UNIVERSITATEA TITU MAIORESCU

FACULTATEA: INFORMATICĂ
DEPARTAMENT: INFORMATICĂ
Programa de studii: INFORMATICĂ
DISCIPLINA: INTELIGENȚĂ ARTIFICIALĂ

IA – *Testul de evaluare nr.* 16

Applet JavaCard

Grupa	Numele și prenumele	Semnătură student	Notă evaluare

Data: ____/____/_____

CS-I dr.ing.
Lucian Ștefăniță GRIGORE

Conf.dr.ing.
Iustin PRIESCU

Ș.L.dr.ing.
Dan-Laurențiu GRECU

Cuprins

1. ADVANCED NFC

1.1 Lucrul cu Technologiile Tag acceptate

Atunci când se lucrează cu etichete NFC și dispozitive Android, formatul principal utilizat pentru a citi și scrie date pe tag-uri este NDEF. Atunci când un dispozitiv scanează o etichetă cu datele NDEF, Android oferă suport în preluarea mesajului și livrarea într-un `NdefMessage` atunci când este posibil.

Există cazuri, cu toate acestea, atunci când se scanează o etichetă care nu conține date NDEF sau atunci când datele NDEF nu au putut fi mapate la un tip MIME sau URI. În aceste cazuri, este necesar de a deschide un canal de comunicare direct cu tag-ul și citirea/scrierea să se efectueze cu propriul protocol (în bytes brut). Android oferă suport generic pentru aceste cazuri de utilizare cu pachetul `android.nfc.tech`, care este descris în tabelul 1.

Se poate utiliza metoda `getTechList ()` pentru a determina tehnologiile susținute de tag-ul respectiv și de a crea un obiect `TagTechnology` corespunzător cu unul din cele furnizate de `android.nfc.tech`

Tabelul 1. Suportul Tehnologiilor Tag

ass	Description
`TagTechnology`	The interface that all tag technology classes must implement.
`NfcA`	Provides access to NFC-A (ISO 14443-3A) properties and I/O operations.
`NfcB`	Provides access to NFC-B (ISO 14443-3B) properties and I/O operations.
`NfcF`	Provides access to NFC-F (JIS 6319-4) properties and I/O operations.
`NfcV`	Provides access to NFC-V (ISO 15693) properties and I/O operations.
`IsoDep`	Provides access to ISO-DEP (ISO 14443-4) properties and I/O operations.
`Ndef`	Provides access to NDEF data and operations on NFC tags that have been formatted as NDEF.
`NdefFormatable`	Provides a format operations for tags that may be NDEF formattable.

Următoarele Tehnologii de tag-uri, mai recente, nu sunt necesare pentru a avea un suport suplimentar pe dispozitivele Android.

Tabelul 2. Suport opțional pentru tehnologiile tag

Class	Description
MifareClassic	Provides access to MIFARE Classic properties and I/O operations, if this Android device supports MIFARE.
MifareUltralight	Provides access to MIFARE Ultralight properties and I/O operations, if this Android device supports MIFARE.

1.1.1 Lucrul cu tehnologii tag și intenția ACTION_TECH_DISCOVERED

Atunci când un dispozitiv scanează o etichetă care are date NDEF pe ea, dar nu a putut fi mapată ca un MIME sau URI, sistemul de expediere a tagurilor, încearcă să înceapă o activitate cu intenția `ACTION_TECH_DISCOVERED`. `ACTION_TECH_DISCOVERED` este, de asemenea, utilizat atunci când o etichetă cu date non-NDEF este scanată.

Această rezervă permite să se lucreze cu datele de pe etichetă, direct în cazul în care sistemul de expediere a etichetei nu a putut efectua o analiză internă. Pașii de bază atunci când se lucrează cu tehnologii tag sunt:

1. Filtru pentru o intenție `ACTION_TECH_DISCOVERED` specificând tehnologiile tag. Vezi Filtering for NFC intents[1] pentru mai multe informații. În general, sistemul de expediere tag-uri încearcă să înceapă o intenție `ACTION_TECH_DISCOVERED` atunci când un mesaj de NDEF nu poate fi mapat ca un tip MIME sau URI, sau în cazul în care eticheta scanată nu conține date NDEF. Pentru mai multe informații cu privire la modul în care acest lucru este determinat, a se vedea The Tag Dispatch System[2].
2. Când solicitarea primește intenția, se obține obiectul `Tag` pentru obiectul de intenție:

```
Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
```

3. Se obține o instanță a unui `TagTechnology, apelând unul dintre get` metodelor aferente claselor din pachetul `android.nfc.tech`. Se pot enumera tehnologiile suportate de tag-ul de apel `getTechList () mai` înainte de a apela un `get`. De exemplu, pentru a obține o instanță a `MifareUltralight` dintr-o `Tag`, se procedează în felul următor:

```
MifareUltralight.get(intent.getParcelableExtra(NfcAdapter.EXTRA_TAG));
```

1.1.1 Citirea și scrierea de tag-uri

Citire și scrierea unei etichete NFC implică obținerea tag-ului de la intenția comunicarea și apoi deschiderea tag-ului. Trebuie definit propriul stack de protocol pentru a citi și scrie date pe etichetă. Următorul exemplu arată cum să se lucreze cu o etichetă MIFARE Ultralight.

```
package com.example.android.nfc;
import android.nfc.Tag;
import android.nfc.tech.MifareUltralight;
import android.util.Log;
import java.io.IOException;
import java.nio.charset.Charset;
public class MifareUltralightTagTester {
    private static final String TAG = MifareUltralightTagTester.class.getSimpleName();
    public void writeTag(Tag tag, String tagText) {
        MifareUltralight ultralight = MifareUltralight.get(tag);
        try {
            ultralight.connect();
            ultralight.writePage(4, "abcd".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(5, "efgh".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(6, "ijkl".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(7, "mnop".getBytes(Charset.forName("US-ASCII")));
        } catch (IOException e) {
            Log.e(TAG, "IOException while closing MifareUltralight...", e);
        } finally {
            try {
```

[1] http://developer.android.com/guide/topics/connectivity/nfc/nfc.html#tech-disc
[2] http://developer.android.com/guide/topics/connectivity/nfc/nfc.html#tag-dispatch

```
                ultralight.close();
            } catch (IOException e) {
                Log.e(TAG, "IOException while closing MifareUltralight...", e);
            }
        }
    }
    public String readTag(Tag tag) {
        MifareUltralight mifare = MifareUltralight.get(tag);
        try {
            mifare.connect();
            byte[] payload = mifare.readPages(4);
            return new String(payload, Charset.forName("US-ASCII"));
        } catch (IOException e) {
            Log.e(TAG, "IOException while writing MifareUltralight
            message...", e);
        } finally {
            if (mifare != null) {
              try {
                  mifare.close();
              }
              catch (IOException e) {
                  Log.e(TAG, "Error closing tag...", e);
              }
            }
        }
        return null;
    }
}
```

1.2 Utilizarea sistemului Dispatch de prim-plan

Sistemul de expediere în prim-plan permite interceptarea prioritară a intenției și revendicarea față de alte activități care se ocupă de aceeași intenție. Folosind acest sistem, se presupune că anterior se construiește o structură cu puține date pentru sistemul Android pentru a putea trimite intențiile adecvate pentru o solicitare. Pentru a activa sistemul de expediere prim-plan, se procedează, astfel:

1. se adăugă următorul cod în `onCreate ()`:

i) se creează un obiect `PendingIntent` astfel încât sistemul Android poate completa cu detaliile necesare eticheta atunci când este scanată.

```
PendingIntent pendingIntent = PendingIntent.getActivity(
  this, 0, new Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
```

ii) se declară filtrele intenție care să se ocupe de intențiile care trebuie intercepta. Sistemul de expediere prim-plan verifică filtrele intenție specificate care sunt primite atunci când aparatul scanează o etichetă. Dacă se potrivește, atunci aplicația se ocupă de intenția respectivă. În cazul în care nu se potrivește, sistemul de expediere prim-plan revine la sistemul de expediere a intenției. Specificarea `null` pentru o serie de filtre intenție și filtre de tehnologie, specifică faptul că se dorește filtrarea pentru toate etichetele care rezervă intenția la `TAG_DISCOVERED`. Fragmentul de cod de mai jos se ocupă de toate tipurile MIME pentru `NDEF_DISCOVERED.`

```
IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
    try {
        ndef.addDataType("*/*");    /* Handles all MIME based dispatches.
                                       You should specify only the ones that you need. */
    }
    catch (MalformedMimeTypeException e) {
        throw new RuntimeException("fail", e);
    }
    intentFiltersArray = new IntentFilter[] {ndef, };
```

iii) configurarea unei serii de tehnologii tag la o solicitare specifică înseamnă că se dorește apelarea metodei `Object.class.getName ()` pentru a obține clasa tehnologiei dorită ca suport.

```
techListsArray = new String[][] { new String[] { NfcF.class.getName() } };
```

2. se vor suprascrie următoarele activități callback ale ciclului de viață și care se adaugă în logica pentru a activa și dezactiva expedierea prim-plan atunci când activitatea pierde `(onPause ())` și își recapătă focalizarea pe `(onResume ()). EnableForegroundDispatch ()` trebuie să devină principal și numai atunci când activitatea este în prim-plan (apel în `onResume ()` garantează acest lucru). De asemenea, trebuie să pună în aplicare apelul invers `onNewIntent` pentru a procesa datele de la tag-ul NFC scanat:

```
public void onPause() {
    super.onPause();
    mAdapter.disableForegroundDispatch(this);
}

public void onResume() {
    super.onResume();
    mAdapter.enableForegroundDispatch(this, pendingIntent, intentFiltersArray, techListsArray);
}

public void onNewIntent(Intent intent) {
    Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
    //do something with tagFromIntent
}
```

2. HOST-BASED CARD EMULATION

Există foarte multe dispozitive Android, care oferă funcționalitate NFC ca suport pentru emularea NFC a cardurilor. În cele mai multe cazuri, cardul este emulat de un cip introdus separat în dispozitiv, numit un *element de siguranță*. Multe cartele SIM oferite de operatorii de transport fără fir conțin, de asemenea, un element securizat.

