instalment plan option	Vrsta plačila MC obrokov
OPTION	Option	Številka sheme

MIN-NUM-INSTL	Minimum number of instalments	Najmanjše število obrokov
MAX-NUM-INSTL	Maximum number of instalments	Največje število obrokov
INRTRST-RATE	Interest rate	Obrestna mera
INSTL-FEE	Instalment fee	Provizija obrokov
ANNUAL-PCTG-RATE	Annual percentage rate	Letna obrestna mera
TTL-AMT-DUE	Total amount due	Skupni dolg zneska
FIRST-INSTL-AMT	First instalment amount	Prvi obrok
INSTL-AMT	Instalment amount	Znesek obroka
NUM-INS	Number of instalments	Število obrokov
ICC	Issuer Country Code	Oznaka države izdajatelja
INTRST-RAT-PRD	Time period over which the interest rate is calculated	Časovno obdobje preko katerega je izračunana obrestna mera
ISS-CRNCY-CDE	Numeric currency code	Številka kode valute
FIRST-PMNT-DAT	Due date of first instalment	Rok plačila prvega obroka
INSTL-GRATUITY-PRD	Grace period before the first instalment payment	Obdobje pred plačilom prvega obroka
XML	Extensible markup language	Razširljiv označevalni jezik

Povzetek

Naslov: nakup z magnetnim zapisom z MC obročno shemo na Verifone Vx520 POS terminalu.

Idejo diplomske naloge sem dobil v službi na podjetju Printec S.I. d. o. o. Tam sem imel tudi obvezno prakso v tretjem letniku. Ker so me kasneje zaposlili, saj so potrebovali dodatno pomoč pri POS plačilnih terminalih za produkcijo v letu 2019, sem se za nakup z MC obročno shemo odločil zato, ker je bila naloga bolj obsežna. Najprej sem moral spoznati strukturo ter arhitekturo obstoječe POS aplikacije, ki so jo izdelali v Printec Romuniji, nato razumeti vse zahteve MC za obročno shemo ter pravilno obnašanje terminala. Sledila sta programiranje oz. izdelava funkcionalnosti. Po združitvi izvorne kode je bilo potrebno obsežno testiranje, da nikakršna sprememba ni vplivala na že obstoječe delujoče funkcionalnosti. Ko je bilo vse pravilno implementirano, je sledilo ponovno testiranje zaposlenih na Bankartu, da so preverili pravilnost delovanja. Potek vsakega koraka je podrobneje zapisan v diplomskem delu, ki se najprej začne z razumevanjem arhitekture POS aplikacije ter njenega delovanja. Nato sledi implementacija plačila z MasterCard obročno shemo, če je na voljo.

Ključne besede: MasterCard obroki, POS terminal.

Abstract

Title: Magstripe Sale with MasterCard installments on Verifone Vx520 POS terminal.

I got the idea for the thesis at work in the company Printec S.I. d.o.o. I also had mandatory practice in the 3rd, the final year. Later they employed me because they needed extra help with POS terminals which should be ready for the production in 2019. Since this task was more extensive, I decided I will take it for my thesis. Firstly, I had to learn and understand how existing application works since the framework was done in Printec Romania. Then, I had to read and understand all the MasterCard requirements for installments and how should POS terminal behave during the transaction. After that, it was a programming described functionality. When the code was done, it was merged into the existing solution and thoroughly tested and fixed where needed since the added feature must not break something which has already worked. Later, it was tested again by Bankart, i.e. ATM and Card processing Center in Slovenia. How each step was done, it is explained in detail later in the thesis, firstly, with an explanation of the framework used for programming and after that with implementation for MasterCard installments payment option, if there is such an option.

Keywords: MasterCard installments, POS terminal.

Poglavje 1

Uvod

Da lahko bralec lažje razume celotno diplomsko delo, je potrebno začeti s tem, kako je prišlo do ideje diplomske naloge. V tretjem letniku študija mora vsak študent opraviti tudi obvezno prakso, ki traja neprekinjeno devet tednov. Za svojo prakso sem izbral podjetje Printec S.I. d.o.o., ki se primarno ukvarja z vzdrževanjem ter programiranjem bankomatov ter POS terminalov. V času moje prakse so me potrebovali za izdelavo ECR-POS simulatorja, ki deluje kot katerakoli blagajna, ki omogoča serijsko komunikacijo med obema napravama ter finančne transakcije na POS terminalu. Sama inicializacija transakcije se začne ter konča na ECR-u oziroma blagajni, POS pa glede na zahtevke pravilno odgovarja nazaj ECR blagajni. Ker je bil simulator dokončan ter testiran pravočasno in so bili z njim zadovoljni, so me kasneje tudi zaposlili, da bi pomagal pri zaključku POS aplikacije za Verifone Vx520 terminale, ki naj bodo pripravljeni za produkcijo v letu 2019.

Ker je bila aplikacija delno izdelana, je bilo potrebno dodati še veliko dodatnih funkcionalnosti, da ustreza vsem NLB ter posledično MC ter Visa standardom. Lokalne spremembe, ki so različne pri vsaki banke preverja Bankart, ki je transakcijski procesni center. Bankart opravi tako imenovani integralni test, ki pove, ali so vse zahteve pokrite, ter se POS terminal obnaša tako, kot je pričakovano. V nasprotnem primeru je potrebno POS aplikacijo spremeniti oz. dodelati, dokler integralni test ni uspešno opravljen. Pomembno je poudariti, da mora biti POS terminal predhodno certificiran po vseh MC ter Visa standardih oz. mandatih.

POS aplikacija je zgrajena po specifični arhitekturi, ki so jo izdelali v Printec Romuniji. Programer, ki je zasnoval arhitekturo, je leto razdelil na tri ločene nivoje. Nivo ena ter dva sta namenjena branju plačilnih kartic, procesiranju EMV polj, preverjanju PIN-ov, varni zaščiti PIN ključev, skrbi za mrežno komunikacijo ter veliko ostalim stvarem. Najvišji, tretji nivo je namenjen samemu poteku transakcije oz. kaj se bo zgodilo na vsakem koraku transakcije oziroma ob pritisku gumbov. S tretjim nivojem uredimo, kako se POS terminal obnaša, kaj bo prikazal na zaslonu; vsebuje vse klice funkcij, ki se zgodijo v aplikaciji, oziroma kliče

funkcije, ki so zapisane na tretjem ter drugem nivoju. Za tako imenovan potek vsake funkcionalnosti skrbi le ena datoteka z imenom »flowchart.xml«, ki je tudi bolj podrobno opisana v poglavju Razlaga Flowchart., saj je ključna pri razumevanju strukture, v kateri je zasnovana aplikacija.

Ker je programer iz Romunije, ki je začrtal arhitekturo, odšel iz Printec Group Romunija S.R.L., preden je zapisal dokumentacijo, sta pri razumevanju pomagala dva programerja iz Romunije, ki sta že razumela, kako arhitektura deluje. Ker se zahteve med vsako državo razlikujejo, je bilo potrebno dodati veliko manjših funkcionalnosti ter popravkov, ki so ustrezali slovenskim razmeram. MC obročni nakup je bil obsežnejši za implementacijo, zato sem se zanj tudi odločil za temo diplomske naloge. Ker je bilo potrebno naprej spoznati ter razumeti, kako aplikacija deluje ter kako se dodaja novo funkcionalnost, je bil izziv večji in posledično tudi zanimiv. Ker se MC obročna shema, ki se večinoma uporablja za tuje kartice, razume kot ločena transakcija, je potrebno paziti, da se vsak korak zgodi takrat, ko je glede na zahteve MC-a potrebno [3, 4, 5]. Poseg v aplikacijo je bil posledično velik. Veliko stvari gre lahko narobe. In to mi je dalo motivacijo, da uspešno razvijem opisano funkcionalnost. Kako mi je to uspelo ter s kakšnimi izzivi sem se srečal, je podrobneje opisano v diplomskem delu.

Poglavje 2

Priprava za izdelavo diplomske naloge

Na začetku izdelave diplomske naloge sem prejel prenosni računalnik, na katerem je že bila naložena vsa programska oprema, ki sem jo potreboval. Potreboval sem Visual Studio 2015 za programiranje, Notepad ++ za pregled logov POS terminala, ki jih je zajel Putty, podpisno kartico, s katero se nove spremembe kode podpiše in jo le tako POS prejme kot veljavno. Podpisna kartica je le eden od varnostnih elementov, ki preprečujejo zlorabe POS terminala oziroma strank, ki ga posledično uporabljajo. Poleg programov sem potreboval poznavanje obstoječe strukture ter dokumentacijo MC obročne sheme [3] s primerom transakcije, ki je bila podana s strani Bankarta. Seveda je bil potreben tudi delujoč POS terminal Verifone Vx520. Pred začetkom razvoja nove funkcionalnosti je bilo naprej potrebno razumeti delovanje arhitekture, da dodana funkcionalnost ne bo uničila oz. onemogočila že obstoječih funkcionalnosti.

2.1 EVO Arhitektura

Arhitekturo, ki je uporabljena za izdelavo POS aplikacije, je načrtal Romun v Printec Romuniji, ki je kasneje zapustil podjetje. Znanje o delovanju so imeli tudi ostali zaposleni, a dokumentacija ni bila zapisana. Arhitektura je sestavljena iz treh nivojev. Najnižja sta prvi ter drugi nivo, ki skupaj skrbita za procesiranje kartic, branje kartic, procesiranje EMV polj oz. tako imenovano polje 55. Tu so vsi podatki kartice, kot je na primer PAN, CVC, veljavnost kartice, IFR oznaka itd. Do vpogleda v omenjena nivoja nismo imeli dostopa, niti nismo imeli vpogleda v izvorno kodo. Zanašati se je bilo potrebno na vrnjene vrednosti ter se na njih pravilno odzvati na nivoju tri. Nivo tri je večinoma zapisan z xml-i [6]. Celoten potek vsake funkcionalnosti ter vsakega dogodka, ki se zgodi na POS terminalu, je zapisan ter primerno voden v eni xml datoteki z imenom flowchart.xml. Slednja kliče ustrezno funkcionalnost glede na dogodek, kot je na primer pritisk tipke in nato ustrezno kliče naslednji ukaz ali funkcijo glede na vrnjeno vrednost funkcije. Vse metode, ki skrbijo za pravilni potek POS-a, so zapisane v omenjeni xml datoteki. Specifični ukaz lahko predstavlja nastavitev vrednosti

spremenljivke, primerjavo vrednosti, prikaz na zaslonu ali podrobneje zapisano funkcionalnost v podrejenih razredih. Vsak ukaz, ki je klican v flowchart-u, je definiran z zaporedjem številk. Slednje pa so vezane na metodo. Ta na koncu vrne rezultat, ki je število. Nivo 1 ter 2 sta podrobneje opisana v poglavju Opis nivoja 1 in 2, tretji nivo pa v poglavju Struktura ter delovanje nivoja 3. Aplikacija, ki smo jo prejeli iz Printec Romunija, je bila delno izdelana na nivoju tri a bolj ali manj zaključena na nivoju ena ter dva. Za funkcionalnost MC obrokov je bilo potrebno prilagoditi le nivo tri.

2.2 Opis nivoja 1 in 2

Ker vpogleda v izvorno kodo nivoja 1 ter 2 nimamo, saj ga imajo le Romuni, so nam le predstavili, čemu je namenjen, ter določene elemente, iz katerih je sestavljen in so vezani tudi na nivo 3. Slednji elementi so:

- Batch Master (skrbi za zbirnik transakcij),
- Crypto Master (skrbi za kriptiranje PAN-a, preveranje PIN-a, dekriptiranje shranjenih vrednosti v zbirniku, uporabo Master ključa itd.),
- Duty Master (vsebuje raznorazne funkcije računanja, nastavitve vrednosti, preverjanje vrednosti, branje iz lokacij itd.),
- Periodic Event Master (skrbi za avtomatične funkcije, kot so na primer avtomatični zaključek prometa oz. zbirnika, prenos parametrov, ponovni zagon POS terminala itd.)
- Print Master (skrbi za ustrezno izpisan račun, da so pravilni pisava, razmiki; preveri, ali je papir še na voljo itd.)
- Protocol Master (skrbi za SPDH sporočila, ki se pošiljajo med POS terminalom ter Bankartom, ter procesiranje le-teh.)
- UI Master (skrbi za prikaz na zaslonu, pravilno pisavo, velikost pisave, razmike itd.)

Vsak od zgornjih elementov je povezan tudi z nivojem 3, v katerem najdemo enake elemente, le da imajo končnico »Slave« in ne »Master«. Obstajajo tudi drugi elementi, ki skrbijo za TLS enkripcijo, povezovanje z zunanjim Pin Pad-om, izmenjavo ukazov med ECR ter POS terminalom, povezavo s TMS strežnikom itd.

2.3 Struktura ter delovanje nivoja 3

Nivo 3 je sestavljen iz različnih elementov oz. razredov, ki so podrejeni nivoju 2 ter raznih xml-jev, ki se razlikujejo med sabo po strukturi in funkcionalnosti, ki jo opravljajo [6]. Spodaj našteti elementi so kasneje podrobneje opisani v podpoglavjih. Zaradi obsežnosti so predstavljeni le tisti, ki sem jih uporabil za potrebe diplomske naloge.

- BatchChart.xml,
- CheckChart.xml,
- Flowchart.xml,
- PolicyChart.xml,
- ReceiptChart.xml,
- ShiftChart.xml,
- UIChart.xml,
- BatchSlave,
- DutySlave,
- ProtocolSlave,
- UISlave.