Android 4.4 introduce o metodă suplimentară de emulare a cardurilor care nu implică un element securizat, numit *emulare carte **host-based***. Acest lucru permite ca orice aplicație Android să realizeze un card virtual care să comunice direct cu cititorul NFC. Acest document descrie modul în care funcționează emularea unui card (HCE)-gazdă bazat pe tehnologia Android și care poate dezvolta o aplicație care emulează un card NFC folosind această tehnică.

2.1 Card Emulation with a Secure Element

Când cardul este prevăzut cu un element de siguranță și are loc emularea pentru NFC, cardul va fi virtualizat și prevăzut în elementul securizat pe dispozitivul printr-o aplicație Android. Apoi, când utilizatorul ține dispozitivul peste un terminal NFC, controlerul NFC trimite în traseele dispozitivului toate datele din cititorul direct la elementul securizat. Figura 1 ilustrează acest concept.

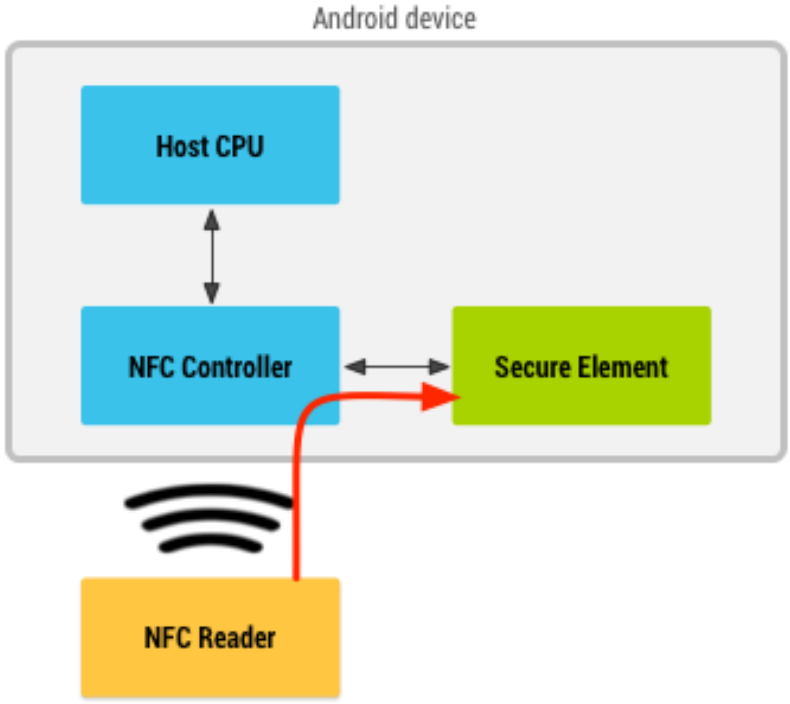

Figura 1. NFC - emularea unui card care deține un element securizat.

Elementul securizat realizează comunicarea cu terminalul NFC, astfel încât, nici o cerere a dispozitivului Android nu este implicată în tranzacție. După ce tranzacția este completă, o aplicație a dispozitivului Android poate interoga elementul securizat direct pentru statutul tranzacției și va notifica utilizatorul.

2.2 Host-based Card Emulation

Când un card NFC este emulat folosind *host-based card emulation*, datele sunt dirijate la CPU gazdă pe care aplicațiile dispozitivului Android rulează direct, în loc de rutare a cadrelor de protocol NFC la un element securizat. Figura 2 ilustrează modul în care funcționează *host-based card emulation*.

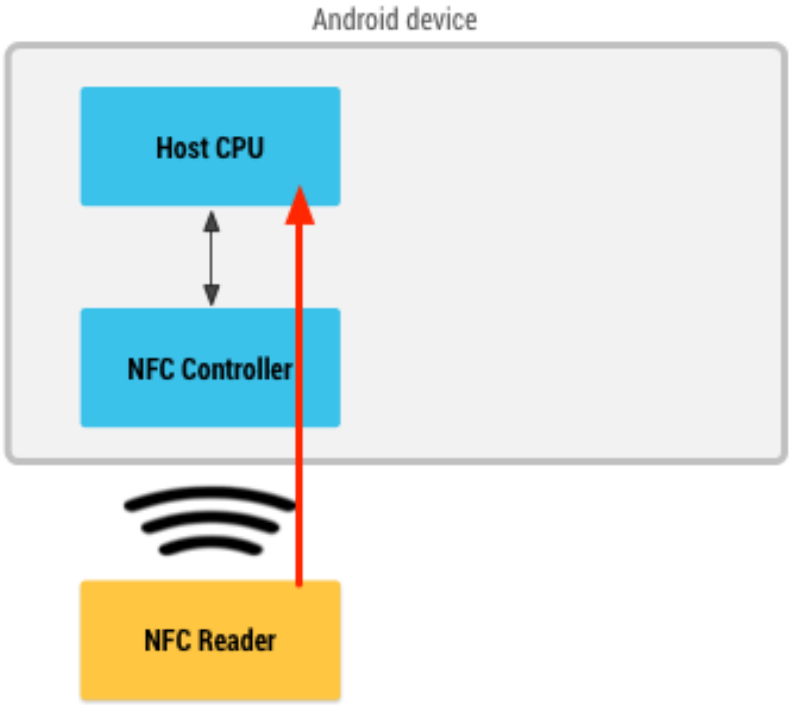

Figura 2. NFC - emulare card fără element securizat.

2.3 Supported NFC Cards and Protocols

Standardele NFC oferă suport pentru mai multe protocoale diferite și există diferite tipuri de carduri care pot fi emulate.

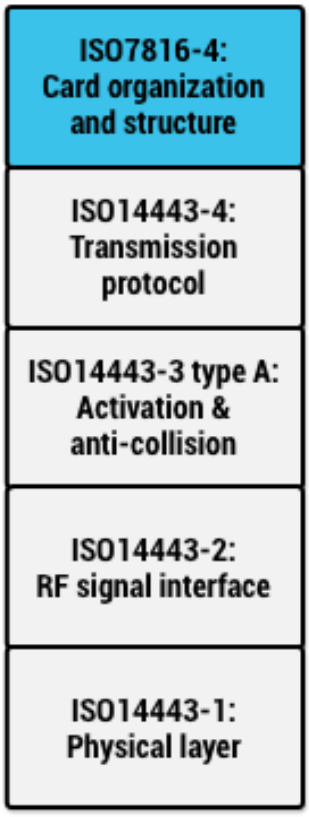

Fig. 3

Android 4.4 suportă mai multe protocoale care sunt, deja, comune pe piața de astăzi. Multe carduri contactless existente sunt realizate pe baza acestor protocoale, cum ar fi cardurile de plată contactless.

Aceste protocoale sunt, de asemenea, sprijinite de multe cititoare NFC de pe piață, inclusiv dispozitivele Android NFC, care funcționează ca cititoare se (vezi calasa `IsoDep`). Acest lucru permite să se construiască și să se implementeze o soluție ***end-to-end NFC*** în jurul valorii de HCE folosind numai dispozitive Android.

Mai exact, Android 4.4 are suportul necesar pentru emularea cardurilor care se bazează pe specificația NFC-Forum ISO-DEP (bazat pe ISO / IEC 14443-4) și procesul de aplicare a unități de date Protocol (APDUs), astfel cum este definit în caietul de sarcini ISO / IEC 7816-4. Dispozitivele Android sunt mandatate cu emularea ISO-DEP numai în partea de sus a tehnologiilor NFC-A (ISO / IEC 14443-3 A Type). Tehnologia Suport pentru NFC B (B ISO / IEC 14443-4 este opțională. Stratificarea tuturor acestor specificații este prezentată în figura 3.

2.4 HCE Servicii

Arhitectura HCE implementată pe dispozitivele Android se bazează pe componentele `Android Service` de completare (cunoscute sub numele de "servicii HCE"). Unul dintre avantajele cheie ale unui serviciu este că se poate rula în fundal, fără nici o interfață cu utilizatorul.

Aceasta este o potrivire naturală pentru multe aplicații HCE, cum ar fi loialitate sau tranzit de carduri, cu care utilizatorul nu ar trebui să lanseze aplicația pentru a-l folosi. În schimb, atingând dispozitivul de cititorul NFC începe serviciul corect (dacă nu rulează deja) și execută tranzacția în fundal. Desigur, este liber să lanseze UI suplimentare (cum ar fi notificări de utilizator), doar dacă acest lucru este strict necesar.

2.4.1 Selecția de servicii

Atunci când utilizatorul deschide canalul de comunicație cu un dispozitiv de la un cititor NFC, sistemul dispozitivului Android trebuie să știe ce serviciu HCE are cititorul NFC, de fapt să înțeleagă cum trebuie să comunice.

Acest lucru este prevăzut în caietul de sarcini ISO / IEC 7816-4 care definește o modalitate de a selecta aplicații, centrate în jurul unui ID a unei aplicație (AID). Un AID constă secvențe de până la 16 bytes.

Dacă se efectuează emularea unui card pentru o infrastructură existentă de cititor NFC, AID ale cititorilor care sunt căutați pentru identificare sunt cunoscuți și a căror înregistrare este publică (de exemplu, AID pentru rețelele de plată, cum ar fi Visa și MasterCard).

Dacă se dorește implementarea unei noi infrastructuri de cititor pentru o aplicație proprie, va trebui să se înregistreze propriul AID (e). Procedura de înregistrare pentru AIDs este definită în caietul de sarcini ISO / IEC 7816-5. Google recomandă înregistrarea unui AID ca pe 7816-5 dacă implementarea unei cereri de HCE este pentru un dispozitiv Android, deoarece, astfel se vor evita coliziunile cu alte aplicații.

2.4.2 Grupurile AID

În unele cazuri, un serviciu de HCE ar putea avea nevoie să se înregistreze cu mai multe AIDs pentru a pune în aplicare o anumită aplicație, și trebuie să se asigure că aceasta este gestionată implicit pentru toate aceste AID (spre deosebire de unele AIDs în care grupul se deplasează către alt serviciu).

Un grup de AID este o listă care ar trebui să fie considerată ca aparținând sistemului de operare. Pentru toate AIDs dintr-un grup, dispozitivul Android garantează una dintre următoarele:

- toate AIDs din grup sunt dirijate către serviciul HCE;
- nu toate AIDs din grupul sunt dirijate la acest serviciu HCE (de exemplu, dacă utilizatorul preferă un alt serviciu care a solicitat unul sau mai multe AIDs din respectivul grup).