2.3.1 Razlaga Flowchart.xml-a

Ker je Flowchart.xml najpomembnejša datoteka na nivoju 3, je tudi prva razložena, saj je potrebno razumeti, kako deluje, da lahko razumemo, kako delujejo tudi ostali elementi nivoja 1, 2 ter 3. Kot že ime xml-a nakazuje, flowchart skrbi za potek vsakega dogodka, ki se zgodi na samem POS terminalu od trenutka, ko je priklopljen v električno omrežje, ko preveri, ali je bil opravljen prenos novih parametrov, katero pisavo naj uporabi za prikaz, v kakšnem jeziku naj bo besedilo prikazano (slovensko ali angleško), kateri tip profila je uporabljen, nastavitev sekvenčne številke in tako naprej, dokler se na zaslonu prikaže glavni menu. Ta je prikazan oz. definiran glede na tip profila s pomočjo lokacije, UIChart.xml-a, UISlave-a ter UIMaster-ja. Vsaka izbira gumba s funkcijsko tipko na POS terminalu začne nov dogodek, ki mu sledi

niz ukazov, pri katerem je vsak naslednji odvisen od vrnjene vrednosti predhodnega ukaza. Vsak tako imenovan ukaz je klican z ukazom ObjectID. Slednji je sestavljen iz treh delov:

- prvi dve števili označujeta id knjižnice (01 UIMaster, 10 EMVMaster, 11 DutyMaster …),
- drugi dve označujeta funkcijo knjižnice nivoja 2,
- ostanek števil označuje funkcijo parametrizacije nivoja 3.

Obstajajo tudi izjeme, kjer prve štiri števila nakazujejo xml na nivoju 3, ostala števila pa ID v klicanem xml-ju. Da bo bolj razumljivo, spodnji primer prikazuje klic CheckChart.xml-a:

```
<CheckIfSaleWithInstallments Outcome="default" ObjectID="03223340">
    <CheckIfMCInstallments Outcome="1" ObjectID="0322901">
        <DomesticInstallments Outcome="default" Label="SaveOnlineTxn" Input="1200">
            ...
        </DomesticInstallments>
            ...
    </CheckIfMCInstallments>
</CheckIfSaleWithInstallments>
```

Oznaka »0322« nakazuje na CheckChart, 3340 pa na ID 3340 v omenjenem xml-ju. Slednji vrne vrednost 0, 1 ali -1. Če je vrednost enaka »1«, se ponovno kliče CheckChart.xml z id-jem 901, slednji pa ponovno vrne vrednost 0,1 ali -1. V primeru »1« je naslednji korak funkcija z vhodno vrednostjo 703. V primeru drugačne vrednosti je naslednji korak vezan na »default«, ki kliče funkcijo z vhodno vrednostjo 07.

Različni razredi (BatchSlave, DutySlave itd) imajo tudi različne vrednosti, ki jih lahko vrnejo. Večina jih ima le vrednosti -1, 0 ter 1, a so tudi izjeme, kot sta na primer EMVMaster ter EMVSlave, ki lahko vračata tudi trimestna števila, ki podrobneje opišejo rezultat metode (priročno pride pri procesiranju EMV podatkov kartice).

Po vsaki transakciji ali drugem opravilu (pregled zbirnika, nastavitev POS parametrov, nalaganje ključev …) se vse spremenljivke izničijo oz. postanejo prazne. Tako ne pride do raznih pomot, da bi se nam določena vrednost prenesla v naslednji transakciji ali česa podobnega. Vse spremenljivke se začnejo z »RT_« in nato z angleškim pripisom, ki skuša opisati namen spremenljivke.

Flowchart.xml skrbi za pravilno obnašanje POS terminala, da ustreza vsem zahtevam MasterCard-a, Vise ter lokalnim zahtevam banke. Le tako lahko terminal oz. aplikacija pridobi certifikat ustreznosti ter kasneje tudi uspešen integralni test Bankarta, ki omogoča produkcijo certificiranega POS terminala na izbranem trgu za potrebe banke.

Koraki, ki so bili dodani za potrebe MC obrokov, bodo podrobneje opisani v poglavju Plačilo MasterCard obrokov z uporabo vlečne steze (Magstripe), saj je trenutno pomembno le razumevanje nivoja 3, da bo kasneje lažje razumljiv opis razvoja oz. implementacije MC obročne transakcije.

2.3.2 Razlaga BatchChart.xml-a

Spodnji primer prikazuje strukturo xml-a.

```
<Table TableID="1">
    <BatchElement Size="6"                >RT_Unique</BatchElement>
    <BatchElement Size="40"               >RT_Transaction_Name</BatchElement>
    <BatchElement Size="4"                >RT_txntype</BatchElement>
    <BatchElement Size="12"               >RT_Amount</BatchElement>
    <BatchElement Size="13"               >RT_FormattedAmount</BatchElement>
    <BatchElement Size="3"                >RT_currency_name</BatchElement>
    <BatchElement Size="8"                >RT_date_from_host</BatchElement>
    <BatchElement Size="6"                >RT_time_from_host</BatchElement>
    <BatchElement Size="12"               >RT_rrn</BatchElement>
    <BatchElement Size="2"   Update="1"   >RT_ResponseCode</BatchElement>
    <BatchElement Size="20"  Crypto="1"   >RT_Pan</BatchElement>
    <BatchElement Size="4"   Crypto="1"   >RT_ExpDate</BatchElement>
    <BatchElement Size="4"                >RT_Field55Len</BatchElement>
    <BatchElement Size="304" Crypto="1"   >RT_Field55</BatchElement>
    <BatchElement Size="4"   Update="1"   >RT_isVoided</BatchElement>
    <BatchElement Size="4"   Update="1"   >RT_isUploaded</BatchElement>
    <BatchElement Size="1"                >RT_EMV</BatchElement>
    <BatchElement Size="1"                >RT_MagStripe</BatchElement>
```

TableID opisuje vrsto transakcije: Online, Offline, predavtorizacija … Vsak element ima določeno največjo velikost oz. dolžino ter ime spremenljivke. BatchChart.xml deluje s BatchSlave-om, ki skrbi, da se zapisane spremenljivke v xml-ju pravilno zapišejo na pomnilnik POS terminala. Vsaka spremenljivka je deljiva z 8. RT_Amount se tako zapiše na 16 mest v pomnilnik z začetnimi ničlami in nato z vrednostjo z dolžino 12 mest. Podatki, ki niso definirani v BatchChart.xml-ju, se po uspešni transakciji ovržejo. V primeru zavrnjene transakcije se v zbirnik ne zapiše nič. Vrednosti Crypto povedo, da je vrednost spremenljivke kriptirana in tako je na primer PAN kartice zaščiten pred možnostjo zlorabe. Vrednost Update je namenjena v primeru, ko imamo opravljen na primer nakup in kasneje transakcijo storniramo. V zbirniku se transakciji posodobi zastavica, ki pove, da je transakcija že stornirana in je tako ni mogoče ponovno stornirati. Dodatne vrednosti, ki so bile potrebne za MC obroke, bodo opisane kasneje v poglavju Plačilo MasterCard obrokov z uporabo vlečne steze (Magstripe).

2.3.3 Razlaga CheckChart.xml-a

Spodnji primer prikazuje strukturo xml-a:

```
<CheckIf CheckID="3338">
    <RTValue>941</RTValue>
    <Value>1</Value>
</CheckIf>
<CheckIf CheckID="3339">
    <RTValue>RT_SaleWithInstalments</RTValue>
    <Value>1</Value>
</CheckIf>
<CheckIf CheckID="3340">
    <RTValue>RT_SaleWithInstalmentsPerformed</RTValue>
    <Value>1</Value>
</CheckIf>
```

CheckChart.xml kličemo iz flowchart-a. CheckID z vrednostjo 3338 bi klicali z ObjectID="03223338". Zgoraj lahko vidimo dva primera, kako CheckChart.xml deluje. Deluje kot enostavni IF-ELSE pogojni ukaz v vsakem programskem jeziku. Preveri, ali se prva vrednost ujema oz. je enaka z drugo. V primeru 3338 preveri, Ali je vrednost na lokaciji pomnilnika 941 enaka vrednosti 1. V drugem primeru preveri, ali je vrednost spremenljivke enaka vrednosti 1. CheckChart lahko vrne tri različne vrednosti, in sicer:

- 1 = vrednosti sta enaki,
- 0 = vrednosti se razlikujeta,
- -1 = spremenljivka ne obstaja, vrednost ni sprejemljiva, drugo.

2.3.4 Razlaga PolicyChart.xml-a

Spodaj je primer strukture xml-a:

```
<SetValue   Constraint="11x0" Candidate="RT_NonEMVCardName"/>
<Permission  Constraint="11x5" Candidate="RT_Sale"   ControlStringPosition="0"/>
<Permission Constraint="11x5" Candidate="RT_Void"    ControlStringPosition="2"/>
<SetValue   Constraint="11x5" Candidate="RT_SaleWithInstalments"
   ControlStringPosition="3"/>
```

PolicyChart se uporablja za pravice uporabljene kartice. Vsak POS terminal ima svojo BIN tabelo, ki vključuje vse kartične produkte, ki jih je mogoče uporabljati na POS terminalu. Tako vsako kartico, ki jo stranka uporabi (na primer Karanta z vlečno stezo), POS terminal preveri BIN tabelo, kateremu produktu pripada. Produkt je lahko seveda tudi poljuben, urejen je z lokalnimi potrebami ter zahtevami banke. Tako lahko na primer NLB vse svoje kartice

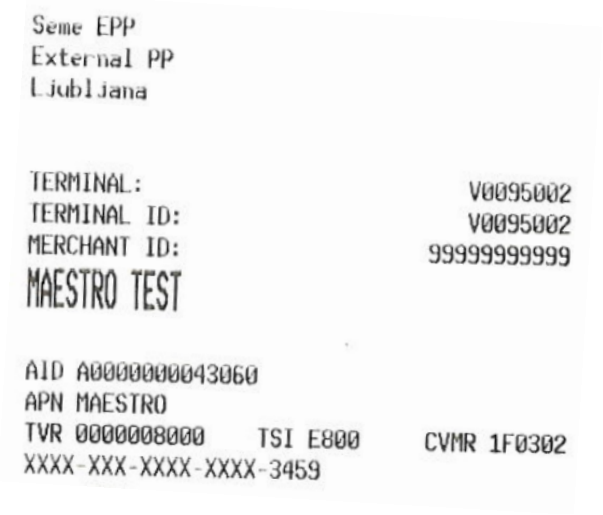
loči v eno skupino pod imenom, kot je na primer »NLB kartica«. Na omenjeno skupino bi bile nato vezane vse kartice, ki jih izdaja NLB svojim strankam. Ko POS terminal najde ustrezno BIN območje, preveri, kam pripada ter pridobi vse pravice, ki jih ima slednja kartica. Tu pride na pomoč PolicyChart.xml, ki nastavi oz. preveri specifične vrednost kartičnega produkta, ki jih želimo. V zgornjem primeru PolicyChart.xml-a lahko vidimo, da imamo na lokaciji 11x5, kjer je »x« vrednost grupe, kartice (prva kartica je na lokacijah 1100 – 1109, naslednja 1110 – 1119 …), medtem ko ControlStringPosition označuje, katero indeks mesto naj se napolni v spremenljivko Candidate. Zatem lahko poljubno preverimo vrednost spremenljivke, če jo potrebujemo. Na primer: ali lahko prebrana kartica naredi nakup z obroki ...

2.3.5 Razlaga ReceiptChart.xml-a

Spodaj je prikazana struktura xml-a.

```
<Receipt ReceiptID="03">
    <Ticket PrintIf="RT_ReprintCopy" File="I:Slip_copy.t">
        <Manage FontFile="p.pft"/>
    </Ticket>
    <Ticket File="I:Authorisation.t">
        <Manage LogoFile="I:Logo.LGO" LogoSlot="1"/>
        <Manage FontFile="p.pft"/>
    </Ticket>
    <Ticket PrintIf="RT_isQuasiCashNegated" File="I:approved.t">
        <Manage FontFile="p.pft"/>
    </Ticket>
    <Ticket PrintIf="RT_isQuasiCash"   File="I:Quasi_cash.t">
        <Manage FontFile="p.pft"/>
    </Ticket>
    <Ticket File="I:PaperFeed.t"/>
</Receipt>
```

ReceiptChart kličemo iz flowchart-a, in sicer z 0501X, kjer X predstavlja ReceiptID. V zgornjem primeru imamo prikazan ID 03, ki je namenjen odobreni čip transakciji. Račun je lahko sestavljen iz več različnih delov, pri katerem je glava enaka; spremembe so le v telesu. Odobrena ima izpisek zneska ter podatkov kartice, zavrnjena pa primerno besedilo. Pri zgornjem primeru se najprej preveri, ali je račun dvojnik oz. kopija originalnega potrdila. Če je, se izpiše najprej »Slip_copy.t«, ki ima besedilo »KOPIJA POTRDILA«. Nato se natisne »Authorisation.t«, ki vsebuje glavo. Glava vsebuje podatke, kje se POS nahaja, ID terminala, skrito številko kartice itd. Spodnje je prikazano na sliki Slika 2.1.



Slika 2.1: Primer tiska Authorisation.t.

Za vsak račun se na začetku uporabi sliko logotipa, če je potreben, ter pisavo. Pisava za Slovenijo je bila prilagojena, saj ima posebne znake oziroma šumnike. Po natisnjeni glavi računa se izpiše telo; v prikazanem primeru se lahko izpiše »approved.t« ali »Quasi_cash.t«. Izpisek telesa računa izgleda, kot prikazuje slika 2.7 spodaj:

Slika 2.2: Primer tiska approved.t.

Vsebina natisnjenega računa, ki je potreben za MC obroke, je razložena v diplomskem delu kasneje v poglavju Tiskanje računa odobrene transakcije ter zaključek transakcije.

2.3.6 Razlaga ShiftChart.xml-a

Spodaj je prikazana struktura xml-a:

```
<Shift ShiftID="3319">
    <Pair Source="RT_FormattedAmount"    Destination="RT_AmountBeforeTIPSave"/>
    <Pair Source="4010"                  Destination="RT_MaxTipAllowed"/>
    <Pair Source="v123"                  Destination="RT_NoUse"/>
</Shift>
```

ShiftChart.xml je namenjen nastavljanju vrednosti spremenljivk. Kličemo ga iz flowchart-a, in sicer z ObjectID 0303X, kjer je X ShiftID v samem dokumentu. Zgornji primer bi bil tako klican z ObjectID 03033319. ShiftChart lahko nastavlja vrednosti spremenljivki na več načinov. Lahko se kopira vsebina iz spremenljivke v novo, lahko se nastavi vrednost spremenljivke z vsebino, ki je na lokaciji (primer `Source="4010"` lahko vidimo na zgornjem prikazu), ali poljubno vrednost. Poljubna vrednost je prikazana s primerom `Source="v123"`, ko se vrednost »123« shrani v spremenljivko »RT_NoUse«. Mala črka »v« označuje, da sledi vrednost, ki jo je potrebno shraniti. Lahko je število, znaki ali mešano. Vrednosti, ki jih lahko vrne klic ShiftChart.xml-a so lahko naslednje:

- 0 = zapis je bil uspešen,
- 1 = zapis ni bil uspešen,
- -1 = napaka.

2.3.7 Razlaga UIChart.xml-a

Spodaj je prikazana struktura xml-a.

```
<Menu MenuID="61">
    <StaticText StaticTextID="0" X="-1" Y="1" Anchor="1">Sistemski meni</StaticText>
    <Button ButtonID="0" X="-2" X240="-
   2" Y="3"  Y240="3"  Outcome="2">PRENOS</Button>
    <Button ButtonID="1" X="-2" X240="-
   2" Y="5"  Y240="5"  Outcome="6">NASTAVITVE</Button>
    <Button ButtonID="2" X="-2" X240="-
   2" Y="7"  Y240="7"  Outcome="4">DATUM in URA</Button>
    <Button ButtonID="3" X="-2" X240="-
   2" Y="11" Y240="9"  Outcome="0">LOKACIJE</Button>
    <Button ButtonID="4" X="-2" X240="-
   2" Y="13" Y240="11" Outcome="3" FontFilename="fnt000.vft">BANČNI MENI</Button>
    <Button ButtonID="5" X="-2" X240="-
   2" Y="15" Y240="13" Outcome="7">CONTACTLESS</Button>
    <Button ButtonID="6" X="-2" X240="-
   2" Y="19" Y240="18" Outcome="1" FontFilename="fnt000.vft">KLJUČARJENJE</Button>
    <Button ButtonID="8" X="-2" X240="-
   2" Y="23" Y240="22" Outcome="15">JEZIK</Button>
</Menu>
```

UIChart je namenjen prikazu zaslona na POS terminalu. Je zasnovan malo drugače kot ostali xml-i, saj je tesno povezan s UISlave-om ter UIMaster-jem. Ker je tematika obsežna, bom določene stvari izpustil ter skušal razložiti osnove, ki bodo zadostovale za razumevanje. V UIChart-u lahko naredimo gumb, izpis besedila, vnosno polje ter prikaz logotipa oz. slike v bitmap obliki. Na zgornjem primeru kode xml-a je primer Sistemskega menija. Najprej imamo besedno zvezo, ki je pozicionirana na zaslonu POS terminala v prvi vrstici s sredinsko poravnavo. Oznaka X predstavlja poravnavo, ki je lahko:

- -1 = sredinska poravnava,
- -2 = desna poravnava,
- 1…21 = število zamika iz leve strani. Število 5 pomeni, da imamo najprej pet praznih znakov, nato pa besedilo, gumb…

Oznaka Y določa vrstico prikaza. Zaslon na POS terminalu ima 8 vrstic. Če imamo Y z vrednostjo več kot 8, se ta ustrezno prikaže kot nov zaslon oziroma se premikamo po meniju levo/desno z ustreznimi gumbi. Zgoraj prikazani primer bi bil na POS terminalu prikazan, kot je razvidno na sliki Slika 2.3.

Slika 2.3: Prikaz Sistemskega Meni-ja na POS terminalu.

X240 ter Y240 sta namenjena enaki funkcionalnosti, le da sta uporabljena za POS terminal Verifone Vx 675, ki ima drugačno resolucijo zaslona in tako UIMaster ustrezno prikaže enako besedilo. Oznaka Outcome je zelo pomembna, saj jo vsebuje vsak gumb. Outcome nam pove, kaj bo UIChart vrnil flowchartu glede na specifičen pritisk vsakega gumba. Glede na vrnjeno vrednost se bo flowchart ustrezno premaknil v ustrezno vejo ukazov. Potrebno je še vedeti, da imamo na voljo blokirane ter neblokirane menije. Blokiran meni je vsak, ki pričakuje, da uporabnik na njem nekaj pritisne oz. da se zgodi dogodek, ki bo vrnil neko vrednost nazaj flowchart-u, ta pa ustrezno reagiral. Neblokiran meni uporabljamo v primeru, ko želimo le

izpisati neko informacijo uporabniku ter jo prikazati na zaslonu za določen čas ali do naslednjega menija. Flowchart oz. aplikacija bo tedaj že izvajala naslednje ukaze, saj tak meni vedno vrne vrednost 0. Primer blokiranega menija je na primer vnos zneska, ne-blokiranega pa izpis besedila »Transakcija odobrena«.

2.3.8 Razlaga BatchSlave-a

BatchSlave je namenjen podpori oz. bolj podrobnim nastavitvam za shranjevanje ter iskanje po zbirniku z uporabo BatchChart-a. Nadzoruje, da se vrednosti uspešne transakcije pravilno zapišejo na pravilna mesta v pomnilniku, kjer je vsaka vrednost večkratnik števila 8.

2.3.9 Razlaga DutySlave-a

DutySlave lahko opišemo kot razširitev, ko nam CheckChart ter ShiftChart ne zadostujeta in je potrebno bolj podrobno preverjanje vrednosti, nastavitev vrednosti itd. Je zelo uporaben, saj je preprost za uporabo ter omogoča programiranje v programskem jeziku C++ in ni omejen na xml oz. predpostavke ter specifično strukturo. Pri DutySlave-u je potrebno vedeti, da vsebuje metode, kjer je vsaka registrirana z določenim ID-jem. Vsaka metoda se kliče iz flowchart-a. Klic metode iz flowchart-a je z ObjectID=1101X, kjer X predstavlja številko, ta pa metodo, ki je zapisana v datoteki Registration.cpp. Vsaka metoda vrne vrednost, ki je lahko 0, 1 ali -1. Kdaj vrnemo katero izmed vrednosti, je odvisno od samega programerja. Uporabiti oziroma uloviti jo le mora ustrezno v flowchart-u, da se nadaljujejo naslednji koraki aplikacije. Novo metodo dodamo na naslednji način: Najprej jo registriramo ter ji določimo edinstven ID in nato ji zapišemo definicijo v TxnDuties.h. Zatem ime metode ustrezno uporabimo v TxnDuties.cpp. Za boljšo predstavo so prikazane tudi na naslednjih primerih. Primer registracije nove metode v Registration.cpp dokumentu.

```
1. template class DerivedDuty<     CheckVersion,     3328>;
```

Sledi primer definicije metode v dokumentu TxnDuties.h.

```
1. int CheckVersion(iMemoryPool* pool);
```

Ter nazadnje še primer zapisane metode v dokumentu TxnDuties.cpp.

```
int CheckVersion(iMemoryPool* pool)
{
    //get POS version
    char PosVersionFull[7] = { 0, };
    pool->RetreiveInformation(564552, PosVersionFull, sizeof(PosVersionFull) - 1);
    TRACE1("POS Version: ", PosVersionFull);
    string posVersion(PosVersionFull);
```

```
    //get TMS latest version
    char TMSVersionFull[7] = { 0, };
    pool->RetreiveInformation(4018, TMSVersionFull, sizeof(TMSVersionFull) - 1);
    TRACE1("TMS Version: ", TMSVersionFull);
    string tmsVersion(TMSVersionFull);

    if (posVersion.compare(tmsVersion) == 0) {
        //same version
        return 0;
    }
    else{
        //download version
        return 1;
    }

    return 0;
}
```

Zgoraj je metoda, ki preveri, ali je verzija aplikacije na POS terminalu enaka verziji, ki je na TMS-u. Metoda se uporabi pri avtomatičnemu prenosu aplikacije, ki se zgodi, če sta aplikaciji različni. V nasprotnem primeru imamo na POS terminalu že najnovejšo verzijo in posodobitev ni potrebna.

2.3.10 Razlaga ProtocolSlave-a

ProtocolSlave je namenjen sestavljanju SPDH zahtevka, ki glede na transakcijo (nakup, storno, dobropis, predavtorizacija, zaključek pred-avtorizacije, zaključek prometa, prijava…) pravilno sestavi SPDH sporočilo oziroma zahtevek, ki ga pošlje na Bankartov procesni strežnik, in kasneje pa razčleni odgovor ter ustrezno napolni RT spremenljivke, ki jih lahko POS uporabi za naslednje korake. Vsaka transakcija ima določena polja, ki se razlikujejo tudi na način, kako je bila kartica uporabljena na POS terminalu (brezstično, stično, ročni vnos…). ProtocolSlave je ravno tako kot DutySlave zapisan s programskim jezikom C++. Polja, ki bodo uporabljena, ter polja, ki jih POS pričakuje v odgovoru, so definirana v datoteki PresentToShop.cpp. Kako se napolnijo ter razčlenijo sama polja, je definirano v datoteki SPDH_OptionalDataField.cpp. Več podrobnosti je opisano ter prikazano v poglavju 3.3 za namene MC obrokov.

2.3.11 Razlaga UISlave-a

UISlave je namenjen podpori oz. bolj podrobnim nastavitvam ter prikazu zaslona s pomočjo UIChart-a. Slednji se ne kliče iz flowchart-a, temveč je direktno povezan s UIChart-om. Z njim lahko dodatno upravljamo s prikazom zaslona POS terminala. Za primer vzamemo primer iz poglavja 2.3.7 Razlaga UIChart.xml-a, kjer je prikazan Sistemski meni. Če bi želeli dodatne pogoje, prikazati dinamično besedilo glede na vrednost določene spremenljivke, lokacije, vrednosti ali če želimo prikazati le nekaj gumbov, bi za to uporabili UISlave. Ta ima

podobno strukturo, kot jo ima DutySlave, kjer moramo definirati menu v Registration.cpp, vendar je ID oz. številka metode enaka meniju, ki ga bomo dodatno spreminjali. Tako bi bila za Sistemski meni definirana vrednost 61. Ker je metoda nato pisana v programskem jeziku C++, imamo dostop do raznih spremenljivk ter preverb, ki lahko nato določajo, kako se bo meni prikazal. UISlave pride prav v primerih, ko se na primer prebere tuja kartica in mora posledično biti vsak naslednji meni prikazan v tujem jeziku in ne slovenščini. En primer takega menija lahko vidimo na spodnjem primeru.

```
<Menu MenuID="207" TimeOutSec="1">
    <StaticText StaticTextID="0" X="-1" Y="3">Zavrnjena kartica</StaticText>

    <StaticText StaticTextID="1" X="-1" Y="3">Card not accepted</StaticText>
</Menu>
```

V primeru domače kartice, se menu 207 prikaže s StaticTextID 0, v nasprotnem primeru pa z StaticTextID 1.

2.4 Podpisna kartica ter njena uporaba

Aplikacije, ki teče na POS terminalu, ne moremo kar tako naložiti. Vso kodo je potrebno podpisati s podpisno kartico, ki jo lahko prejme podjetje glede na ustrezno specializacijo. Podpisne kartice so različne za vsak terminal oz. družino, v katero spadajo. Družina Verix Evo vključuje več različnih modelov POS terminalov ter tudi več različnih različic le-teh. Za vse se uporablja enaka podpisna kartica. Izvorno kodo, ki se bo izvajala na POS terminalu podpišemo z ustreznim orodjem (File Signing Tool). Šele nato lahko kodo s podpisom naložimo na POS terminal. Ta ob prejemu spremembe preveri, ali se podpis ujema s spremembo kode:. Šele nato spremembo uporabi. V nasprotnem primeru javi napako ter gre v tako imenovani »Tamper« način. Je le ena od varovalk, ki ščitijo POS terminal pred zlorabo ter posledično tudi stranko, ki POS terminal uporablja. Vsaka podpisna kartica je edinstvena, zato se lahko najde programerja, ki jo uporablja v zle namene, saj kartica pusti svoj pečat v podpisu, ki se naloži na POS terminal. Sliki kartice z obeh strani lahko vidimo spodaj na Slika 2.4 ter Slika 2.5: Podpisna kartica z zadnje strani., a je določen del skrit z razlogom.

Slika 2.4: Podpisna kartica s sprednje strani.

Slika 2.5: Podpisna kartica z zadnje strani.

Poglavje 3

Plačilo MasterCard obrokov z uporabo vlečne steze (Magstripe)

Pred pregledom poglavja je za bralca priporočljivo, da si najprej prebere Poglavje 2, saj bo tako lažje razumel, kako posamezni elementi delujejo ter zakaj sem jih uporabil na način, ki se mi je takrat zdel najbolj primeren.

3.1 Domači ter MasterCard obroki

Zaradi MasterCard-ovega mandata mora vsak POS, ki ga ima banka oz. kdorkoli v produkciji in je uporabljen za finančne transakcije, podpirati tudi možnost plačila domačih obrokov ter MasterCard obrokov [3]. POS mora imeti možnost, da so obroki nastavljivi ter ima možnost vklopa ali izklopa plačila z obroki.