2.4.3 *Grupuri și categorii de AIDs*

Fiecare grup AID poate fi asociat cu o categorie. Acest lucru permite dispozitivelor Android accesul la serviciile HCE împreună cu un grup sau pe categorii, și care, la rândul său permite utilizatorului pentru a seta valorile implicite, la nivelul categoriei aferentă nivelului AID. În general, se evită menționarea AIDs în orice user-facing cu care se confruntă utilizatorii aplicației: ei nu reprezintă nimic pentru utilizatorul mediului respectiv.

Android 4.4 suportă două categorii: `CATEGORY_PAYMENT` (acoperă aplicațiile de plată Industry-standard) și `CATEGORY_OTHER` (pentru toate celelalte aplicații HCE).

Notă: Doar un singur grup din categoria `CATEGORY_PAYMENT` poate fi activat în sistemul la un moment dat. De obicei, aceasta va fi o aplicație care înțelege majoritatea acestor protocoale, de plată cu cardul de credit și care pot lucra cu orice comerciant.

Pentru aplicații de plată de tip *buclă închisă* care funcționează numai la un comerciant (cum ar fi cărțile de valoare stocată), ar trebui să se utilizeze `CATEGORY_OTHER.` Grupurile AIDs din această categorie pot fi întotdeauna active, și pot să acorde prioritate cititorilor NFC în timpul selecției dacă este necesar.

2.5 Implementarea unui serviciu de HCE

Pentru a emula un card NFC folosind *host-based card emulation*, trebuie să fie creată o componentă `Service` care se ocupă de tranzacțiile NFC.

2.5.1 *Verificarea pentru sprijin HCE*

Solicitarea unui utilizator poate verifica dacă un dispozitiv acceptă HCE prin verificarea caracteristică `FEATURE_NFC_HOST_CARD_EMULATION`. Pentru acest lucru trebuie să fie utilizată `<uses-feature>` tag în manifestul solicitării care declare că aplicația utilizează funcția HCE, și dacă este necesar pentru ca aplicația să funcționeze sau nu.

2.5.2 *Implementarea de servicii*

Android 4.4 vine cu un set de `Service` de clasă, care poate fi folosit ca bază pentru punerea în aplicare a unui serviciu HCE: a clasei `HostApduService`.

Primul pas este, prin urmare, este să se extindă `HostApduService.`

```
public class MyHostApduService extends HostApduService {
  @Override
  public byte [] processCommandApdu ( byte [] apdu , Bundle extras ) {
    ...
  }
  @Override
  public void onDeactivated ( int reason ) {
    ...
  }
}
```

`HostApduService` declară două metode abstracte care trebuie să fie înlocuite și puse în aplicare.

`processCommandApdu ()` este numit de fiecare dată când un cititor NFC trimite o unitate de aplicare Protocol de date (APDU). APDUs sunt definite în caietul de sarcini ISO / IEC 7816-4. APDUs sunt pachetele la nivel de aplicație care sunt schimbate între cititorul NFC și serviciul de

HCE. Acest protocol la nivel de aplicație este de tipsemi-duplex: cititorul NFC va trimite un APDU comandă, și va aștepta în schimb un răspuns APDU.

Notă: ISO / IEC 7816-4 caietul de sarcini definește, de asemenea, conceptul de mai multe canale logice, în cazul în care pot avea loc mai multe schimburi APDU paralele pe canale logice separate.

Punerea în aplicare a HCE Android, suportă doar un singur canal logic, deci nu e doar un singur fir de schimb de APDUs.

Așa cum s-a menționat anterior, Android folosește AID pentru a determina care serviciu HCE poate fi accesat de cititor. De obicei, primul APDU al unui cititor NFC trimite un "SELECT AID" APDU; acest APDU conține AIDs de care cititorul are nevoie. Android extrage APDU, care trebui rezolvat de un serviciu HCE, mai înainte de APDU.

Se poate trimite un răspuns APDU de returnare a bytes de APDU, răspuns de la `processCommandApdu` (). Această metodă va fi transmisă în meniul principal al aplicației, pentru a nu fi blocată. Prin urmare, dacă un răspuns APDU nu revine imediat, el întoarce null. Aceasta permite lucrul pe un alt canal și folosirea metodei `sendResponseApdu ()` definită în clasa `HostApduService` pentru a putea trimite răspunsul.

Android va păstra noile APDUs de la cititor, fără a le expedia până când:

1. Cititorul NFC trimite un alt APDU "SELECT AID", pe care sistemul de operare îl rezolvă la un serviciu diferit;
2. Link-ul NFC dintre cititor NFC și aparatul este corupt.

În ambele cazuri, clasa de punerea în aplicare a `onDeactivated ()` este apelată cu un argument care să indice care dintre cele două sa întâmplat.

Dacă se lucrează cu infrastructura existentă la cititor, atunci este nevoie de a pune în aplicare protocolul la nivel de aplicație existentă pe care cititorii îl așteaptă în serviciul de HCE.

Dacă avem de-a face cu implementarea unei noi infrastructuri la cititor care poate controla la fel de bine, se va putea defini propriul protocol și secvența APDU. În general, se încearcă să se limiteze cantitatea de APDUs și dimensiunea datelor care trebuie să fie schimbate: acest lucru face mai sigur că dispozitivul utilizatorilor care vor trebui doar să dețină lor asupra cititorul NFC pentru o scurtă perioadă de timp. O limită superioară este de aproximativ 1KB de date, care pot fi, de obicei, schimbate cu viteza 300ms.

2.5.3 Declararea serviciilor manifest și înregistrarea AID

Serviciul trebuie să fie declarate în manifest, ca de obicei, dar unele piese suplimentare trebuie să fie adăugate la declarația de serviciu.

În primul rând, pentru a spune că platforma este un serviciu HCE, este necesară implementarea unei interfețe `HostApduService`, iar noile servicii declarate trebuie să conțină un filtru intenție pentru acțiunile `SERVICE_INTERFACE`.

În plus, platforma care este solicită grupurile SIDA de către acest serviciu, o `SERVICE_META_DATA<meta-date>` tag trebuie să fie inclusă în declarația de serviciu, arătând spre o resursă XML cu informații suplimentare despre serviciul HCE.

În cele din urmă, trebuie setat `Android:` atributul `trebui exportat` ca adevărat, și necesită `"android.permission.BIND_NFC_SERVICE"` în declarația de servicii. Serviciul poate fi legat către aplicații externe. Acestea din urmă impun apoi că numai aplicațiile externe care dețin `"android.permission.BIND_NFC_SERVICE"` au permisiunea de a se folosi de serviciul respectiv. Din moment ce `"android.permission.BIND_NFC_SERVICE"` este un sistem de permisiune, acesta impune în mod eficient sistemului de operare al dispozitivului Android legătura la serviciul manifest.

Un exemplu de declarație manifest `HostApduService`:

```
<service android:name = ".MyHostApduService" android:exported = "true"
```

```
        android:permission = "android.permission.BIND_NFC_SERVICE" >
    <intent-filter>
        <action  android:name = "android.nfc.cardemulation.action.HOST_APDU_SERVICE" />
    </intent-filter>
    <meta-data  android:name = "android.nfc.cardemulation.host_apdu_service"
            android:resource = "@xml/apduservice" />
</service>
```

Această etichetă meta-date vă trimite la un fișier `apduservice.xml`. Un exemplu de astfel de dosar cu o declarație a unui singur grup de ajutor care conține două dintre proprietățile SIDA este prezentată mai jos:

```
<host-apdu-service  xmlns:android = http://schemas.android.com/apk/res/android
        android:description = "@string/servicedesc"
        android:requireDeviceUnlock = "false" >
    <aid-group  android:description = "@string/aiddescription"
            android:category = "other" >
        <aid-filter  android:name = "F0010203040506" />
        <aid-filter  android:name = "F0394148148100" />
    </aid-group>
</host-apdu-service>
```

`<Host-APDU-serviciu>` tag este necesar să conțină un `<Android: description>` atribut care conține o descriere user-friendly a serviciului care pot fi afișate în UI. `RequireDeviceUnlock` atribut poate fi folosit pentru a specifica faptul că dispozitivul trebuie să fie deblocat înainte de acest serviciu poate fi invocat pentru a gestiona APDUs.

`<Host-APDU-serviciu>` trebuie sa contina una sau mai multe `<ajutor de grup>` tag-uri. Fiecare `<ajutor grup>` tag este necesar pentru a:

- conține un atribut `Android: descriere`, care conține o descriere user-friendly a grupului AID, potrivit pentru afișare în UI;
- de asemenea `Android: categoria` este un atribut setat pentru a indica categoria grupului AID de care aparține, de exemplu, constantele șir definite de CATEGORY_PAYMENT sau CATEGORY_OTHER.;
- fiecare `<ajutor grup>` trebuie sa conțină una sau mai multe `<ajutor de filtrare>` tag-uri, fiecare dintre ele conțin un ajutor unic. Ajutorul trebuie să fie specificate în format hexazecimal, și conțin un număr par de caractere.

Ca o notă finală, cererea dumneavoastră trebuie de asemenea să dețină `NFC` permisiunea pentru a putea să se înregistreze ca un serviciu HCE.

2.6 AID pentru rezolvarea conflictelor

Mai multe `HostApduService` componente pot fi instalate pe un singur dispozitiv, și același ajutor poate fi înregistrat de mai mult de un serviciu. Platforma Android soluționează conflictele AJUTOR în funcție de categoria din care face parte un ajutor. Fiecare categorie poate avea o politică diferită de soluționare a conflictelor.

De exemplu, pentru unele categorii, cum ar fi plata () utilizatorul poate fi capabil de a selecta un serviciu implicit în setările Android UI. Pentru alte categorii, se poate cere întotdeauna ca utilizatorul care urmează să fie invocat în caz de conflict. Pentru a interoga politica de soluționare a conflictelor pentru o anumită categorie, a se vedea `getSelectionModeForCategory` ().

2.6.1 Verificarea dacă serviciul este implicit

Cererile pot verifica dacă serviciul lor HCE este serviciul implicit pentru o anumită categorie utilizând `isDefaultServiceForCategory (ComponentName, String)` API.

Dacă serviciul nu este implicit, se poate solicita să fie implicit. Vezi `ACTION_CHANGE_DEFAULT.`

2.7 Cererile de plată

Android consideră serviciile HCE care au declarat un grup AIDs cu categoria "plată" ca cereri de plată.