3.1.1 Razlike med Domačimi ter MasterCard obroki

Domači obroki so predvsem namenjeni domačim karticam, predvsem MasterCard-ovim, ter Diners. MasterCard specifični obroki so namenjeni predvsem tujim MasterCard plačilnim karticam, a imajo lahko to možnost tudi domače kartice. Katera kartica bo imela možnosti plačila z obroki, je vezano na BIN tabelo. Različen je tudi sam potek transakcije ter SPDH struktura zahtevka in odgovor Bankartovega strežnika. Pri domačih obrokih je razlika med navadnim nakupom ter nakupom z obroki ta, da se v SPDH zahtevku pošlje še dodatno polje, in sicer mala črka »a«, ki predstavlja število vpisanih obrokov stranke [1]. Pri MasterCard obrokih pa je proces, v katerem se z zahtevkom nakupa pošlje dodatna zastavica. Ta lahko nato vrne plačilno shemo, ki jo mora potem stranka ustrezno uporabiti glede na izbrano shemo obročnega nakupa, ki vključuje različno število obrokov, vrednost prvega ter ostalih obrokov itd. Zatem se ponovno pošlje SPDH zahtevek na Bankartov strežnik in nato je transakcija odobrena ali zavrnjena. Sledi ustrezno tiskanje računa, ki mora vključevati vse podatke, ki so bili izbrani na plačilni shemi. [3]

3.2 Pogoji za plačilo z MasterCard obroki

Za plačilo z MC obroki imamo več nastavitev, ki jih lahko uporabimo. Vse nastavitve so nastavljene v TMS-u, ki jih kasneje POS terminal uporabi. Potrebno je določiti BIN območje, ki bo vključevalo specifične PAN-e, ki nakazujejo, da je kartica MasterCard. Karticam, ki spadajo v opisano BIN območje, nato aktiviramo možnost plačila z obroki. V TMS-u je treba določiti tudi minimalen znesek, ki je potreben, da se v SPDH zahtevku pošlje tudi zastavica, ki označuje, da lahko pri poslanemu nakupu uporabimo MC obroke, če je shema na voljo [3]. Glede na odgovor procesnega centra je potrebno v odgovoru pregledati, ali vsebuje shemo obrokov ter kakšni so pogoji in ustrezno zaključiti transakcijo.

3.3 Potek transakcije nakupa z MC obroki z uporabo vlečne steze

Ker je prikaz celotnega procesa daljši, je najprej opisan zahtevan postopek MC obročne transakcije na kratko. Podrobneje je opisan v podpoglavjih. Transakcija nakup se začne iz glavnega menija z izbiro gumba. Sledi preverba zbirnika; nastavi se ime transakcije ter omogoči CTLS transakcijo, če je POS CTLS sposoben, ter začetne vrednosti, kot je na primer sekvenčna številka.

Sledi vnos zneska. Nato sledi več pogojev, ki so lahko namenjeni nakupu z napitnino, ali POS začne s predpovezavo na Bankartov strežnik. Znesek se preuredi v obliko, ki jo bo POS prikazal stranki … Prikaže se zaslon, ki je viden na sliki Slika 3.1.

Slika 3.1: Zaslon, ki čaka stranko na uporabo kartice.

POS terminal čaka na stranko POS terminala, da ustrezno uporabi plačilno kartico. Če kartica omogoča brezstično uporabo, jo je smiselno tako tudi najprej uporabiti. Če POS zahteva, da

se kartica vstavi, se to tudi stori. Če kartico, ki ima čip in jo mi sprva uporabimo na vlečni stezi, nas bo POS terminal na to opozoril ter zahteval, da se kartica vstavi, preden se transakcija nadaljuje. Glede na uporabo kartice se ponovno nastavijo določene vrednosti, kot je na primer dodajanje imena transakcije. Preveri se sama kartica, ali že obstaja enaka transakcija z enako vrednostjo ter enakim PAN-om … Če je vse tako kot mora biti oziroma POS pričakuje, bo nadaljeval s pripravo SPDH zahtevka. Ta se sestavi na podlagi imena transakcije ter napolni polja SPDH sporočila z vrednostmi, ki so se v času transakcije nastavile. Pri sami sestavi POS opravi še nekaj dodatnih preverb. Te so podrobneje opisane s sestavljanjem zahtevka v podnaslovu 3.3.1. POS počaka ter zatem razčleni odgovor Bankarta. Če najde polje 5 oziroma možnost MC obrokov, POS shrani posamezne vrednosti v spremenljivke, ki jih bo prikazal stranki [5, 3, 4]. POS glede na odgovor prikaže eno izmed možnosti:

- Plačaj v celoti.
- Plačaj z MC obroki.
- Plačaj v celoti ali plačaj z MC obroki.

Če ima stranka možnost plačila z MC obroki in jo izbere, se glede na prejet SPDH odgovor na POS zaslonu prikaže ena možnost:

- vnos števila obrokov med prikazanima številoma,
- izbira plačilne sheme med podanimi, ki so na voljo. Za vsak nakup je glede na odgovor procesnega centra lahko med 1 ter 12 različnih plačilnih shem.

Ko se stranka odloči ter potrdi plačilne sheme ali število obrokov, se ponovno sestavi ter pošlje SPDH zahtevek. Če je odgovor zavrnjen, se transakcija zaključi, POS prikaže ustrezne informacije na zaslonu ter vrne v glavni meni. V primeru odobritve se natisne račun, ki vsebuje podatke, ki jih je stranka sprejela pri izbiri plačila [3]. Transakcija se tudi zapiše v zbirnik, povezava z Bankartom se prekine in nastavijo se določene nove vrednosti za naslednjo transakcijo. Potek je definiran s flowchart-om, ki določa vsak naslednji korak glede na vrnjeno vrednost posameznega ukaza, metode ali odločitve uporabnika, ki jo sprejme na zaslonu s pritiskom gumbov. Za to pa so ustrezno uporabljeni vsi elementi arhitekture opisane v Poglavje 2.

3.3.1 Sestavljanje prvega SPDH zahtevka z dodatno zastavico

Predpostavljamo, da je bila izbrana transakcija nakupa ter da je stranka uporabila MasterCard plačilno kartico, ki nima čipa, torej je brez CTLS ter CT podpore. Uporabljen znesek je nad minimalno vrednostjo za možnost MC obrokov. Spodaj je prikazana definicija vseh polj, ki so uporabljena za sestavljanje SPDH zahtevka za transakcijo z imenom »sale_magstripe_online«.

```
  1,  Required::Device_Type           >("sale_magstripe_online");
  2,  Required::Transmission_Number   >("sale_magstripe_online");
  3,  Required::Terminal_ID           >("sale_magstripe_online");
  4,  Required::Employee_ID           >("sale_magstripe_online");
  5,  Required::Current_Date          >("sale_magstripe_online");
  6,  Required::Current_Time          >("sale_magstripe_online");
  7,  Required::Message_Type          >("sale_magstripe_online");
  8,  Required::Message_Subtype       >("sale_magstripe_online");
  9,  Required::Transaction_Code      >("sale_magstripe_online");
 10,  Required::Processing_Flag_1     >("sale_magstripe_online");
 11,  Required::Processing_Flag_2     >("sale_magstripe_online");
 12,  Required::Processing_Flag_3     >("sale_magstripe_online");
 13,  Required::Response_Code         >("sale_magstripe_online");
'B',  ODFS::Amount_1                 >("sale_magstripe_online");
'D',  ODFS::Application_Account_Type >("sale_magstripe_online");
'F',  ODFS::Approval_Code            >("sale_magstripe_online");
'S',  ODFS::Invoice_Number           >("sale_magstripe_online");
'b',  ODFS::PIN_Customer             >("sale_magstripe_online");
'd',  ODFS::Retailer_ID              >("sale_magstripe_online");
'e',  ODFS::POS_Condition_Code       >("sale_magstripe_online");
'h',  ODFS::Sequence_Number          >("sale_magstripe_online");
'q',  ODFS::Track_2_Customer         >("sale_magstripe_online");
'z',  ODFS::Product_SubFIDs          >("sale_magstripe_online");
```

Prvi stolpec predstavlja polje, ki ni nujno enako imenovanju v stolpcu. Polja, ki se bodo zapolnila v zahtevku, so definirana s primerom spodaj.

```
…("sale_magstripe_online", New_Slot_Enumerator('B', 'D', 'S', 'b', 'd', 'e', 'h
    ', 'q', 'z'),
```

Za MC obroke nas trenutno zanima le polje »z«, ki je polje »6«. To pa je sestavljeno iz več različnih pod polij. Eno od teh je polje »0«, ki v našem primeru nakazuje, da je POS sposoben procesiranja MC obrokov za uporabljeno plačilno kartico in da je znesek večji od minimalne zahtevane vrednosti [3]. Vse to je prikazano na spodnjem primeru, medtem ko je sama koda v ProtocolSlave-u v datoteki SPDH_OptionalDataField.cpp.

```
std::string transNameSaleCheck = std::string(transaction_name);
if (transNameSaleCheck.find("sale") != string::npos) {
    TRACE("MC SALE Check!!");

    char amount[12 + 1] = { 0, };
    pool->RetreiveInformation("RT_amount", amount, sizeof(amount) - 1);
```

```
    std::string amountValue(amount);
    int amountIntValue = atoi(amountValue.c_str());

    TRACE1("Amount entered is (bigger)", amountValue.c_str());

    char amountCheck[12 + 1] = { 0, };
    pool->RetreiveInformation(4015, amountCheck, sizeof(amountCheck) - 1);
    std::string amountCheckValue(amountCheck);
    int amountIntCheckValue = atoi(amountCheckValue.c_str());

    TRACE1("Amount from position is (smaller)", amountCheckValue.c_str());

    if (amountIntValue >= amountIntCheckValue) {
        char mcInstallmentsFlag = 0x00;
        pool-
    >RetreiveInformation(4013, &mcInstallmentsFlag, sizeof(mcInstallmentsFlag));

        TRACE1("POS Allows MC Installments (0/1)", mcInstallmentsFlag);

        if (mcInstallmentsFlag == '1') { //POS allows MC installments
            TRACE("First IF is done");
            char mcInstallmentsRTEnable = 0x00;
            pool-
    >RetreiveInformation("RT_MasterCard_Installments", &mcInstallmentsRTEnable, sizeof(m
    cInstallmentsRTEnable));
            if (mcInstallmentsRTEnable == '1') { // mastercard is used
                TRACE("Adding opt. fld. '6'.  Installments flag code will be added");
                size = AddField6(tmp, '0', "0 MCI0");
            }

        }
    }
}
```

Najprej preverimo, ali je transakcija nakup, torej »sale«. Zatem primerjamo, da je znesek nakupa enak ali večji od zahtevane minimalno vrednosti, ki je nastavljena s strani TMS-a in je v pomnilniku POS terminala na lokaciji 4015. Sledi preverba lokacije 4013, ali POS podpira MC obroke. Nato imamo le še zadnji korak, ki preveri vrednost spremenljivke »RT_MasterCard_Installments«, ki se napolni s PolicyChart-om ob potegu kartice po vlečni stezi. Vrednost »1« pomeni, da je bila uporabljena MasterCard kartica, ki glede na definirano BIN območje v TMS-u omogoča plačila z MC obroki. Tako se v polje »6« doda podpolje »0« z vrednostjo »MCI0«. Del SPDH zahtevka je prikazan na spodnji sliki Slika 3.2.

Slika 3.2: Primer polja »q« ter »6« s podpolji »E«, »I« ter »0«.

3.3.2 Prejem Bankartovega odgovora na začetni zahtevek

Po poslanem zahtevku, ki je bil prikazan v naslovu 3.3.1, počakamo na odgovor Bankarta. Po sprejemu ga razčlenimo na polja ter shranimo vrednosti, ki jih najdemo v samem sporočilu [2, 3, 4, 5]. Del primera odgovora lahko vidimo na sliki Slika 3.3.

Slika 3.3: Del primera SPDH odgovora na zahtevek.

Če najdemo polje »5«, ga dodatno razčlenimo. Polje »5« je namenjeno MC obrokom, a bo lahko v prihodnosti namenjen tudi drugi funkcionalnosti. Ker polje »5« na prvi pogled nima preveč smisla, lahko spodaj najdemo krajšo razlago. Polje 5 vsebuje tako imenovane »token-e«. Token FD definira, kakšna so polja v odgovoru za možnost plačila z obroki [2]. Če je polje »5« prisotno, preverimo glavo polja, ki nam pove, koliko token-ov vsebuje ter kakšna je skupna dolžina vseh skupaj [2]. Če je prisoten token, preverimo, ali je le ta FD. Vsak token ima svojo glavo, ki določa njegovo dolžino z glavo skupaj [2]. Prikaz zgornjega opisa lahko vidimo na spodnji rešitvi.

```
case '5': {
    TRACE("found field 5 do something");
    TRACE("PROCESS MC Installments!");
    //1.) find how many tokens we have (header + how many)
    //2.) if more than 1, find ! and check if next is FD or SM, (loop over "!")
    //3.)   if u find FD, check its size and loop over 0 to end of FD
    //4.)       Within the loop, save fields in RTs
    //5.)   if u find SM, save that for RECEIPT Data to Print!
    //6.)   otherwise dont do anything

    //1.)
    //std::string countTokens = "";
    std::string fullTokenField = std::string(Source.begin(), Source.end());

    int i = 0;
    int tokenCounter = 0;
    std::string headerCount = "";
    while (i < fullTokenField.length()) {
        if (fullTokenField[i] == '&') { //get Header '&'
            headerCount = fullTokenField.substr((i + 2), 5);
            tokenCounter = (int)atoi(headerCount.c_str());
            break;
        }
        i = i + 1;
    }

    //2.)
    if (tokenCounter > 1) { //go and look for FD and SM
        TRACE("At Least 1 token is here.");
        std::string numOfOffers = "RT_NumOfOffers";
        std::string intrstRate = "RT_IntrstRate";
        std::string intrstRateFormated = "RT_IntrstRateFormated";
        std::string instlFee = "RT_InstlFee";
```

```
        std::string instlFeeFormated = "RT_InstlFeeFormated";
        std::string annualPctgRate = "RT_AnnualPctgRate";
        std::string annualPctgRateFormated = "RT_AnnualPctgRateFormated";
        std::string ttlAmtDue = "RT_TtlAmtDue";
        std::string ttlAmtDueFormated = "RT_TtlAmtDueFormated";
        std::string firstInstlAmt = "RT_FirstInstlAmt";
        std::string firstInstlAmtFormated = "RT_FirstInstallmentAmountFormated";
        std::string instlAmt = "RT_InstlAmt";
        std::string instlAmtFormated = "RT_InstlAmtFormated";
        std::string numInstl = "RT_NumInstl";

        std::string instalmentOption = "RT_InstallmentOption";
        std::string minNumInstl = "RT_MinNumInstl";
        std::string maxNumInstl = "RT_MaxNumInstl";

        //currency used
        char    CurrencyUsed[4] = { 0, };
        pool-
    >RetreiveInformation("RT_currency_name", CurrencyUsed, sizeof(CurrencyUsed) - 1);
        TRACE1("RT_currency_name", CurrencyUsed);
        std::string currencyUsed(CurrencyUsed);

        int j = 0;
        int findMark = fullTokenField.find('!');
        std::string tokenID = "";
        while (findMark != string::npos) {
            TRACE("got one !");
            tokenID = fullTokenField.substr((findMark + 2), 2);

            //3.)
            if (tokenID == "FD") { //all needed installment Fields need parsing
                //set Flag on 1
                char weHaveMCI = '1';
                pool->PackInformation(4016, &weHaveMCI, 1);
```

Ko najdemo polje FD, shranimo zastavico, da smo jo našli oziroma da si POS zapomni, da mora procesirati MC obroke po razčlenitvi odgovora z vrednostjo »1« na lokacijo »4016« v pomnilniku. Dolžino posamezna polja ter kaj vsebuje oz. kakšne so možne vrednosti, je opisano v viru [2] od strani 821 naprej, in sicer prvi del, ki vsebuje polja, ali so pravilne vrednost v poljih »FRMT-CDE«, »INSTL-PLAN-TYP« ter kakšna je vrednost v polju »INSTL-OPT«, ki je prvo od pomembnejših polij. Slednje predstavlja eno od treh možnosti plačila, ki so lahko plačilo v celoti, na obroke ali eno od obeh. Zapisana koda je prikazana spodaj.

```
//3.)
if (tokenID == "FD") { //all needed installment Fields need parsing
    //set Flag on 1
    char weHaveMCI = '1';
    pool->PackInformation(4016, &weHaveMCI, 1);

    std::string tokenFDsize = fullTokenField.substr((findMark + 4), 5); //00086

    //get 01 = MC Installments
    std::string frmtCode = fullTokenField.substr((findMark + 10), 2);
    if (frmtCode != "01") {
        TRACE("ERROR, not correct FRMT CODE!!");
        break;
```

```
    }

    //installment plan type, should be "20" - Issuer Financed
    std::string instlPlanType = fullTokenField.substr((findMark + 16), 2);
    if (instlPlanType != "20") {
        TRACE("ERROR, not correct INSTALLMENT PLAN TYPE!!");
        break;
    }
    pool-
    >PackInformation("RT_InstlPlanType", &instlPlanType[0], instlPlanType.length());

    //Payment Option
    std::string pmntOpt = fullTokenField.substr((findMark + 18), 1);
    //I = Installments only, F = Full only, B = Full or Installments
    if (pmntOpt != "I" && pmntOpt != "F" && pmntOpt != "B") {
        TRACE("ERROR, not correct PAYMENT OPTION!!");
        break;
    }
    pool->PackInformation("RT_PmntOpt", &pmntOpt[0], 1);
```

Če ima polje »INSTL-OPT« vrednost enako 1, vemo, da imamo v odgovoru na voljo le najmanjše ter največje možno število obrokov [2]. Polji sta definirani kot »MIN-NUM-INSTL« ter »MAX-NUM-INSTL«. Več je prikazano na spodnji implementaciji.

```
//Installment Option
std::string instlOption = fullTokenField.substr((findMark + 19), 1);
//0 = Varying payment amounts and/or number of payments
//1 = Varying number of payments only
if (instlOption == "1") { //get Min and Max number of Installments
    //RT_MinNumInstl
    std::string minNumInstlments = fullTokenField.substr((findMark + 20), 2);
    //RT_MaxNumInstl
    std::string maxNumInstlments = fullTokenField.substr((findMark + 22), 2);

    const char *min = minNumInstl.c_str();
    const char *max = maxNumInstl.c_str();

    pool->PackInformation(min, &minNumInstlments[0], 2);
    pool->PackInformation(max, &maxNumInstlments[0], 2);

}
const char *saveInstlOption = instalmentOption.c_str();
pool->PackInformation(saveInstlOption, &instlOption[0], 1);
```

V primeru vrednosti 0 v polju »INSTL-OPT« imamo na voljo do 12 plačilnih shem. Vsaka shema je dolga 64 znakov. Pomembna polja, ki jih vsebuje vsaka shema, so:

- številka sheme, polje »OPTION«,
- obrestna mera, polje »INRTRST-RATE«,
- provizija obrokov, polje »INSTL-FEE«,
- letna obrestna mera, polje »ANNUAL-PCTG-RATE«,

- skupni dolg zneska, polje »TTL-AMT-DUE«,
- prvi obrok, polje »FIRST-INSTL-AMT«,
- znesek obroka, polje »INSTL-AMT«,
- število obrokov, polje »NUM-INSTL«.

Polja se napolnijo z uporabe zanke. Ena iteracija zanke procesira eno shemo, katere vrednosti shrani v spremenljivke za uporabo kasneje. Podrobnosti so prikazane v spodnji implementaciji.

```
//Number of payment options
std::string numberOfOptions = fullTokenField.substr((findMark + 30), 2); //(01 -
     12) valid values
const char * numOffers = numOfOffers.c_str();
pool->PackInformation(numOffers, &numberOfOptions[0], numberOfOptions.length());

int checkOptionsCount = (int)atoi(numberOfOptions.c_str()); //01 to 12 integer
//64 = length of 1 option
std::string allOptions = fullTokenField.substr((findMark + 32), (checkOptionsCount * 64)
    ); //64=length of 1 option

int i = 0;
int optionPointer = 0;
const char *save = 0;
//go over options
while (i < checkOptionsCount) {
    //integer "x" to str with leading "0"
    int x = i + 1;
    string rtName = intToString(x);

    //1.)Interest Rate
    std::string interestRateString = allOptions.substr(optionPointer, 5); //with 2 decim
    al places
    std::string intRateStringOriginal = interestRateString;
    saveToRTString(intrstRate, rtName, intRateStringOriginal, pool);

    std::string intrRateStrFormated = formatValue(interestRateString);
    intrRateStrFormated = intrRateStrFormated + " %";
    saveToRTString(intrstRateFormated, rtName, intrRateStrFormated, pool);

    //2.)Installment Fee
    std::string instlFeeString = allOptions.substr((optionPointer + 5), 12);
    std::string instlFeeStringOriginal = instlFeeString;
    saveToRTString(instlFee, rtName, instlFeeStringOriginal, pool);

    std::string instlFeeStrFormated = formatValue(instlFeeString);
    instlFeeStrFormated = instlFeeStrFormated + " " + currencyUsed;
    saveToRTString(instlFeeFormated, rtName, instlFeeStrFormated, pool);

    //3.)Annual Percentage Rate
    std::string annualPctgRateString = allOptions.substr((optionPointer + 17), 5);
    std::string annualPctgRateStringOriginal = annualPctgRateString;
    saveToRTString(annualPctgRate, rtName, annualPctgRateStringOriginal, pool);

    std::string annualPctgRateStrFormated = formatValue(annualPctgRateString);
```

```
    annualPctgRateStrFormated = annualPctgRateStrFormated + " %";
    saveToRTString(annualPctgRateFormated, rtName, annualPctgRateStrFormated, pool);

    //4.)Total Amount Due
    std::string ttlAmtDueString = allOptions.substr((optionPointer + 22), 12);
    std::string ttlAmtDueStringOriginal = ttlAmtDueString;
    saveToRTString(ttlAmtDue, rtName, ttlAmtDueStringOriginal, pool);

    std::string ttlAmtDueStrFormated = formatValue(ttlAmtDueString);
    ttlAmtDueStrFormated = ttlAmtDueStrFormated + " " + currencyUsed;
    saveToRTString(ttlAmtDueFormated, rtName, ttlAmtDueStrFormated, pool);

    if (instlOption == "0") {
        //5.) First Installment Amount
        std::string firstInstallmentAmount = allOptions.substr((optionPointer + 34), 12)
    ;
        std::string firstInstlAmtStringOriginal = firstInstallmentAmount;
        saveToRTString(firstInstlAmt, rtName, firstInstlAmtStringOriginal, pool);

        std::string firstInstlAmountFormated = formatValue(firstInstallmentAmount);
        firstInstlAmountFormated = firstInstlAmountFormated + " " + currencyUsed;
        saveToRTString(firstInstlAmtFormated, rtName, firstInstlAmountFormated, pool);

        //6.)Installment Amounts
        std::string installmentAmounts = allOptions.substr((optionPointer + 46), 12);
        std::string instlAmtStringOriginal = installmentAmounts;
        saveToRTString(instlAmt, rtName, instlAmtStringOriginal, pool);

        std::string installmentAmountsFormated = formatValue(installmentAmounts);
        installmentAmountsFormated = installmentAmountsFormated + " " + currencyUsed;
        saveToRTString(instlAmtFormated, rtName, installmentAmountsFormated, pool);

        //7.)Number of Installments
        int numberOfInstallments = (int)atoi(allOptions.substr((optionPointer + 58), 2).
    c_str()); //02-99
        saveToRT(numInstl, rtName, numberOfInstallments, pool);

    }

    optionPointer = optionPointer + 64; //pointer to start of next Option
    i = i + 1; //to next Pointer
}
```

Vsaka spremenljivka se shrani z imenom, ki predstavlja tudi njeno definicijo. Imena spremenljivk so enaka po imenu vse do zaključka, kjer imajo pripis številke možnosti sheme. Tako se vrednosti ne zamešajo ter so smiselno shranjene, da bodo lahko kasneje prikazane uporabniku POS terminala na zaslonu.

Za pregledom celotnega FD token-a je potrebno preveriti, ali je v polju »5« tudi token SM [2]. Slednji je namenjen tekstovnemu besedilu, ki ga je potrebno izpisati na račun v primeru odobrene transakcije. Celotno besedilo se shrani, kot je prikazano v implementaciji.

```
//5.)
else if (tokenID == "SM") { //message to print
    TRACE("POTATO MESSAGE HERE TO:DO");
    TRACE("SAVE message to RT for printing");
    std::string tokenSMdata = fullTokenField.substr((findMark + 10), 4); //00018
```

```
    int sizeToReadSM = (int)atoi(tokenSMdata.c_str());
    std::string dataToSave = fullTokenField.substr((findMark + 14), sizeToReadSM);

    pool-
    >PackInformation("RT_MCInstallmentMsgPrint", &dataToSave[0], (dataToSave.length()));


}
//6.) find next token
findMark = fullTokenField.find('!', findMark + 1);
```

Trenutno so tokeni, ki niso FD ali SM ignorirani, saj se polja »5« trenutno za druge funkcionalnosti ne uporablja.

3.3.3 Prikaz ter postopek možnosti MC obrokov

Sedaj, ko imamo vse podatke, ki so potrebni za prikaz samih MC obrokov, to tudi storimo. Najprej prekinemo povezavo zs Bankartovim strežnikom ter pripravimo možno storniranje transakcije, ker je prvi odgovor nekako že odobritev transakcije, ki bi se posledično zabeležila v zbirnik. Problem nastane, ko ima uporabnik na primer možnost le plačilo z obroki in ne tudi v celoti in tako lahko posledično zavrne ter prekine transakcijo. Flowchart po analizi SPDH odgovora najprej preveri, ali je transakcija odobrena. Nato preveri, ali vsebuje možnost plačila z MC obroki. Če jih ni, se izpiše odobreno na zaslon in transakcija se shrani v zbirnik ter natisne se račun. V nasprotnem primeru se MC obroki morajo procesirati [3]. Pri tem lahko sklenemo s štirimi različnimi postopki za dokončanje transakcije, in sicer:

- Shema MC obrokov je izbrana; pošlji nov SPDH zahtevek v preverjanje.
- Zaključi odobreno transakcijo, saj je uporabnik izbral plačilo v celoti (če je možno).
- Uporabnik je zavrnil obvezne obroke; zaključi transakcijo kot zavrnjeno.
- Napaka pri procesiranju, napačen odgovor …

Zgoraj opisane možnosti so prikazane tudi na Slika 3.4 in Slika 3.3 spodaj.

```
<ParseResponse Outcome="0" ObjectID="093101">
    <!--Host Approved, check MC Installments-->
    <CheckMCInstallmentSent Outcome="0" ObjectID="03223400">
        <ProcessMCInstallments Outcome="1" Label="ProcessMCInstallments">
            <!--MC Instl. Approved (Send back message), write Download flag to file if needed-->
            <MCInstallmentsOnlineAuthProc Outcome="1" Label="MCInstallmentsOnlineAuthProc">
                <ParseResponse Outcome="0" ObjectID="093101">...</ParseResponse>
                <!--Send Transaction Error-->
                <BeepNotOK Outcome="-4" ObjectID="11013336">...</BeepNotOK>
                <!--Host Unreachable-->
                <BeepNotOK Outcome="-5" ObjectID="11013336">...</BeepNotOK>
                <!--Online Authorisation Procedure Failed-->
                <BeepNotOK Outcome="-2" ObjectID="11013336">...</BeepNotOK>
            </MCInstallmentsOnlineAuthProc>
            <!--Approved (Full payment), write Download flag to file if needed-->
            <Approved Outcome="0" ObjectID="030650">...</Approved>
            <!--User Declined Mandatory Installments-->
            <MarkMCIRefused Outcome="-1" ObjectID="03033408">...</MarkMCIRefused>
            <!--Error-->
            <MarkMCIRefused Outcome="default" ObjectID="03033408">...</MarkMCIRefused>
        </ProcessMCInstallments>
        <!--HostApproved, MC Installments were NOT used-->
        <WriteDwlNeededToFile Outcome="default" ObjectID="030650">...</WriteDwlNeededToFile>
```

Slika 3.4: Prikaz začetnega postopka uporabe MC obrokov.