Versiunea Android 4.4 conține Setări de nivel superior de intrare în meniu numite generic "de la robinet și plată", care enumeră toate aceste cereri de plată. În aceste setări din meniu, utilizatorul poate selecta aplicația de plată implicită, care va fi invocată atunci când un terminal de plată este utilizat.

2.7.1 Activele necesare pentru cererile de plată

Pentru a oferi o experiență de utilizare mai atractivă punct de vedere vizual, cererile de plată HCE sunt obligate să furnizeze un avantaj suplimentar pentru serviciul lor, așa-zisul banner service.

Acest activ ar trebui să fie dimensionat 260x96 PD, și poate fi specificate în fișierul XML meta-date prin adăugarea `Android: apduServiceBanner` atributul la `<host-APDU de serviciu>` tag-ul, ceea ce indică resursa drawable.

Un exemplu este arătat mai jos:

```
<host-apdu-service  xmlns:android = http://schemas.android.com/apk/res/android
      android:description = "@string/servicedesc"
     android:requireDeviceUnlock = "false"
      android:apduServiceBanner = "@drawable/my_banner" >
   <aid-group  android:description = "@string/aiddescription"
         android:category = "payment" >
      <aid-filter  android:name = "F0010203040506" />
      <aid-filter  android:name = "F0394148148100" />
   </aid-group>
</host-apdu-service>
```

2.8 Screen Off and Lock-screen Behavior

Implementările actuale ale dispozitivului Android transformă complet controlerul NFC și procesorul de aplicare atunci când ecranul dispozitivului este oprit. Prin urmare, serviciile de HCE nu vor funcționa atunci când ecranul este oprit.

Serviciile HCE pot funcționa cu toate acestea și cu ecranul blocat: însă acest lucru este controlată de `Android: requireDeviceUnlock` atribut în `<gazdă-APDU-service>` etichetă a serviciului HCE. În mod implicit, dispozitivul de deblocare nu este necesar, iar serviciul va fi invocat chiar dacă dispozitivul este blocat.

Dacă este setat `Android: requireDeviceUnlock` atunci acesta capătă atributul de „*true*” pentru serviciul utilizatorului.

HCE Android va cere utilizatorului pentru a debloca dispozitivul atunci când atinge un cititor NFC, care selectează un AID care să poată rezolva acest aspect.

După deblocare, Android va afișa un dialog determinat de utilizator pentru a-l putea atinge din nou astfel încât să se poată finaliza tranzacția.

Acest lucru este necesar pentru a debloca, deoarece utilizatorul poate a efectuat o modificare de poziție (fizic) mai departe de dispozitivul cititorului NFC.

2.9 Coexistența cu elementele de securitate ale cardurilor

Această secțiune este de interes pentru dezvoltatorii care au desfășurat o aplicație care se bazează pe un element sigur pentru emularea cartelei. Punerea în aplicare a HCE Android, care este proiectată să funcționeze în paralel cu alte metode de implementare pentru emularea cartelei, inclusiv utilizarea unor elemente de securitate.

Notă: Dispozitivul Android nu oferă API-uri pentru comunicarea directă cu un element securizat în sine.

Această coexistență se bazează pe un principiu numit „*AID routing*”: rcontrolerul NFC păstrează o tabelă de rutare care constă dintr-o listă (finită) a regulilor de rutare. Fiecare regulă de rutare conține un AID și o destinație. Destinația poate fi CPU gazdă (în cazul în care aplicațiile Android rulează), sau un element securizat conectat.

În cazul în care cititorul NFC trimite un APDU cu o „*SELECT ajutor*”, controlerul NFC o verifică și analizează dacă AIDs se potrivește cu orice AID din tabelul de rutare. Dacă se potrivește, APDU și toate celelalte APDUs vor fi trimise la destinația asociată cu ajutorul, până când un alt „*SELECT ajutor*” APDU este primit sau pe link-ul NFC dacă este corupt.

Notă: În timp ce ISO / IEC 7816-4 definește conceptul de „*partial matches*”, lucru care de asemenea nu este acceptat de dispozitivele Android HCE.

Această arhitectură este ilustrată în figura 4.

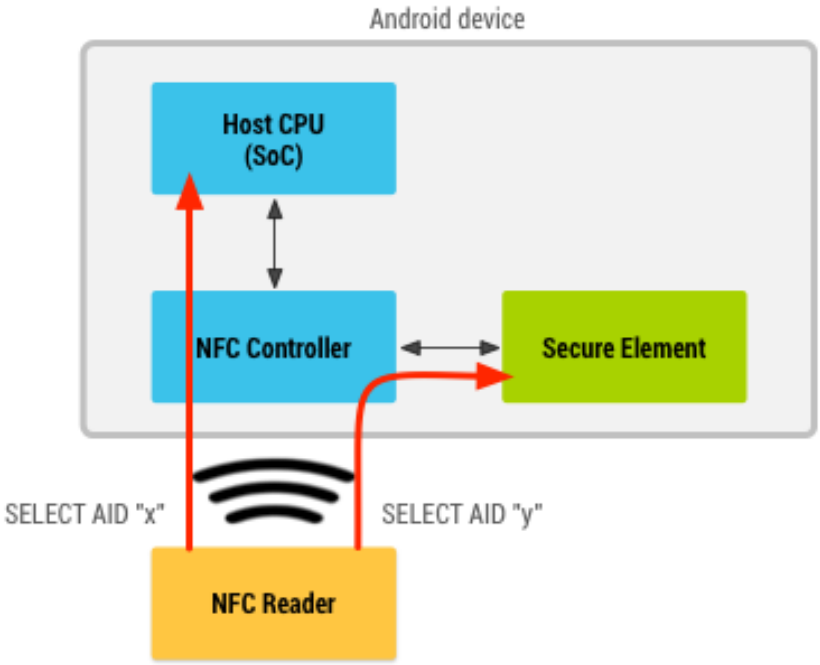

Figura 4. Android funcționează atât cu elemente de siguranță cât și cu host-card emulation.

Controlerul NFC conține de obicei o rută implicită pentru APDUs. Atunci când un AID nu se găsește în tabelul de rutare, se utilizează ruta default. Începând cu Android 4.4, este necesar ca ruta default să fie stabilită de la început în CPU gazdă. Acest lucru înseamnă că tabela de rutare conține de obicei doar intrări pentru AIDs, care au nevoie de a merge la un element securizat.

Aplicațiile Android care implementează un serviciu de HCE sau care folosesc un element de siguranță nu trebuie să își facă griji pentru configurarea tabelului de rutare - care se ocupă de dispozitivul Android în mod automat. Dispozitivul Android trebuie doar să știe care AID pot fi manevrate de serviciile HCE și care pot fi manipulate de către elementul securizat. Bazat pe serviciile

instalate și configurate de către utilizator, ca preferate, tabela de rutare este configurată astfel în mod automat.

2.9.1 *Înregistrare AJUTOR element de Secure*

Aplicațiile care folosesc un element securizat pentru „*host-card emulation*” pot declara un așa-numit „*service off gazdă*” în manifest. Declarația unui astfel de serviciu este aproape identică cu declarația unui serviciu HCE. Excepțiile sunt:

- acțiunea folosită în filtru-intenție trebuie să fie setat la `SERVICE_INTERFACE;`
- numele atributului meta-date trebuie să fie setat la `SERVICE_META_DATA;`
- fișierul XML meta-date trebuie să utilizeze tag-ul rădăcină `<offhost-APDU-service>`.

```
<service  android:name = ".MyOffHostApduService"  android:exported = "true"
       android:permission = "android.permission.BIND_NFC_SERVICE" >
   <intent-filter>
      <action  android:name = "android.nfc.cardemulation.action.OFF_HOST_APDU_SERVICE" />
   </intent-filter>
   <meta-data  android:name = "android.nfc.cardemulation.off_host_apdu_ervice"
          android:resource = "@xml/apduservice" />
</service>
```

Un exemplu de fișier corespunzător `apduservice.xml` care înregistrează două AIDs:

```
<offhost-apdu-service  xmlns:android = http://schemas.android.com/apk/res/android
        android:description = "@string/servicedesc" >
   <aid-group  android:description = "@string/subscription"  android:category = "other" >
      <aid-filter  android:name = "F0010203040506" />
      <aid-filter  android:name = "F0394148148100" />
   </aid-group>
</offhost-apdu-service>
```

`Android: requireDeviceUnlock` este un atribut care nu se aplică serviciilor „*service off gazdă*”, deoarece procesorul gazdă nu este implicat în tranzacție și, prin urmare, nu poate împiedica elementul securizat să execute tranzacțiile atunci când aparatul este blocat.

`Android: apduServiceBanner` este atributul care trebuie să fie utilizat pentru servicii „*service off gazdă*”, cum ar fi cererile de plată, precum și în scopul de a fi selectate ca o cerere de plată implicită.

2.9.2 *Off host service invocation*

Android nu va începe sau nu se va lega de un serviciu care este declarat „*service off gazdă*”. Acest lucru se datorează faptului că tranzacțiile efective sunt executate de către elementul de siguranță și nu de serviciul Android în sine. Declarația de serviciu permite numai aplicațiilor să înregistreze AIDs prezente pe elementul securizat.

2.10 HCE and Security

Arhitectura HCE este văzută ca un element de bază al elementului de siguranță: deoarece serviciul este protejat de sistemul de permisiune `BIND_NFC_SERVICE`, numai sistemul de operare se poate loga la el și de a comunica cu serviciul. Acest lucru asigură că orice APDUs primit este de fapt un APDU care a fost primit de către sistemul de operare de la operatorul NFC și orice APDU trimis înapoi va merge doar la sistemul de operare, care la rândul său transmite APDUs direct la controlerul NFC.

Restul de piese de bază, în cazul în care se obțin datele pe care aplicația le trimite la cititor NFC, au grijă ca decuplarea să se efectueze în mod intenționat încă din faza de proiectare a HCE: deoarece sistemul nu trebuie să fie interesat de unde vin datele, el trebuie doar să se asigure că acesta este transportat în condiții de siguranță la controlerul și la cititorul NFC.

Pentru stocarea în siguranță și recuperarea datelor care trebuie trimise de la serviciul HCE, se va utiliza aplicația Sandbox Android, care izolează datele aplicației de alte aplicații.