Možnost, kaj lahko uporabnik izbere, je prikazana na zaslonu v spodnji obliki. Zaslon je prikazan glede na vrednost polja »PMNT-OPT«. [2] Slednji ima lahko tri vrednosti:

- I = Plačilo je možno le z obroki.
- F = Plačilo je možno le v celoti brez obrokov.
- B = Plačilo je možno v celoti ali z obroki.

Na spodnji Slika 3.5 je prikazan zaslon v primeru, če imamo možnost plačila v celoti ali z obroki.

Slika 3.5: Prikaz možnosti plačila MC obrokov.

Zgornji prikaz je definiran z UIChart-om, kot je prikazano z implementacijo.

```
<Menu MenuID="3401" DisableCancel="1">
    <StaticText StaticTextID ="0" X="-1" Y="1" >MASTERCARD</StaticText>
    <StaticText StaticTextID ="1" X="-
    1" Y="3" FontFilename="fnt000.vft">Možnosti plačila</StaticText>
    <Button ButtonID="0" X="-
    2" Y="5" Outcome="0" FontFilename="fnt000.vft">PLAČAJ V CELOTI</Button>
    <Button ButtonID="1" X="-
    2" Y="7" Outcome="1" FontFilename="fnt000.vft" Focus="1">PLAČAJ Z OBROKI</Button>

    <StaticText StaticTextID ="5" X="-1" Y="1">MASTERCARD</StaticText>
    <StaticText StaticTextID ="6" X="-1" Y="3">Payment Options</StaticText>
    <Button ButtonID="5" X="-2" Y="5" Outcome="0">IN FULL</Button>
    <Button ButtonID="6" X="-2" Y="7" Outcome="1">WITH INSTALLMENTS</Button>
</Menu>
```

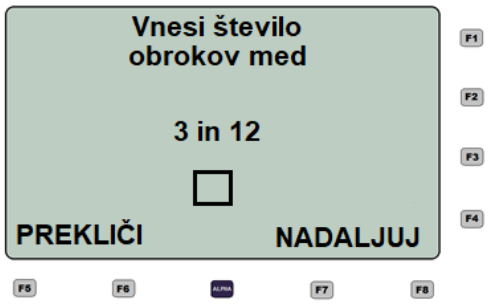
Pravilo POS obnašanja je, da vse naslednje zaslone prikaže v tujem jeziku, ko prebere tujo kartico. Da ne bi imeli prevelikega podvajanja kode, lahko definiramo en zaslon v UIChart-u z elementi za domač ter tuj jezik. Nato uporabimo UISlave, ki ima možnost določanja, kateri elementi specifičnega zaslona se bodo prikazali. Tako lahko tam preverimo, ali je kartica tuja. Nato na zaslonu ustrezno prikažemo elemente. Zgornji primer bi tako v UISlave-u izgledal takole, kot prikazuje del implementacije spodaj.

```
bool    MCPaymentOption::SetMenuRules(vector<int>& Buttons, vector<int>& EditBoxes ...)
{
    TRACE("MCPaymentOption");
    /* Default value */
    m_Skip = false;

    //get RT_ICC_Check
    char    fileContent[3 + 1] = { 0, };
    pool->RetreiveInformation("RT_ICC_Check", fileContent, sizeof(fileContent) - 1);
    TRACE1("RT_ICC_Check", fileContent);

    std::string iccCheck(fileContent);

    if (iccCheck.length() == 3 && iccCheck != "705") //ENGLISH language
    {
        TRACE("MCPaymentOption ENGLISH");
        StaticTexts.push_back(5);
        StaticTexts.push_back(6);
        Buttons.push_back(5);
        Buttons.push_back(6);

    }
    else //Slovenian language
    {
        TRACE("MCPaymentOption SLOVENIAN");
        StaticTexts.push_back(0);
        StaticTexts.push_back(1);
        Buttons.push_back(0);
        Buttons.push_back(1);
    }

    return true;
}
```

Vsi zasloni, ki bodo prikazani v naslednjih korakih, so izdelani na podoben način kot tisti, ki so opisani zgoraj. V primeru plačila v celoti se transakcija ustrezno zaključi, saj je že odobrena. Pomembni podatki se shranijo v zbirnik. Nato pa natisne račun za trgovca ter imetnika kartice.

3.3.4 Prikaz ter postopek pri možnosti vnosa števila MC obrokov

Če je uporabnik imel možnost le plačila z obroki ali jih je izbral, je eden izmed možnih dogodkov ta, da uporabnik izbere število obrokov glede na najmanjšo ter največjo možno število. Ti dve polji sta že shranjeni v poljih, saj sta bili prisotni v polju »MIN-NUM-INSTL« ter »MAX-NUM-INSTL«. [2] Prikaz vnosa števila obrokov je prikazan na spodnji Slika 3.6; vrednosti spremenljivk najmanjšega ter največjega števila sta le za boljšo predstavo.



Slika 3.6: Prikaz vnosa števila obrokov.

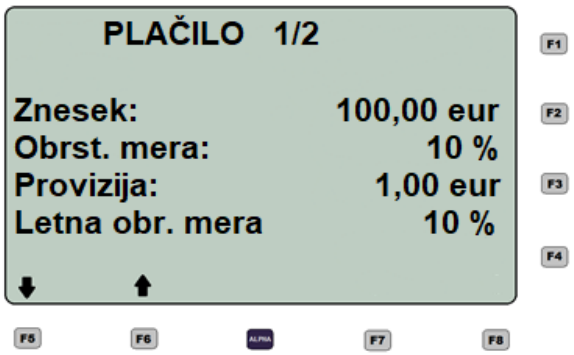
Zaslon je definiran v UIChartu z implementacijo:

```
1.  <Menu MenuID="3402">
2.      <StaticText StaticTextID ="0" X="-
    1" Y="1" FontFilename="fnt000.vft">Vnesi število</StaticText>
3.      <StaticText StaticTextID ="1" X="-1" Y="2" >obrokov med</StaticText>
4.      <StaticText StaticTextID ="2" X="7" Y="4" TextFromPool="1">RT_MinNumInstl</Stati
    cText>
5.      <StaticText StaticTextID ="3" X="10" Y="4" >in</StaticText>
6.      <StaticText StaticTextID ="4" X="13" Y="4" TextFromPool="1">RT_MaxNumInstl</Stat
    icText>
7.      <EditBox EditBoxID="0" Tag="RT_MCPickedNumberOfPayments" X="-
    1" Y="6" TextDirection="1" Focus="1" InputType="2" Outcome="0" NonEmpty="1">__</Edit
    Box>
8.      <Button ButtonID="0" X="-2" Y="8" Outcome="0" >NADALJUJ</Button>
9.      <Button ButtonID="1" X="1" Y="8" Outcome="20" FontFilename="fnt000.vft">PREKLIČI
    </Button>
10.
11.     <StaticText StaticTextID ="5" X="-1" Y="1">Enter number of</StaticText>
12.     <StaticText StaticTextID ="6" X="-1" Y="2" >installments between</StaticText>
13.     <StaticText StaticTextID ="7" X="-
    1" Y="4" TextFromPool="1">RT_MinNumInstl</StaticText>
14.     <StaticText StaticTextID ="8" X="-1" Y="4" >and</StaticText>
15.     <StaticText StaticTextID ="9" X="-
    1" Y="4" TextFromPool="1">RT_MaxNumInstl</StaticText>
```

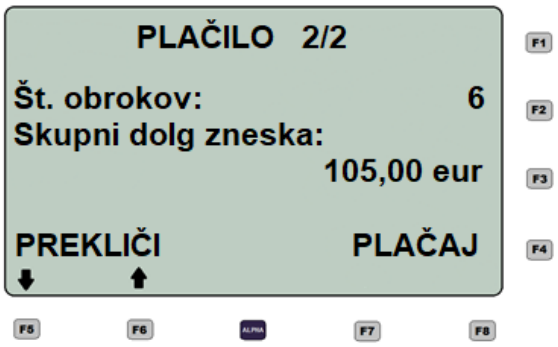
```
16.     <EditBox EditBoxID="5" Tag="RT_MCPickedNumberOfPayments" X="-
    1" Y="6" TextDirection="1" Focus="1" InputType="2" Outcome="0" NonEmpty="1">__</Edit
    Box>
17.     <Button ButtonID="5" X="-2" Y="8" Outcome="0" >CONTINUE</Button>
18.     <Button ButtonID="6" X="1" Y="8" Outcome="20">CANCEL</Button>
19. </Menu>
```

Tu je potrebno vedeti, da ima vnosno polje zastavico, ki določa, da mora polje vsebovati vrednost, preden lahko uporabnik izbere gumb »NADALJUJ«.

Zatem se preveri vpisana vrednost, če je ustrezna. V nasprotnem primeru na to uporabnika opozori ter ponovno zahteva vpis vrednosti. Če je vrednost ustrezna, se uporabniku prikaže dvostranski zaslon, ki prikaže vse informacije, s katerimi se bo moral uporabnik strinjati za plačilo. V tem primeru bi zaslona izgledala tako, kot prikazujeta Slika 3.7 ter Slika 3.8.



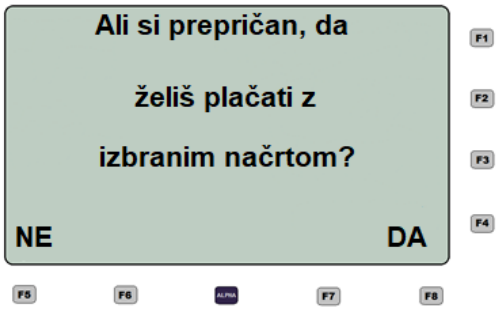
Slika 3.7: Prva stran izbranih vrednosti.



Slika 3.8: Druga stran izbranih vrednosti.

Kot se lahko vidi, lahko izbiro »PLAČAJ« uporabnik izbere le na drugem, torej zadnjem zaslonu, kjer so prikazani vsi podatki.

V primeru preklica uporabnika vrne na izbiro plačila z obroki ali v celoti, prikazano s sliko Slika 3.5. S pritiskom na gumb »PLAČAJ« se uporabniku prikaže še potrditveni zaslon. Primer prikaza je na Slika 3.9 [3].



Slika 3.9: Zadnja pritrditev pred plačilom.

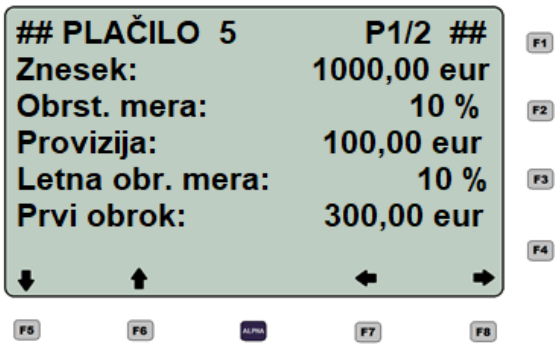
S pritiskom na gumb »DA« se vse izbrane vrednosti shranijo v nove spremenljivke, ki bodo nato uporabljene za shranjevanje v zbirnik, izpis na račun in sprva za pošiljanje v novem SPDH zahtevku na Bankartov procesni center. Kratek izsek opisanega postopka lahko vidimo na spodnji Slika 3.10 flowchart-a z imeni, ki skušajo predstaviti vsak korak, ki se izvede.

```
<Label Name="MasterCardInstallmentsOptionSelection">
    <CheckInstlOptMode ObjectID="03223401">
        <ManyOptions Outcome="0" ObjectID="03033406">...</ManyOptions>
        <NumberOfPaymentsOnly Outcome="1" ObjectID="03033406">
            <EnterNumber Outcome="default" ObjectID="01013402">
                <CheckTheInput Outcome="0" ObjectID="11013402">
                    <FinalAgreement Outcome="0" ObjectID="01013404">
                        <SamePage Outcome="10" TurnNode="01013404"/>
                        <SamePage Outcome="11" TurnNode="01013404"/>
                        <PreviousPage Outcome="96" TurnNode="01013404"/>
                        <NextPage Outcome="95" ObjectID="01013405">
                            <MasterCardInstallmentsUserInput Outcome="0" Label="MasterCardInstallmentsUserInput">
                                <UserAccept Outcome="1" ObjectID="03033414">
                                    <SaveForFID5 Outcome="default" ObjectID="03033418">...</SaveForFID5>
                                </UserAccept>
                                <UserCancel Outcome="default" ReturnOutcome="0"/>
                            </MasterCardInstallmentsUserInput>
                            <SamePage Outcome="95" TurnNode="01013405"/>
                            <PreviousPage Outcome="96" TurnNode="01013404"/>
                            <Cancel Outcome="20" TurnNode="01013402"/>
                            <DoNotContinue Outcome="default" ReturnOutcome="0"/>
                        </NextPage>
                        <DoNotContinue Outcome="default" ReturnOutcome="0"/>
                    </FinalAgreement>
                    <InvalidValue Outcome="1" ObjectID="01013403">...</InvalidValue>
                    <Error Outcome="default" ObjectID="01013403">...</Error>
                </CheckTheInput>
                <GoBackToOptions Outcome="default" ObjectID="01013401">...</GoBackToOptions>
            </EnterNumber>
        </NumberOfPaymentsOnly>
```

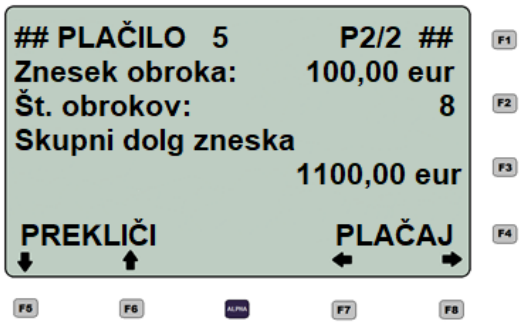
Slika 3.10: Prikaz postopka korakov izbire števila MC obrokov v flowchart-u.