2.11 Parametrii de protocol și detalii

Această secțiune este de interes pentru dezvoltatorii care doresc să înțeleagă ce parametrii de protocol HCE folosesc dispozitivele în timpul anti-coliziune și de activare a etapelor aferente protocoalelor NFC. Acest lucru permite construirea unei infrastructuri pentru cititor care este compatibilă cu dispozitivele Android HCE.

2.11.1 NFC-A (ISO / IEC 14443 tip A) activarea protocolului anti-coliziune

Ca parte a protocolului de activare a NFC-A, mai multe cadre sunt schimbate.

În prima parte a schimbului dispozitivul HCE va prezenta acesteia UID; dispozitivele HCE ar trebui să aibă un UID aleatoriu. Acest lucru înseamnă că, pe fiecare traseu, UID care este prezentat cititorului ca un UID generat aleatoriu. Din acest motiv, cititorii NFC nu ar trebui să depindă de UID al dispozitivelor HCE, ca o formă de autentificare sau de identificare.

Cititorul NFC poate selecta ulterior dispozitivul HCE prin trimiterea unei comenzi SEL_REQ. Răspunsul SEL_RES dispozitivului HCE va avea cel puțin al 6-lea bit (0x20) set, indicând faptul că dispozitivul acceptă ISO-DEP. Alți biți în SEL_RES pot fi stabiliți, de asemenea, indicând de exemplu, suportul pentru protocolul NFC-DEP (p2p). Deoarece pot fi stabiliți alți biți, cititorii care doresc să interacționeze cu dispozitivele HCE ar trebui să verifice în mod explicit cei 6 biți, și nu să compare complet SEL_RES cu o valoare de 0x20.

2.11.2 Activarea ISO-DEP

După ce protocolul NFC-A este activat, activarea protocolului ISO-DEP este inițiată de către cititorul NFC. Acesta trimite o comandă "RATS" (solicitare de răspuns). Răspunsul, ATS, este complet generat de operatorul NFC și nu este configurabil de către serviciile HCE. Cu toate acestea, implementările HCE sunt necesare pentru a satisface cerințele NFC Forum pentru răspunsul ATS, astfel încât cititorii NFC să poată conta pe acești parametri ca fiind stabiliți în conformitate cu cerințele NFC Forum pentru orice dispozitiv HCE.

Secțiunea de mai jos oferă mai multe detalii cu privire la biții individuali ai răspunsului ATS furnizați de către operatorul NFC de pe un dispozitiv HCE:

- TL: lungimea răspunsului ATS. nu trebuie să indice o lungime mai mare de 20 bytes;
- T0: biți 5, 6 și 7 trebuie să fie fixată pe toate dispozitivele HCE, indicând TA (1), TB (1) și TC (1) sunt incluse în răspunsul ATS. Biți de la 1 la 4 indica FSCI, codificare dimensiunea maximă a cadrelor. Pe dispozitivele HCE valoarea FSCI trebuie să fie între 0h și 8h;
- T (A) 1: definește rate de transfer între cititor și emulator, precum și dacă acestea pot fi asimetrice. Nu există cerințe sau garanții pentru dispozitivele HCE;
- T (B) 1: 1 până la 4 biți se indică pornirea la un anumit interval de timp Guard Integer (SFGI). Pe dispozitivele HCE, SFGI trebuie să fie <= 8h. Biți 5 până la 8 indică Frame așteptare timp Integer (FWI) și codurile cadrului Timp de așteptare (FWT). Pe dispozitivele HCE, FWI trebuie să fie <= 8h;
- T (C) 1: 5 bit indică suport pentru "caracteristici Protocolul de avansate". Dispozitive HCE poate sau nu poate suporta "caracteristici Protocolul de avansate". Bit 2 indică un sprijin pentru DID. Dispozitive HCE poate sau nu poate suporta DID. Bit 1 indică un sprijin pentru NAD. Dispozitivele HCE nu trebuie să susțină DNA și a stabilit bit 1 la zero;

- Bytes istorice: dispozitive HCE poate reveni până la 15 bytes istorice. Cititorii NFC care doresc să interacționeze cu serviciile HCE ar trebui să facă nici o presupuneri despre conținutul octeți istorice sau prezența lor.

De reținut că multe dispozitive HCE sunt probabil realizate în conformitate cu cerințele de protocoale pe care rețelele de plată unite în EMVCo au specificat în caietul de sarcini „*Contactless Communication Protocol*”. În special:

- FSCI în T0 trebuie să fie între 2 ore și 8 ore;
- T (A) 1 trebuie să fie setat la 0x80, indicând doar 106 kbit / s este susținută, și rate de transfer asimetrice între cititor și emulator nu sunt acceptate;
- FWI în T (B) 1 trebuie să fie <= 7h.

2.11.3 Schimbul de date APDU

După cum s-a menționat mai devreme, implementarea HCE se sprijină doar pe un singur canal logic. Încercarea de a selecta aplicațiile pe diferite canale logice nu va funcționa pe un dispozitiv HCE.

3. SCRIEREA UNUI APPLET pentru CARDURI JAVA

Ce este un card inteligent?

Este greu de imaginat că ați făcut achiziții importante - și multe tipuri de minori - fără carduri de credit. Au devenit aproape omniprezente în economiile moderne. Dar, la fel de familiar ca aceste carduri de plastic au devenit, li se alătură ceva care are mult mai multă putere și flexibilitate: „*Smart Card*”[3]. O cartelă inteligentă este o cartelă de plastic care are aspectul unui card de credit - este de aproximativ aceeași dimensiune și formă - dar cu o diferență semnificativă: o cartelă inteligentă are un microprocesor încorporat.

Microprocesorul este capabil să facă lucruri asemănătoare computerului, cum ar fi rularea programelor, procesarea intrărilor și ieșirilor și stocarea datelor. Dar, spre deosebire de cele mai multe computere, o cartelă inteligentă nu include un afișaj sau o tastatură - de fapt, nu are alimentare cu energie. Deoarece are nevoie de o modalitate de a obține rezultate de afișare și de afișare, o cartelă inteligentă funcționează în tandem cu un dispozitiv de acceptare a cardurilor (CAD), adică un cititor de carduri sau un terminal. Ambele tipuri de dispozitive au sloturi în care este plasat un card pentru citire. Cititoarele de carduri sunt conectate la computere; terminalele sunt ele însele calculatoare.

Un microprocesor cu cartelă inteligentă poate fi programat și din acest motiv puteți utiliza cartele inteligente pentru a face lucruri pe care nu le puteți face cu cardurile de credit. De exemplu, aveți posibilitatea să utilizați o cartelă inteligentă pentru a „*plăti în avans*”. O aplicație care utilizează o cartelă inteligentă pentru a plăti taxele de pod. Aici achiziționați un card care are o valoare monetară inițială. De fiecare dată când treceți printr-o cabină de taxare, un scaner citește valoarea actuală a cardului (probabil că cardul este montat pe tabloul de bord al mașinii) și deduce taxa de la valoarea curentă a cardului. Când valoarea curentă a cardului scade la o valoare scăzută sau zero, cumpărați o altă persoană sau mergeți la persoanele potrivite și le obligați să adauge mai multă valoare cardului.

O cartelă inteligentă poate fi, de asemenea, utilizată pentru a menține înregistrări personale, de exemplu, dosarele medicale. De fiecare dată când vizitați un cabinet medical, spuneți cabinetul medicului, o clinică sau un spital, personalul administrativ vă actualizează cardul pentru a ține evidența dvs. curentă. Acest lucru poate fi deosebit de valoros în cazuri de urgență, atunci când este posibil să nu fiți capabil fizic să comunicați cu personalul medical.

Acestea sunt doar două dintre un număr aproape nelimitat de aplicații posibile pentru carduri inteligente. Nu numai că o carte poate fi programată pentru mai multe tipuri diferite de aplicații, dar mai multe aplicații pot fi încărcate pe o cartelă. Dacă aceste aplicații sunt codificate folosind tehnologia Java Card, codul și datele pentru o aplicație sunt protejate împotriva accesului (și posibilelor deteriorări) de către ceilalți. În terminologia Java aceste aplicații se spune că rulează în „*sandbox-ul*” propriu.

În plus, cardurile inteligente pot fi reprogramate. Aceasta este o convenienţă importantă pentru emitenții de carduri inteligente. De exemplu, imaginați-vă că o bancă a emis carduri inteligente către peste 500.000 de clienți. Dacă banca are nevoie să actualizeze un program pe aceste carduri, acesta face pur și simplu actualizarea disponibilă pentru descărcare pe un computer atașat la cititorul de carduri. Data viitoare când un client al băncii transmite cardul prin cititor, actualizarea este descărcată pe card; banca nu trebuie să remită niciun fel de carduri inteligente.

3.1 Despre Java Card Technology

Inițial, dezvoltarea aplicațiilor smartcard a fost, în esență, proprietară. Deși toate cardurile inteligente păreau, în general, la fel, fiecare software-ul smartcardului era specific proiectării microprocesorului încorporat. Aceasta înseamnă că, în cazul în care compania **A** fabrică plăci inteligente și compania **B** fabrică carduri inteligente, nu a existat o modalitate generică de codificare a aplicațiilor care ar putea funcționa pe carduri produse de ambii producători. Ca rezultat, dezvoltarea

[3] https://www.oracle.com/technetwork/java/embedded/javacard/documentation/intro-139322.html#dclmem

aplicațiilor smartcard a fost limitată la un grup relativ mic de dezvoltatori care au lucrat fie pentru producătorii de carduri inteligente, fie pentru emitenții cardurilor inteligente.

Tehnologia Java Card oferă o arhitectură pentru dezvoltarea de aplicații deschise pentru carduri inteligente, utilizând limbajul de programare Java.

Cu toate acestea, în ultimii ani, dezvoltarea de aplicații smartcard a evoluat astfel încât nu mai este proprietate. Acum, dezvoltatorii de carduri inteligente pot produce aplicații care pot rula pe carduri de la diferiți producători. Aplicațiile de la diferiți dezvoltatori pot fi difuzate pe aceeași cartelă inteligentă. Acest lucru a deschis dezvoltatorii de carduri inteligente pentru furnizorii independenți de aplicații.