3.3.5 Prikaz ter postopek pri možnosti plačilnih shem MC obrokov

Če dobimo v SPDH odgovoru sheme, ki so lahko uporabljene za plačilo nakupa, jih je potrebno na POS terminalu prikazati uporabniku. Slednji se nato odloči za shemo, ki mu najbolj ustreza. Shem je lahko ena ali največ dvanajst. Vsaka je dolga 64 znakov, ki predstavljajo vse potrebne podatke za transakcijo. Vsaka shema je predstavljena na dveh straneh. Shema se za prikaz prikaže z uporabo DutySlave-a, ki nastavi vrednosti za številko sheme, UIChart-om ter UISlave-om za prikaz na zaslonu, ShiftChart-om za nastavljanje začetnih vrednosti ter CheckChart-om za določene pogoje. Flowchart vse skupaj nadzira oz. definira, kaj bo po vsakem koraku sledilo. Primer prikaza sheme se lahko vidi na Slika 3.11 ter Slika 3.12 .



Slika 3.11: Stran ena plačilne sheme 5.



Slika 3.12: Stran 2 plačilne sheme 5.

S puščicami gor ter navzdol se pomikamo med stranjo ena in dva. Puščici levo in desno sta namenjeni spreminjanju plačilne sheme. Puščica levo bi nam prikazala plačilno shemo 4, desno pa 6. Premikanje ter določanje potek, ki se zgodi ob pritisku gumbov, je prikazan tudi s flowchart-om na Slika 3.13.

```
<Label Name="DisplayOffersInstallments">
    <InitializeOffersBuffers ObjectID="03033419">
        <SetMCIOffersBuffers Outcome="0" ObjectID="11013403">
            <ShowMCIInfo Outcome="0" ObjectID="01013409">
                <BrowseDown Outcome="95" ObjectID="01013410">
                    <FinalAgreement Outcome="0" Label="MasterCardInstallmentsUserInput">
                        <UserAccept Outcome="1" ObjectID="03033416">
                            <Back Outcome="default" ReturnOutcome="1"/>
                        </UserAccept>
                        <UserCancel Outcome="default" ReturnOutcome="2"/>
                    </FinalAgreement>
                    <Cancel Outcome="20" ReturnOutcome="2"/>
                    <DisplayDown Outcome="95" TurnNode="01013410" />
                    <DisplayDownButton Outcome="11" TurnNode="01013410" />
                    <DisplayTop Outcome="96" TurnNode="01013409" />
                    <DisplayTopButton Outcome="10" TurnNode="01013409" />
                    <DisplayPrevious Outcome="97" ObjectID="11013405">...</DisplayPrevious>
                    <DisplayNext Outcome="98" ObjectID="11013404">...</DisplayNext>
                    <Error Outcome="default" TurnNode="11013403"/>
                </BrowseDown>
                <BrowseDownButton Outcome="11" ObjectID="01013410">...</BrowseDownButton>
                <BrowseUp Outcome="96" TurnNode="01013409"/>
                <BrowseUpButton Outcome="10" TurnNode="01013409"/>
                <DisplayPrevious Outcome="97" ObjectID="11013405">
                    <LoopAgain Outcome="default" TurnNode="11013403" />
                </DisplayPrevious>
                <DisplayNext Outcome="98" ObjectID="11013404">
                    <LoopAgain Outcome="default" TurnNode="11013403" />
                </DisplayNext>
                <Cancel Outcome="default" ReturnOutcome="2"/>
            </ShowMCIInfo>
            <ReturnError Outcome="-1" ReturnOutcome="-1"/>
        </SetMCIOffersBuffers>
    </InitializeOffersBuffers>
</Label>
```

Slika 3.13: Prikaz klicev funkcij v flowchart-u.

Ko se uporabnik odloči za shemo, ki jo želi, se mu pojavi še zaslon za potrditev, ki ga lahko vidimo na sliki 3.21. V primeru izbire gumba »DA« se izbrane vrednosti ustrezno shranijo za kasnejšo uporabo.

3.3.6 Sestavljanje novega SPDH zahtevka

Sedaj, ko je uporabnik izbral želeno shemo, so vse vrednosti shranjene v spremenljivkah. Aplikacija spremeni ime transakcije tako, da ji na konec doda »`_mcInstl`«. Nato se ponovno sestavi SPDH zahtevek, ki pa sedaj doda tudi polje »5«. V polju doda tudi token FE, ki vsebuje podatke sheme, ki jo je uporabnik izbral. Definirana polja za ime transakcije so definirana s ProtoclSlave-om v datoteki Present_To_Shop.cpp, kot je prikazano z implementacijo kode spodaj.

```
  1,  Required::Device_Type          >("sale_magstripe_online_mcInstl");
  2,  Required::Transmission_Number  >("sale_magstripe_online_mcInstl");
  3,  Required::Terminal_ID          >("sale_magstripe_online_mcInstl");
  4,  Required::Employee_ID          >("sale_magstripe_online_mcInstl");
  5,  Required::Current_Date         >("sale_magstripe_online_mcInstl");
  6,  Required::Current_Time         >("sale_magstripe_online_mcInstl");
  7,  Required::Message_Type         >("sale_magstripe_online_mcInstl");
  8,  Required::Message_Subtype      >("sale_magstripe_online_mcInstl");
  9,  Required::Transaction_Code     >("sale_magstripe_online_mcInstl");
 10,  Required::Processing_Flag_1    >("sale_magstripe_online_mcInstl");
 11,  Required::Processing_Flag_2    >("sale_magstripe_online_mcInstl");
 12,  Required::Processing_Flag_3    >("sale_magstripe_online_mcInstl");
 13,  Required::Response_Code        >("sale_magstripe_online_mcInstl");
'B',  ODFS::Amount_1                 >("sale_magstripe_online_mcInstl");
'D',  ODFS::Application_Account_Type >("sale_magstripe_online_mcInstl");
'F',  ODFS::Approval_Code            >("sale_magstripe_online_mcInstl");
'S',  ODFS::Invoice_Number           >("sale_magstripe_online_mcInstl");
'b',  ODFS::PIN_Customer             >("sale_magstripe_online_mcInstl");
'd',  ODFS::Retailer_ID              >("sale_magstripe_online_mcInstl");
'e',  ODFS::POS_Condition_Code       >("sale_magstripe_online_mcInstl");
'h',  ODFS::Sequence_Number          >("sale_magstripe_online_mcInstl");
'q',  ODFS::Track_2_Customer         >("sale_magstripe_online_mcInstl");
'x',  ODFS::Tokens_SubFIDs           >("sale_magstripe_online_mcInstl"); // s5
'z',  ODFS::Product_SubFIDs          >("sale_magstripe_online_mcInstl"); // s6
```

Polje »x« predstavlja Polje 5, ki vsebuje token FE s podatki. Spodnja delna definicija prikazuje polja, ki jih bo SPDH zahtevek vseboval za opisano transakcijo.

```
("sale_magstripe_online_mcInstl", New_Slot_Enumerator('B', 'D', 'F', 'S', 'b', 'd', 'e',
    'h', 'q', 'x', 'z')
```

Kako se polje »5« oziroma token FE v njem zapolni pravilno, je prikazano z implementacijo pod sliko Slika 3.14 v primeru, ko imamo število obrokov (vsebuje polja iz »if« pogoja), ter zatem implementacijo kode v primeru, ko se uporablja izbrana shema (vsebuje polje z »else«). Ker so določene vrednosti neuporabne ampak morajo biti zaradi pravilne dolžine, saj je vsako območje točno začrtano in predstavlja določeno vrednost, so za pravilno strukturo dodane ničle, kjer je potrebno [2]. Lahko se uporabi tudi prazen znak, kot je na primer presledek. Primer polja »5« oz. token-a FE v SPDH zahtevku za izbrano število obrokov lahko vidimo na Slika 3.14.

```
Field 5: [FS5& 0000200114! FE00092 00202400000002000000000002000000000020000000000002000000020000011M97800000000M]
```

Slika 3.14: Primer poslanega polja »5« s token-om FE v SPDH zahtevku.

```
// FID 5 - 0 to 2000 bytes  //tokens, more or less used for MC Installments
Payload_Data Tokens_SubFIDs(const Business_Target& Targeted_Need, iMemoryPool* pool)
{
    TRACE("empty field 5 for MCInstallments");

    TRACE("Adding Field 5");
    string field5("5");

    //1.) Header + token number + full length
```

```
    field5 = field5 + "& " + "00002" + "00114";

    //2.) Token FE + lenght
    field5 = field5 +"! FE" + "00092 ";

    //3.) FRMT-CDE
    field5 = field5 + "00";

    //4.) INSTL-PLAN-TYP
    char    InstlPlanType[3] = { 0, };
    pool-
    >RetreiveInformation("RT_InstlPlanType", &InstlPlanType, sizeof(InstlPlanType) -
     1);
    TRACE1("RT_InstallmentOption", InstlPlanType);
    std::string instlPlanTyp(InstlPlanType);
    field5 = field5 + instlPlanTyp;

    char    InstlOpt[2] = { 0, };
    pool->RetreiveInformation("RT_InstallmentOption", InstlOpt, sizeof(InstlOpt) - 1);
    TRACE1("RT_InstallmentOption", InstlOpt);
    std::string instlOption(InstlOpt);

    if (instlOption == "1") { //Number of payments ONLY
        //1.) NUM-INSTL
        {
            char    NumInstl[3] = { 0, };
            pool->RetreiveInformation("RT_MCNumOfPymts", &NumInstl, sizeof(NumInstl) -
     1);
            TRACE1("RT_MCNumOfPymts", NumInstl);
            std::string numOfInstl(NumInstl);
            field5 = field5 + numOfInstl;
        }

        //2.) FIRST-INSTL-AMT
        field5 = field5 + "000000000000";

        //3.) INSTL-AMT
        field5 = field5 + "000000000000";

        //4.) TTL-AMT-DUE
        {
            char    TtlAmtDue[13] = { 0, };
            pool->RetreiveInformation("RT_TtlAmtDue", &TtlAmtDue, sizeof(TtlAmtDue) -
     1);
            TRACE1("RT_TtlAmtDue", TtlAmtDue);
            std::string ttlAmountDue(TtlAmtDue);
            field5 = field5 + ttlAmountDue;
        }

        //5.) INSTL-FEE
        {
            char    InstFee[13] = { 0, };
            pool->RetreiveInformation("RT_InstlFee", &InstFee, sizeof(InstFee) - 1);
            TRACE1("RT_InstlFee", InstFee);
            std::string installmentsFee(InstFee);
            field5 = field5 + installmentsFee;
        }

        //6.) ANNUAL-PCTG-RAT
        {
            char    AnnPctgRt[6] = { 0, };
            pool-
    >RetreiveInformation("RT_AnnualPctgRate", &AnnPctgRt, sizeof(AnnPctgRt) - 1);
```

```
            TRACE1("RT_AnnualPctgRate", AnnPctgRt);
            std::string annualPctgRate(AnnPctgRt);
            field5 = field5 + annualPctgRate;
        }

        //7.) INTRST-RAT
        {
            char    IntrRat[6] = { 0, };
            pool->RetreiveInformation("RT_IntrstRate", &IntrRat, sizeof(IntrRat) - 1);
            TRACE1("RT_IntrstRate", IntrRat);
            std::string interestRate(IntrRat);
            field5 = field5 + interestRate;
        }

        //8.) INTRST-RAT-PRD
        field5 = field5 + "M";

        //9.) ISS-CRNCY-CDE
        field5 = field5 + "978";

        //10.) FIRST-PMNT-DAT
        field5 = field5 + "000000";

        //11.) NUM-MM-GRATUITY
        field5 = field5 + "00";

        //12.) INSTL-GRATUITY-PRD
        field5 = field5 + "M";

    }
    else { //PLAN to BUILD
```

Na zgoraj opisanem drugem primeru, ko imamo izbrano shemo in ne le števila obrokov, lahko vidimo, da imamo poleg izbranih vrednosti uporabnika še določena druga polja. Skupno število znakov, ki predstavljajo vrednosti polja FE, je 92 [2]. Dodatna uporabna polja so naslednja:

- INTRST-RAT-PRD, kjer je W = tedensko, M = mesečno ter Y = letno plačilo,
- ISS-CRNCY-CDE, ki predstavlja kodo uporabljene valute,
- FIRST-PMNT-DAT, ki predstavlja datum prvega obroka,
- INSTL-GRATUITY-PRD, ki predstavlja enoto prehodnega obdobja pred prvim plačilom obroka.