Tehnologia Java Card oferă o arhitectură pentru dezvoltarea de aplicații deschise pentru carduri inteligente, utilizând limbajul de programare Java. Tehnologia poate fi, de asemenea, utilizată pentru a dezvolta aplicații pentru alte dispozitive care au memorie extrem de mică, cum ar fi cartelele SIM pentru telefoane fără fir. O cartelă SIM este o cartelă inteligentă cu un substrat plastic mult mai mic decât cardurile inteligente de pe cardurile de credit. De obicei, o cartelă SIM are mai multă memorie decât alte tipuri de carduri inteligente, cum ar fi cardurile bancare.

Tehnologia Java Card cuprinde un set de specificații pentru următoarele:

- ***interfață de programare a aplicațiilor (API)***: Specificația API identifică principalele biblioteci de clase Java Card;
- ***o mașină virtuală 1***: Specificația mașinii virtuale descrie caracteristicile mașinii virtuale pentru manipularea aplicațiilor Java Card;
- ***un mediu de rulare***: Specificația Java Card Runtime Environment (JCRE) descrie detaliile comportamentului runtime, cum ar fi modul în care este gestionată memoria sau modul în care este aplicată securitatea.

3.2 Istoricul Java Card

Java Card 1.0 a fost inițial propus de inginerii de la Schlumberger. Aceasta a constat într-o specificație pentru API-uri numai. Ulterior, alte companii, cum ar fi Bull și Gemplus, s-au alăturat echipei Schlumberger pentru a forma Forumul de carduri Java. Acesta este un consorțiu din industrie care lucrează pentru adoptarea tehnologiei Java Card în industria de carduri inteligente. Mai târziu, Sun Microsystems a lucrat cu Forumul Java Card și cu alți reprezentanți ai industriei de carduri inteligente pentru a dezvolta Java Card 2.0. Acest nivel al tehnologiei a inclus o specificație JCRE.
Java Card 2.1.1, cea mai recentă versiune a tehnologiei, include specificațiile API, mașină virtuală și JCRE. Este disponibil pentru descărcare.

De asemenea, disponibil pentru descărcare este kitul Java 2.1.1 Development Kit. Kitul de dezvoltare oferă un mediu pentru testarea applet-urilor pentru carduri Java și un instrument de conversie care pregătește applet-urile Java Card pentru descărcarea pe o cartelă inteligentă (sau pe un alt dispozitiv care implementează Java Card 2.1 sau o versiune ulterioară). Acesta include, de asemenea, exemple Java applet-uri.

3.2.1 Arhitectura Java Card Applet

Portofelul electronic este un exemplu de applet pentru Java Card care este ambalat în kitul de dezvoltare Java 2.1.1. Puteți găsi fișierul cu cod sursă, Wallet.java, în directorul de mostre.

Aplicația Wallet transformă un card inteligent într-o versiune electronică a unui portofel. Ca un portofel, o cartela inteligentă cu applet-ul Wallet poate deține bani, deși aici este o versiune digitală a banilor. Apletul poate adăuga bani sau poate scade bani din portofel. De asemenea, poate indica suma curentă în portofel. Desigur, un portofel trebuie protejat astfel încât numai proprietarul său (sau altcineva care este autorizat în mod similar) să poată ajunge la bani. Din această cauză, applet-ul Wallet include, de asemenea, un mecanism de securitate care impune fiecărui utilizator să introducă un număr personal de identificare (PIN). Numai utilizatorii care introduc codurile PIN autorizate pot direcționa cererile legate de bani către applet.

Declaring the Package

În același mod în care se pot grupa clase și interfețe Java asociate într-un pachet, se pot îmbina applet-uri Java Card aferente într-un pachet. Acest lucru se face specificând o instrucțiune de pachet de la începutul fișierului sursă pentru applet. Formatul instrucțiunii pachet și numele utilizat pentru pachet urmează convențiile limbajului Java. De fapt, tehnologia Java Card urmărește convențiile de limbaj Java pentru toți identificatorii. Următoarea declarație specifică Wallet ca membru al pachetului *com.sun.javacard.samples.wallet*.

Importing the Java Card Framework

Tehnologia Java Card definește un set de clase și interfețe pentru programarea aplicațiilor Java Card. Aceste clase și interfețe sunt furnizate în diferite pachete. Unul dintre pachetele, javacard.framework, definește clasele și interfețele care sunt esențiale pentru dezvoltarea applet-urilor Java Card, cum ar fi clasa Applet de bază. Următoarea instrucțiune importă pachetul *thejavacard.framework*: a se vedea clasele de carduri Java utilizate în Applet-ul pentru Wallet pentru o descriere a claselor din pachetul javacard.framework utilizate în applet-ul Wallet.

Extending the Base Applet Class

Pachetul javacard.framework definește clasa javacard.framework.Applet, care este clasa de bază pentru toate applet-urile Java Card. *javacard.framework.Applet* definește metodele pe care un applet de cartelă Java le utilizează pentru a comunica cu JCRE. Toate applet-urile Java Card se extind din clasa de bază, ca în cazul aplicației Apple Wallet:

Secțiunea de referință a acestui articol conține o listă a tuturor *javacard.framework.AppletMethods* utilizate în Applet-ul Wallet.

Declaring Constants

Apletul Wallet declară diferite constante. Unele constante sunt valori de un octet în antetul comenzii APDU (Unități de date protocol de aplicație). APDU-urile sunt pachete de date care sunt schimbate între CAD și o cartelă inteligentă. APDU-urile sunt mijloacele standard de comunicare pentru cartelele inteligente. Există două tipuri: comenzi APDU, care specifică o operație care trebuie efectuată de o cartelă inteligentă; și APDU de răspuns, care conțin răspunsul cardului inteligent (starea și, opțional, datele) la o cerere operațională.

What's an APDU?

Tehnologia Java Card este modelată după un standard de specificație a cartelei inteligente, ISO7816. Standardul specifică faptul că comunicarea dintre o aplicație gazdă și o cartelă inteligentă se realizează prin APDU (Application Protocol Data Units). Un APDU este un pachet de date care se conformează unui format specific. Există două tipuri de APDU: comenzi APDU și APDU-uri de răspuns.

O comandă APDU începe cu un antet și este opțional urmată de un corp. Antetul conține câmpuri care specifică o operație care trebuie efectuată de o cartelă inteligentă. Organismul include orice date care însoțesc cererea; De asemenea, indică numărul maxim de octeți de date așteptați ca răspuns la comandă.

Un răspuns APDU opțional începe cu un corp care conține toate datele returnate în răspunsul. Răspunsul APDU se termină cu doi octeți obligatorii care specifică starea de procesare a cardului. Consultați formatele APDU pentru o ilustrare a formatelor APDU de comandă și răspuns.

În tehnologia Java Card, aplicația gazdă trimite o comandă APDU, iar un atelier Java Card răspunde cu un răspuns APDU. (De fapt, un applet pentru cartea Java se află în așteptare până când primește o comandă APDU.) Cu toate acestea, comunicarea nu este direct gazdă aplicație-la-carte Java applet. În schimb, JCRE acționează ca intermediar. Comanda APDU este transmisă către JCRE, care o trimite la apletul corespunzător Java Card pentru procesare. După procesarea APDU, apletul Java Card transmite un răspuns APDU către JCRE, care îl trimite aplicației gazdă.

Dacă ne referim la formatele APDU, vom observa că primul octet al unei comenzi APDU este octetul CLA. CLA reprezintă o clasă de instrucțiuni. Valoarea octetului CLA identifică o categorie APDU. ISO7816 cere o valoare specifică pentru octetul CLA pentru o singură categorie de comenzi APDU - comanda SELECTAPDU. Valoarea octetului CLA pentru o comandă SELECT APDU trebuie să fie 0. Deși ISO7816 nu solicită o valoare specifică pentru alte categorii de comenzi APDU, identifică un set de cerințe pe care aceste valori trebuie să le îndeplinească. Deci, pentru categoriile de comenzi APDU, altele decât comanda SELECT APDU, există libertatea de a selecta o valoare pentru octetul CLA atâta timp cât valoarea este conformă cu specificația ISO. Pentru aplicația Wallet, valoarea hexazecimală 0xB0 este utilizată pentru a identifica categoria PROCESS a comenzilor APDU. Aceasta înseamnă că pentru operațiile de procesare cum ar fi creditul și debitul, octetul CLA din comenzile APDU aplicabile are o valoare hexazecimală 0xB0.

```
// code of CLA byte in the command APDU header
final static byte Wallet_CLA =(byte)0xB0;
```

Cel de-al doilea octet al unei comenzi APDU este octetul INS. Acest octet identifică o instrucțiune specifică, de exemplu, un tip specific de cerere de procesare. AveM libertatea de a selecta valorile pentru octetul INS. Pentru aplicația Apple Wallet, valorile hexazecimale 0x20, 0x30, 0x40 și 0x50 specifică instrucțiunile pentru verificarea PIN, creditul, debitul și pentru a obține soldul actual.

```
// codes of INS byte in the command APDU header
final static byte VERIFY = (byte) 0x20;
final static byte CREDIT = (byte) 0x30;
final static byte DEBIT = (byte) 0x40;
final static byte GET_BALANCE = (byte) 0x50;
```

Alte constante stabilesc limite ale procesării creditului și debitului efectuate de applet. ConstantaMAX_BALANCE stabilește un sold maxim pentru portofelul electronic de 32.767 USD (adică 0x7FFF hexazecimal), iar valoarea MAX_TRANSACTION_AMOUNT constantă stabilește o valoare maximă pentru tranzacțiile de credit și de debit de 127 USD. Există, de asemenea, constante care controlează numărul maxim de ori pe care un cod PIN nevalid poate fi introdus înainte ca intrarea PIN să fie blocată (3) și dimensiunea maximă a codului PIN (8).

```
// maximum balance
final static short MAX_BALANCE = 0x7FFF;
// maximum transaction amount
final static byte MAX_TRANSACTION_AMOUNT = 127;

// maximum number of incorrect tries before the
// PIN is blocked
final static byte PIN_TRY_LIMIT =(byte)0x03;
// maximum size PIN
final static byte MAX_PIN_SIZE =(byte)0x08;
```

Setul final de constante declarate în appletul Wallet sunt valori ale cuvântului de stare care sunt returnate de applet în anumite circumstanțe, cum ar fi atunci când verificarea PIN nu reușește. Ca parte a protocolului APDU, aceste constante sunt transmise de la applet la JCRE și de la aplicația gazdă.

```
// signal that the PIN verification failed
final static short SW_VERIFICATION_FAILED = 0x6300;
// signal the the PIN validation is required
// for a credit or a debit transaction
final static short SW_PIN_VERIFICATION_REQUIRED = 0x6301;
```

```
// signal invalid transaction amount
// amount > MAX_TRANSACTION_AMOUNT or amount < 0
final static short sw_invalid_transaction_amount = 0x6a83;

// signal that the balance exceed the maximum
final static short sw_exceed_maximum_balance = 0x6a84;
// signal the the balance becomes negative
final static short sw_negative_balance = 0x6a85;
```

Declaring Member Variables

Aplicația Wallet declară două variabile membre care sunt utilizate pentru a conține valori pentru anumite obiecte. Pinul variabil este un obiect OwnerPIN care deține valoarea PIN a utilizatorului. OwnerPIN este o clasă definită în pachetul javacard.framework. Clasa oferă metode pentru efectuarea operațiilor PIN, cum ar fi actualizarea sau verificarea unui cod PIN. Balanța variabilă indică suma curentă de bani din portofelul electronic.

```
/* instance variables declaration */
OwnerPIN pin;
short balance;
```

Specifying a Constructor

O instanță a unui applet Java Card este creată prin metoda de instalare a aplicației (*Installing the Applet*). Când rulează metoda de instalare, el invocă un constructor. Constructorul face trei lucruri:

- creează un obiect OwnerPIN;
- inițiază obiectul;
- înregistrează instanța de aplicație.