Ko uporabnik izbere shemo in ne vnese le števila obrokov, ki se pošlje v polju »5«, prikazano s implementacijo kode, ki je prikazana spodaj, kako se polja ustrezno preberejo iz spremenljivk ter na koncu sestavijo sam token FE, ter nato polje »5« (implementacija je v odseku »else« iz zgornjega primera).

```
else { //PLAN to BUILD
    //1.) NUM-INSTL
    {
        char    NumInstl[3] = { 0, };
        pool->RetreiveInformation("RT_MCNumOfPymts", &NumInstl, sizeof(NumInstl) - 1);
        TRACE1("RT_MCNumOfPymts", NumInstl);
        std::string numOfInstl(NumInstl);
        field5 = field5 + numOfInstl;
    }

    //2.) FIRST-INSTL-AMT
    {
        char    FirstInstlAmt[13] = { 0, };
        pool-
    >RetreiveInformation("RT_FirstInstlAmnt", &FirstInstlAmt, sizeof(FirstInstlAmt) -
     1);
        TRACE1("RT_FirstInstlAmnt", FirstInstlAmt);
        std::string firstInstalAmt(FirstInstlAmt);
        field5 = field5 + firstInstalAmt;
    }

    //3.) INSTL-AMT
    {
        char    InstlAmt[13] = { 0, };
        pool->RetreiveInformation("RT_InstlAmnt", &InstlAmt, sizeof(InstlAmt) - 1);
        TRACE1("RT_InstlAmnt", InstlAmt);
        std::string instalAmt(InstlAmt);
        field5 = field5 + instalAmt;
    }

    //4.) TTL-AMT-DUE
    {
        char    TtlAmtDue[13] = { 0, };
        pool->RetreiveInformation("RT_TtlAmtDue", &TtlAmtDue, sizeof(TtlAmtDue) - 1);
        TRACE1("RT_TtlAmtDue", TtlAmtDue);
        std::string ttlAmountDue(TtlAmtDue);
        field5 = field5 + ttlAmountDue;
    }

    //5.) INSTL-FEE
    {
        char    InstFee[13] = { 0, };
        pool->RetreiveInformation("RT_InstlFee", &InstFee, sizeof(InstFee) - 1);
        TRACE1("RT_InstlFee", InstFee);
        std::string installmentsFee(InstFee);
        field5 = field5 + installmentsFee;
    }

    //6.) ANNUAL-PCTG-RAT
    {
        char    AnnPctgRt[6] = { 0, };
        pool->RetreiveInformation("RT_AnnualPctgRate", &AnnPctgRt, sizeof(AnnPctgRt) -
     1);
        TRACE1("RT_AnnualPctgRate", AnnPctgRt);
        std::string annualPctgRate(AnnPctgRt);
        field5 = field5 + annualPctgRate;
    }

    //7.) INTRST-RAT
    {
        char    IntrRat[6] = { 0, };
        pool->RetreiveInformation("RT_IntrstRate", &IntrRat, sizeof(IntrRat) - 1);
        TRACE1("RT_IntrstRate", IntrRat);
```

```
        std::string interestRate(IntrRat);
        field5 = field5 + interestRate;
    }

    //8.) INTRST-RAT-PRD
    field5 = field5 + "M";

    //9.) ISS-CRNCY-CDE
    field5 = field5 + "978";

    //10.) FIRST-PMNT-DAT
    field5 = field5 + "000000";

    //11.) NUM-MM-GRATUITY
    field5 = field5 + "00";  s

    //12.) INSTL-GRATUITY-PRD
    field5 = field5 + "M";
}

field5 += '\x1c';
TRACE1("Field 5", field5.c_str());
Payload_Data    temp(true);
temp.Raw_Buffer.insert(temp.Raw_Buffer.end(), field5.begin(), field5.end());
return temp;
```

Pri odgovoru na MC obroke ponovno dodamo tudi zastavico v polje »6«, le da je tokrat vrednost podpolja »0« enaka 9 MCI0 in ne več 0 MCI0 [1]. Po sestavljanju celotnega SPDH zahtevka se slednji po vzpostavitvi povezave ponovno pošlje na Bankartov procesni strežnik. Počakamo na odgovor za nadaljevanje transakcije.

3.3.7 Prejem zadnjega SPDH odgovora

Ko ponovno prejmemo odgovor Bankartovega procesnega strežnika, ta odgovor tudi razčlenimo. Delni primer odgovora lahko najdemo na Slika 3.15: Delni zajem celotnega SPDH odgovora..

```
FS00000000FSF005468FSgODOBRENO : 005468FS6RSE021RSI978RSeMSIBQN6C91128
```

Slika 3.15: Delni zajem celotnega SPDH odgovora.

Pomembno je vedeti, da v drugem odgovoru (če je odobren) ne pridobimo določenih novih vrednostih, kot so na primer določena podpolja polja »6« [3]. Polje za referenčno številko ter polja kartice itd. se ne smejo spremeniti, saj je uporabljena kartica ista, kot je bila za prvi SPDH zahtevek. Drugi odgovor je le potrditev uporabljene sheme, ki je že vezan na prvi zahtevek. Celotna opisana procedura velja za eno transakcijo in ne dve posamezni, zato so polja ob prvem odgovoru pomembna, saj se na njih sklicujemo, če želimo opraviti storno transakcije, vpogled vanjo itd. Drugi del transakcije, ki so MC obroki, so – enostavno povedano – le dodatne vrednosti, kako bodo vplivale na stanje uporabljene MasterCard

kartice. Zato je tudi podvrsta transakcije tipa »S«, ki pomeni shrani. Če nam Bankartov procesni strežnik vrne zavrnjeno z drugim odgovorom, naredimo stornacijo celotne transakcije, uporabnika POS terminala o tem obvestimo in se nato vrnemo v glavni meni. V primeru odobritve se natisne račun in vrednosti transakcije se shranijo v zbirnik.

3.3.8 Tiskanje računa odobrene transakcije ter zaključek transakcije

V primeru odobritve drugega SPDH odgovora Bankarta moramo transakcijo le še uspešno zabeležiti v zbirnik, prikazati ustrezna sporočila na zaslonu ter shraniti vrednosti, ki jih potrebujemo za tiskanje računa, narediti njegovo kopijo, kadarkoli je transakcija zabeležena v zbirniku, za mogočo storniranje transakcije. Ker je smiselno najprej shraniti podatke v zbirnik, je to tudi prvi korak, ki ga opravimo. Spremenljivke, ki se shranijo v Zbirnik so določene z BatchChart-om, mesta, na katere se shranijo v pomnilnik, pa z BatchSlave-om. Ker se shrani večja količina podatkov, so na sliki 3.34 prikazane le vrednosti, ki so bile uporabljene specifično za MC obroke.

```
<BatchElement Size="40"          >RT_GroupByCardName</BatchElement>
<BatchElement Size="1"           >RT_Receipt_From_OneOption</BatchElement>
<BatchElement Size="1"           >RT_Receipt_From_MultiOption</BatchElement>
<BatchElement Size="1"           >RT_MCIFullPayment</BatchElement>
<BatchElement Size="2"           >RT_MCNumOfPymts</BatchElement>
<BatchElement Size="5"           >RT_IntrstRate</BatchElement>
<BatchElement Size="12"          >RT_InstlFee</BatchElement>
<BatchElement Size="5"           >RT_AnnualPctgRate</BatchElement>
<BatchElement Size="12"          >RT_FirstInstlAmnt</BatchElement>
<BatchElement Size="12"          >RT_InstlAmnt</BatchElement>
<BatchElement Size="12"          >RT_TtlAmtDue</BatchElement>
<BatchElement Size="2"           >RT_MCPickedNumberOfPayments</BatchElement>
<BatchElement Size="8"           >RT_IntrstRateFormated01</BatchElement>
<BatchElement Size="16"          >RT_InstlFeeFormated01</BatchElement>
<BatchElement Size="8"           >RT_AnnualPctgRateFormated01</BatchElement>
<BatchElement Size="16"          >RT_TtlAmtDueFormated01</BatchElement>
<BatchElement Size="8"           >RT_IntrstRateFormated</BatchElement>
<BatchElement Size="2"           >RT_NumOfInstallments</BatchElement>
<BatchElement Size="16"          >RT_InstlFeeFormated</BatchElement>
<BatchElement Size="8"           >RT_AnnualPctgRateFormated</BatchElement>
<BatchElement Size="16"          >RT_FirstInstallmentAmountFormated</BatchElement>
<BatchElement Size="16"          >RT_InstallmentAmountFormated</BatchElement>
<BatchElement Size="16"          >RT_TtlAmtDueFormated</BatchElement>
<BatchElement Size="512"         >RT_MCInstallmentMsgPrint</BatchElement>
```

Slika 3.16: Shranjena vrednosti polj za MC obroke.

Nikoli niso vse zgoraj definirane vrednosti zapolnjene. Odvisno je, kakšna shema je bila uporabljena, ali je bilo vneseno le število obrokov, ali je bilo prisoten token SM itd. Polja, ki so definirana, uporabljamo le za ponovno tiskanje računa, če je to potrebno. Za storniranje

transakcije ali dodajanje vsote, ko POS izvede zaključek zbirnika, potrebujemo le osnovne informacije, kot so sekvenčna številka, vsota, vrsta transakcije …

Po uspešnem shranjevanju transakcije natisnemo tudi račun, ki je lahko v treh oblikah. Ena izpiše listek, ki je enak ostalim transakcijam tipa nakup, le da ima tudi pripis na dnu, ki pravi »MC obroki so bili na voljo, ampak niso bili uporabljeni« [3]. V nasprotnem primeru se izpišejo vrednosti, kot so definirane po MasterCard mandatu [3]. Zatem se POS postavi nazaj v glavni meni, kjer je že pripravljen na novo transakcijo. Na koncu je sledilo še obsežno testiranje vseh funkcionalnosti POS terminala, da vse deluje točno tako, kot mora, preden se je kodo združilo z glavno, ki je bila kasneje poslana sna integralni test na sBankart.

3.4 Problemi

Glavni problem diplomske naloge je bil razumevanje arhitekture. Zaradi njene obsežnosti in pomanjkanja dokumentacije je bilo potrebno veliko časa, preden sem bil dovolj samozavesten storiti spremembo v manjših xml-ih, kaj šele flowchart-u, ki trenutno obsega okrog 17.000 vrstic in vsaka sprememba v njem spremeni določen postopek izvajanja aplikacije. Ali dodati v že delujočo aplikacijo takšen poseg, kot so MC obroki, je potrebno predhodno dobro razmisliti. Vprašanje je, kako bo implementacija potekala, da se z nobeno spremembo ne pokvari nečesa, kar že deluje, kot mora, saj je certifikacija POS terminala takrat že bila opravjena. Potrebno se je bilo tudi naučiti, kako POS deluje, kakšna so sporočila, ki se pošiljajo med POS terminalom ter Bankartovim procesnim centrom, kako se procesira branje kartice itd [1]. Ko je bilo razumevanje dovolj veliko, tudi sama implementacija MC obrokov ni bila preveč zahtevna. Potrebno je le biti pazljiv, da se držiš vseh specifikacij, katerih se je potrebno držati [2, 3, 4, 5].

3.5 Zanimivosti

Kot zanimivost lahko izpostavim naslednje, in sicer da imamo v pilotnem testiranju POS terminale. Trenutno smo že v drugi fazi pilota. Pri prvi nismo imeli nobenih večjih težav, sedaj pri drugi pa trenutno nimamo težav. Kmalu se bo pilotno testiranje še razširilo na več POS terminalov. Poleg implementacije MC obrokov sem dodal tudi nekaj drugih funkcionalnosti, kot sta uporaba blagajnikov, začetek transakcije s pritiskom številke. Prilagodil sem že obstoječo rešitev aplikacije, ki se izvaja na Master POS-u, ki skrbi za varno nalaganje PIN ključev v varni sobi, ampak to so že druge zgodbe. Pred kratkim sem tudi

prilagodil POS aplikacijo za podjetje A1, ki je v procesu menjave POS terminalov, ki so namenjeni za plačilo položnic v svojih poslovalnicah.

Trenutno sem za POS aplikacijo (da deluje kot je pričakovano, odpravo napak ter testiranje) zadolžen jaz, medtem ko mi nadrejeni pomaga z usklajevanjem z Bankartom ter posodobitvami POS aplikacije, ki jih je potrebno naložiti na TMS testni strežnik za potrebe testiranja. Občasno stopim v stik za pomoč tudi s programerji iz Printec Romunije za nivo 2, katerega se trenutno tudi sam učim. Če je Printec Romunija preveč zasedena z delom, priskoči na pomoč programer iz Printec Makedonije, ki je na voljo za osnove nivoja 2, ki so v večini primerov zadostne.

Poglavje 4

Sklepne ugotovitve

Implementacija MC obrokov mi je bila zelo všeč, saj je predstavljala izziv, ki so mi ga na Printec S.I. d.o.o. zaupali, da ga bom uspešno izvedel. Ugotovil sem, da je bilo razumevanje arhitekture obstoječe rešitve dobro spoznati, preden sem začel s spremembami, ki bi lahko slabo vplivale na celotno delovanje POS terminala. Ob pisanju kode sem jo skušal karseda komentirati in uporabljati imena spremenljivk na razumljiv način. Tako je kdorkoli lahko analiziral, kaj sem naredil, ter to tudi razumel. Slednje se vidi na slikah, ki jih lahko najdemo v diplomski nalogi. Ves čas sem pazil na lovljenje napak ter skušal uloviti najmanjše robne primere, ki posledično ne bi vplivali na POS aplikacijo in kodo karseda optimizirati, da ni nepotrebnih ukazov ter podvajanja kode. Na vse to sem se ves čas spominjal iz teorije na fakulteti ter praksi, ko so nas profesorji ter asistenti skušali pripraviti na nivo, ki ga želijo slovenska ali tuja podjetja. Prav tako je zelo pomembno sodelovanje, saj je bilo v diplomski nalogi vpleteno osebje na Bankartu, celotna ekipa na podjetju Printec Slovenija programerja iz Printec Romunija, ki sta pripomogla k razumevanju arhitekture nivoja 3. Veliko je bilo dela pod pritiskom, saj je imelo podjetje roke, do katerih naj bi bila funkcionalnost delujoča in s tem tudi motivacija večja.

Na koncu videti delujočo funkcionalnost nekje v produkciji, ki jo uporablja nekdo, ki te ne pozna, je zelo zadovoljiv občutek, ki ti na obraz zariše nasmeh z mislijo, da si to sam naredil in je nekaj uporabnega.

Literatura

[1] Printec S.I. d.o.o. ACI Universal Payments, Standard POS Device Message Specification, avgust 2017. [Dostopano 26.1.2019] [Interno gradivo]

[2] Printec S.I. d.o.o. ACI Universal Payments, Tokens Manual Base24, Release 6.0 Post v10, julij 2017. [Dostopano 26.1.2019] [Interno gradivo]

[3] Printec S.I. d.o.o. MasterCard, Slovenia Operations Bulletin No. 1, MasterCard Instalment Payment Service for Slovenia, julij 2016. [Dostopano 25.1.2019] [Interno gradivo]

[4] Printec S.I. d.o.o. MasterCard, MasterCard Installment Payment Service – Point-of-interaction Eligibility and Enablement Implementation Guide, februar 2016. [Dostopano 25.1.2019] [Interno gradivo]

[5] Printec S.I. d.o.o. MasterCard, MasterCard Installment Payment Service – Instalment Calculation and Clearing Implementation Guide, februar 2016. [Dostopano 25.1.2019] [Interno gradivo].

[6] Yergeau, Francois, Cowan, John. Extensible Markup Language (XML) 1.1 (Second Edition) W3C Recommendation 16 August 2006. Dosegljivo: http://www.w3pdf.com/W3cSpec/XML/2/REC-xml11-20060816.pdf. [Dostopano 20.1.2019]