Constructorul este declarat privat. Acest lucru înseamnă că nici o altă clasă nu poate iniția applet-ul Wallet.

```
private Wallet (byte[] bArray,short bOffset,byte
                                bLength){

  // It is good programming practice to allocate
  // all the memory that an applet needs during
  // its lifetime inside the constructor
  pin = new OwnerPIN(PIN_TRY_LIMIT, MAX_PIN_SIZE);

  // The installation parameters contain the PIN
  // initialization value
  pin.update(bArray, bOffset, bLength);
  register();

  } // end of the constructor
```

În legătură cu alocarea memoriei, deși nu este o cerință, este recomandat ca toate obiectele create într-un applet de cartelă Java să fie inițiate de constructorul applet-ului. Acest lucru este necesar pentru a evita pierderea de memorie la timpul de execuție.

De asemenea, se observă că instanța OwnerPIN este specificată cu doi parametri:

- numărul maxim de încercări incorecte înainte de blocarea codului PIN (PIN_TRY_LIMIT);
- dimensiunea maximă a codului PIN (MAX_PIN_SIZE).

Constanta PIN_TRY_LIMIT este de 3 și constanta MAX_PIN_SIZE este 8. Metoda de actualizare este definită în clasa OwnerPIN. Acesta stabilește o valoare nouă pentru codul PIN și stabilește numărul maxim de încercări PIN la valoarea PIN_TRY_LIMIT.

Înainte de a putea executa o instanță a unui applet de cartelă Java, trebuie să fie înregistrată la JCRE. Constructorul Wallet face acest lucru prin apelarea unui registru, o metodă definită *injavacard.framework.Applet*.

Identifying Applets

Standardul ISO7816 specifică faptul că fiecare aplicație pentru cartelă inteligentă trebuie identificată în mod unic de un identificator de aplicație (AID). AID este o serie de octeți. Primii cinci octeți reprezintă un identificator al resurselor (RID); restul de octeți (care pot varia de la zero la unsprezece octeți) este o extensie de identificare proprietară (PIX). ISO atribuie AID-uri solicitanților, cum ar fi producătorii de carduri inteligente; solicitanții sunt liberi să-și aleagă propriile RID-uri.

În tehnologia Java Card, AID-urile sunt folosite pentru a identifica applet-urile Java Card, precum și pachetele de applet-uri Java Card. Când cineva introduce o cartelă inteligentă care implementează tehnologia Java Card într-un dispozitiv de acceptare a cardurilor, aplicația care rulează pe dispozitiv trimite o comandă pe card. Comanda identifică o operație care trebuie efectuată, precum și AID-ul aplicației pentru a efectua operația.

Installing the Applet

Clasa *javacard.framework.Applet* definește o metodă statică de instalare. Metoda de instalare este ca metoda principală într-un program Java. Este punctul principal de intrare în applet. JCRE numește această metodă pentru a crea o instanță a unui applet Java Card și de a le controla. Toate applet-urile Java Card trebuie să implementeze metoda de instalare. Ce face implementarea este până la designerul de aplicații. Totuși, cel puțin, implementarea ar trebui să apeleze constructorul unui applet pentru a crea și inițializa o instanță a applet-ului. Este, de asemenea, tipic pentru constructor să înregistreze instanța. Și acesta este cazul în applet-ul Wallet (consultați Specificarea unui constructor). Deoarece există o metodă de apel pentru a exista în constructorul applet-ului, fiecare instanță din Wallet se înregistrează cu JCRE atunci când instanța este inițializată.

```
public static void install(byte[] bArray,
          short bOffset, byte bLength){
 // create a Wallet applet instance
 new Wallet(bArray, bOffset, bLength);
} // end of install method
```

Se poate observa că metoda de instalare acceptă trei parametri.

- bArray este o matrice de octet de tip care conține parametri de instalare;
- bOffset este o variabilă de tip scurt care conține offsetul de pornire în matrice;
- bLength este o variabilă de byte de tip care conține lungimea, în octeți, a datelor parametrilor din matrice.

Parametrii de instalare includ, de obicei, valori de configurare a applet-ului, cum ar fi:

- dimensiunea fișierelor interne;
- valorile de inițializare a apletului;
- identificare inițială.

Dar de unde provin acești parametri de instalare? Răspunsul este că, de obicei, parametrii de instalare sunt încărcați pe cartela inteligentă atunci când este instalat un applet. Desigur, pentru ca un applet să proceseze corect parametrii de instalare, trebuie să se știe conținutul și formatul parametrilor. Sau mai mult, proiectantul applet-ului trebuie să înțeleagă conținutul și formatul parametrilor de instalare și factorul în logica de procesare a aplicației.

Selecting and Deselecting the Applet

După ce este inițializat, un applett așteaptă într-o stare suspendată până când JCRE îl selectează în mod specific. Procesul de selecție este declanșat când JCRE primește o comandă SELECT APDU din aplicația gazdă. În general, o aplicație gazdă comunică cu un applet de cartelă Java prin intermediul APDU-urilor. Atunci când JCRE primește un APDU, verifică dacă antetul APDU specifică o comandă SELECT APDU.

După cum este indicat în Formatele APDU, antetul unei comenzi SELECT APDU are o valoare standard. Dacă valoarea antetului indică faptul că aceasta este o comandă SELECT APDU, JCRE compară valoarea AID în câmpul de date cu o listă de AID-uri care sunt înregistrate pentru cartela inteligentă. Dacă JCRE găsește o potrivire, ea apelează metoda de selectare pentru acel applet. Aceasta este o metodă definită în *javacard.framework.Appletclass*. Metoda de selectare indică JCRE dacă applet-ul este gata să proceseze cererile, returnând valoarea „*true*”. Dacă aplicația nu este pregătită să accepte cereri de procesare, metoda de selectare returnează valoarea „*false*”. JCRE trimite apoi un APDU de răspuns la aplicația gazdă indicând dacă selecția a avut succes sau nu a reușit. Consultați formatele APDU, pentru formatul APDU-ului de răspuns trimis ca răspuns la o comandă SELECT APDU.

Criteriile pe care un aplet le utilizează pentru a determina dacă ar trebui să returneze „*true*” sau „*false*” depinde de designul aplicației. Pentru aplicația Wallet, metoda de selectare verifică dacă codul PIN este blocat (deoarece a fost atinsă limita de încercare pentru PIN). Dacă PIN-ul este blocat, metoda de selecție *returnsfalse*, în caz contrar se întoarce adevărat.

```
public boolean select() {

// The applet declines to be selected
// if the pin is blocked.
  if ( pin.getTriesRemaining() == 0 )
  return false;

  return true;

}// end of select method
```

Atunci când un alt applet este selectat, adică un applet cu alt AID, JCRE apelează metoda de deselectare a aplicației actuale. Ca metoda selectată, metoda de deselectare este definită în clasa javacard.framework.Applet. Metoda de deselectare face orice curățare necesară înainte ca JCRE să dea noul control de aplet selectat. În appletul Wallet, metoda de deselectare resetează un semnalizator pentru a indica faptul că codul PIN nu mai este validat.

```
public void deselect() {

  // reset the pin value
  pin.reset();

}
```

Processing Requests

După ce un applet este selectat și se întoarce la JCRE adevărat, toate cererile primite de la aplicația gazdă sunt trimise procesului de procesare al applet-ului. Metoda procesului este o altă metodă definită în *javacard.framework.Applet*.

```
public void process(APDU apdu) {
```

Putem constata că metoda procesului acceptă un parametru, un obiect APDU. Obiectul APDU este o instanță a clasei javacard.framework.APDU. JCRE creează un obiect APDU ca modalitate de a comunica o comandă APDU cu un applet de cartelă Java și de a primi un APDU de răspuns din applet-ul Java Card. Este important să reținem că trimiterile la un obiect APDU sunt permise numai în cadrul unei metode, adică ca parametru al metodei sau stocate într-o variabilă locală. Aceasta este pentru a proteja împotriva posibilității ca un applet de la Java Card să acceseze datele APDU care aparțin unui alt applet Java Card.

În apletul Wallet, metoda procesului face următoarele:

- obține o referință la tamponul APDU;
- examinează antetul APDU;
- apelează metoda potrivită pentru a procesa cererea.

Gets a reference to the APDU buffer

După cum se menționează în secțiunea Selectarea și deselectarea appletului, o aplicație gazdă comunică cu un applet de cartelă Java prin APDU-uri.

Când JCRE primește o comandă APDU, scrie antetul APDU într-o matrice internă de byte numită tampon APDU. Antetul APDU cuprinde primii cinci octeți ai APDU, adică octeții CLA și INS, plus alți trei octeți, P1, P2 și Lc descriși în formatele APDU.

Tamponul APDU este încapsulat în obiectul APDU pe care JCRE îl trece în metoda theprocess. Pentru a procesa un APDU, un applet trebuie să primească mai întâi un pointer la tamponul APDU. Ea face acest lucru folosind getBuffer, o metodă definită în clasa APDU:

```
byte buffer[] = apdu.getBuffer();
```

Examines the Header

Metoda procesului examinează primii doi octeți ai antetului APDU, octetul CLA și octetul INS. Dacă valoarea octetului CLA este 0 și valoarea bytesului INS este 0xA4, aceasta indică faptul că acesta este antetul unei comenzi SELECT APDU. În acest caz, metoda de proces readuce controlul la JCRE:

```
// check SELECT APDU command
if ((buffer[ISO7816.OFFSET_CLA] == 0) &&
  (buffer[ISO7816.OFFSET_INS] == (byte)
                    (0xA4)) )
 return;
```

Se observă că utilizarea constantelor ISO7816.OFFSET_CLA și ISO7816.OFFSET_INS, prin Interfața ISO7816 din Java Card API definește constante diferite pentru a fi utilizate de applet-urile Java Card. O serie de aceste constante sunt destinate pentru a fi utilizate ca indici în antetul APDU; aceste constante încep cu OFFSET. Constantele ISO7816.OFFSET_CLA și ISO7816.OFFSET_INS sunt indicele pentru octeții CLA și INS, respectiv.

Metoda procesului verifică apoi octetul CLA pentru a vedea dacă are valoarea Wallet_CLA. Rețineți că în aplicația Wallet, valoarea hexazecimală 0xB0 este utilizată pentru a identifica categoria PROCESS a comenzilor APDU și că Wallet_CLA constant este valoarea 0xB0. Dacă octetul CLA nu are valoarea Wallet_CLA, cererea nu poate fi procesată de applet-ul Wallet și, astfel, metoda theprocess aruncă o excepție cu o constantă care este folosită în cuvântul de stare al răspunsului APDU:

```
// verify the reset of commands have the
// correct CLA byte, which specifies the
// command structure
if (buffer[ISO7816.OFFSET_CLA] != Wallet_CLA)
  ISOException.throwIt
(ISO7816.SW_CLA_NOT_SUPPORTED);
```

Constanta ISO7816.SW_CLA_NOT_SUPPORTED este definită în interfața IS07816 a API-ului Java Card. Constatările interfeței ISO7816 care încep cu SW sunt folosite în cuvântul de stare al unui răspuns APDU.

Calls the appropriate method to process the request

Dacă octetul CLA al antetului APDU indică o categorie PROCESS a comenzii APDU, metoda theprocess verifică octetul INS pentru a determina tipul de solicitare a categoriei PROCESS. Apoi, apelează metoda potrivită din Wallet, după cum urmează, pentru a gestiona solicitarea:

Method Purpose
credit (apdu)Credit an amount to the current balance
debit (apdu) Debit an amount from the current balance
getBalance (apdu) Get the current balance
verify (apdu)Validate the PIN
If the INS byte does not have a value that is recognized by the Wallet applet, the process method throws an exception with a constant that is returned in the status word of the response APDU.

```
 switch (buffer[ISO7816.OFFSET_INS]) {
   case GET_BALANCE:   getBalance(apdu);
                       return;
   case DEBIT:         debit(apdu);
                       return;
   case CREDIT:        credit(apdu);
                       return;
   case VERIFY:        verify(apdu);
                       return;
   default:       ISOException.throwIt
          (ISO7816.SW_INS_NOT_SUPPORTED);
 }

}  // end of process method
```

Credit an Amount to the Current Balance

Metoda de creditare din aplicația Wallet creditează o sumă la soldul curent. Metoda se numește cu un parametru: un obiect APDU. Obiectul APDU încapsulează un tampon APDU care conține comanda APDU CREDIT. Suma care trebuie creditată la soldul curent se află în câmpul de date al APDU.

```
private void credit(APDU apdu) {
```

Metoda începe prin verificarea codului PIN pentru validare. Amintiți-vă că pinul variabil este un obiect OwnerPIN. Metoda este Validated este definită în OwnerPIN; metoda returnează o valoare adevărată dacă PIN-ul este validat în prezent, adică un PIN valid a fost prezentat de la ultima resetare a cartelei sau de la ultimul apel la metoda de resetare. Dacă codul PIN nu este validat, adică! *pin.isValidated () returns true*, metoda de credit aruncă o excepție cu o constantă care este returnată în cuvântul de stare al răspunsului APDU:

```
// access authentication
if ( ! pin.isValidated() )
ISOException.throwIt(
    SW_PIN_VERIFICATION_REQUIRED);
```

Metoda de creditare primește apoi o referință la tamponul APDU (pentru mai multe detalii, consultați obținerea unei referințe la tamponul APDU):

```
byte buffer[] = apdu.getBuffer();
```

Tamponul APDU conține inițial antetul comenzii CREDIT APDU. După cum este ilustrat în formatele APDU, octetul Lc al antetului APDU indică numărul de octeți din câmpul de date. Metoda de creditare utilizează constanta interfeței IS07816 OFFSET_LC pentru a accesa octetul Lc în bufferul APDU:

```
// Lc byte denotes the number of bytes in the
// data field of the command APDU
byte numBytes = buffer[ISO7816.OFFSET_LC];

// indicate that this APDU has incoming data
// and receive data starting from the offset
// ISO7816.OFFSET_CDATA following the 5 header
// bytes.
```

În acest moment, metoda de creditare este gata să primească datele primite, adică suma care urmează să fie creditată la soldul curent și plasată în tamponul APDU. Aceasta face acest lucru prin callingsetIncomingAndReceive, o metodă definită în obiectul APDU. setIncomingAndReceivegets la fel de mulți octeți de date care se vor potrivi în bufferul APDU și returnează numărul de octeți pe care îi citește. Pentru aplicația Wallet, setIncomingAndReceive ar trebui să citească un octet de date.

Asta pentru că octetul Lc, care indică numărul de octeți din câmpul de date, are valoarea 1. Metoda de credit verifică acest lucru. Dacă numărul de octeți citit nu este 1, metoda aruncă o excepție cu o constantă care este returnată în cuvântul de stare al răspunsului APDU:

```
byte byteRead =
       (byte)(apdu.setIncomingAndReceive());

// it is an error if the number of data bytes
// read does not match the number in Lc byte
if ( ( numBytes != 1 ) || (byteRead != 1) )
 ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
```

Cu suma de credit disponibilă în tamponul APDU, creditul execută partea principală a procesării acestuia. Aceasta:

- obține suma creditului din bufferul APDU;
- verifică faptul că suma creditului nu încalcă limita maximă a tranzacției specificată de constantaMAX_TRANSACTION_AMOUNT sau este mai mică de 0;
- verifică dacă noul sold (cu suma creditului adăugat) nu depășește limita maximă a soldului specificată de valoarea constantă MAX_BALANCE;se adaugă suma creditului la soldul curent.

```
// get the credit amount
byte creditAmount =
          buffer[ISO7816.OFFSET_CDATA];

// check the credit amount
if ( ( creditAmount > MAX_TRANSACTION_AMOUNT)
    || ( creditAmount < 0 ) )
   isoexception.throwit
           (sw_invalid_transaction_amount);
```

```
// check the new balance
if ( (short)( balance + creditamount)
                        > MAX_BALANCE )
  ISOException.throwIt
            (SW_EXCEED_MAXIMUM_BALANCE);

// credit the amount
balance = (short)(balance + creditAmount);

} // end of deposit method
```

Observăm cum creditul utilizează o altă constanta de interfață ISO7816, OFFSET_CDATA ca un index al câmpului de date al tamponului APDU. Reținem, de asemenea, că metoda face o excepție cu o constantă dacă limita tranzacției sau limita de sold este depășită. Constanta este returnată în cuvântul de stare al răspunsului APDU.

După ce metoda de credit finalizează cu succes procesarea, acesta returnează controlul JCRE. JCRE trimite apoi un APDU de răspuns la aplicația gazdă care conține valoarea cuvântului de stare 0x9000; acest lucru indică faptul că comanda CREDIT APDU a fost procesată cu succes.

3.3 Clasificarea Applet

java.lang.Object

javacard.framework.Applet

public abstract class **Applet**

extends Object

Această clasă abstractă definește un applet bazat pe tehnologia Java Card.

Clasa Applet trebuie extinsă cu orice aplicație care urmează să fie încărcată, instalată și executată pe o placă inteligentă compatibilă cu tehnologia Java Card.

O platformă compatibilă Java Card poate suporta opțional protocolul APDU cu lungime extinsă definită în ISO7816-4. Subclasa de applet trebuie să implementeze interfața javacardx.apdu.ExtendedLength pentru a accesa această capacitate de protocol APDU cu lungime extinsă a obiectului javacard.framework.APDU.

Exemplu de utilizare a Applet:

```
public class MyApplet extends javacard.framework.Applet {
  static byte someByteArray[];

  public static void install(byte[] bArray, short bOffset, byte bLength) throws ISOException {
    // make all my allocations here, so I do not run
    // out of memory later
    MyApplet theApplet = new MyApplet();

    // check incoming parameter data
    byte iLen = bArray[bOffset]; // aid length
    bOffset = (short) (bOffset + iLen + 1);
    byte cLen = bArray[bOffset]; // info length
    bOffset = (short) (bOffset + cLen + 1);
    byte aLen = bArray[bOffset]; // applet data length
    // read first applet data byte
    byte bLen = bArray[(short) (bOffset + 1)];
```

```
        if (bLen != 0) {
            someByteArray = new byte[bLen];
            theApplet.register();
            return;
        } else
            ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);
    }

    public boolean select() {
        // selection initialization
        someByteArray[17] = 42; // set selection state
        return true;
    }

    public void process(APDU apdu) throws ISOException{
  byte[] buffer = apdu.getBuffer();
  // .. process the incoming data and reply
  if ( buffer[ISO7816.OFFSET_CLA] == (byte)0 ) {
    switch ( buffer[ISO7816.OFFSET_INS] ) {
      case ISO.INS_SELECT:
          ...
          // send response data to select command
          short Le =  apdu.setOutgoing();
          // assume data containing response bytes in replyData[] array.
          if ( Le < ..) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH);
          apdu.setOutgoingLength( (short)replyData.length );
          apdu.sendBytesLong(replyData, (short) 0, (short)replyData.length);
          break;
      case ...
      }
    }
  }
}
```