ATTIVITÀ PROGETTUALE
SISTEMI MOBILI L-M

Studente: Bacchilega Simone

Anno Accademico 2013/2014

INDICE

SCOPO DELL' ATTIVITÀ PROGETTUALE 3
Introduzione 3
Assunzioni 3
Testing 3
ELEMENTI DI BASE 4
Activities 4
SQLite Database 5
NEAR FIELD COMMUNICATION 6
Introduzione 6
Dettagli pratici 7
Lettura dei tag 7
Scrittura dei tag 11
WI-FI PEER-TO-PEER 14
Introduzione 14
Trasferimento dei dati 15
Elaborazione dei dati inviati al device Server 15
Pulizia della table 'Pazienti Cancellati' 16

SCOPO DELL’ ATTIVITÀ PROGETTUALE

Introduzione

Si vuole realizzare un’applicazione per dispositivi Android, utile in questo scenario: alcuni anestesisti vogliono registrare in modo rapido i dati dei pazienti giunti in ambulatorio e i relativi protocolli di trattamento. Inoltre, per tutta la durata della degenza del paziente, si verifica in modo saltuario la risposta dell’organismo ai farmaci somministrati (d’ora in avanti verranno indicate come “rilevazioni”). Per accelerare il processo di reperimento dei dati relativi al paziente per il quale si vuole effettuare la rilevazione, si associa a ciascuno di essi un TAG NFC, caricato all’atto della prima rilevazione. Poiché, in uno scenario realistico, non vi sarà un solo anestesista che registrerà pazienti ed effettuerà rilevazioni, è stato integrato un servizio di scambio e sincronizzazione di dati fra devices via wi-fi direct.

Assunzioni

Pensando al contesto in cui verrà utilizzata l’applicazione: vengono fatte le seguenti ipotesi:

- Per tutta la durata del trattamento, a ciascun paziente viene associato un codice alfanumerico che lo identifica univocamente. Al termine del trattamento, tale codice può essere utilizzato per un altro paziente, ma non nel breve termine (tipicamente non prima di un mese.)
- Dati relativi al paziente e protocollo vengono registrati contestualmente nello stesso momento, e il protocollo rimane il medesimo per tutto il trattamento. Qualora (in casi molto rari) sia necessario cambiare set di farmaci, viene cancellato il profilo corrente del paziente e registrato come nuovo paziente.
- In un determinato istante, un solo anestesista si occupa di un determinato paziente (quindi non accadrà mai per esempio, che due anestesisti nello stesso momento registrino lo stesso paziente. È comunque possibile che un anestesista lo registri, e un altro vada ad effettuare delle rilevazioni)
- Non è richiesta la perfetta sincronizzazione fra devices, ammettiamo l’errore fino a una manciata di minuti.
- Poiché la tecnologia NFC è stata introdotta da poco e legata a dispositivi di fascia medio-alta, l’utilizzo di NFC non è indispensabile per l’applicazione. Dev’essere possibile effettuare rilevazioni e consultare quelle precedenti anche su un dispositivo non dotato di hardware NFC.
- L’implementazione della comunicazione wi-fi direct è basata sull’idea che siano solo due i partecipanti alla comunicazione (due anestesisti si incontrano casualmente in corridoio e decidono di scambiarsi rapidamente i dati). La sincronizzazione fra *n* devices avviene quindi a 2 a 2.

Testing

Per testare su dispositivi reali l’app, sono stati utilizzati un Samsung Galaxy SII (GT-i9100) e un Nexus 4.

Per quanto riguarda i tag NFC, sono stati utilizzati alcuni adesivi con chiptype NTAG203 (basso costo, buona capacità, massima compatibilità)

ELEMENTI DI BASE

Activities

Per cominciare, una breve descrizione molto rapida delle Activities di maggiore rilievo.

Una activity lanciata all'apertura dell'app mostra sotto forma di tasti le tre azioni principali:

- Registrazione di un nuovo paziente (dati personali e protocollo di trattamento).

- Visualizzazione dei pazienti registrati, ordinati per piano. Tramite un breve clic sul paziente, verranno mostrate le rilevazioni già effettuate. Tramite clic lungo si aprirà un menu contestuale dal quale si potrà: cancellare il paziente, modificarne i dati principali (es. per trasferimento piano, cambio di letto, ecc.), visualizzazione del protocollo di trattamento, effettuazione di una nuova rilevazione.

- Scambio di dati via wi-fi direct con un altro dispositivo per sincronizzazione registrazioni e rilevazioni (approndimenti su questo aspetto più avanti).

SQLite Database

A livello realizzativo, si è deciso di sfruttare un database SQLite per salvare tutti i dati. Sono presenti le seguenti tabelle:

- Una tabella per i dati personali dei pazienti
- Una tabella diversa per ciascun protocollo
- Una tabella contenente tutte le rilevazioni
- Una tabella per salvare i codici dei pazienti cancellati (necessario per la sincronizzazione fra devices, in modo da propagare l’eliminazione dei trattamenti terminati, più dettagli nel capitolo su wi-fi direct)

Per ogni tabella è stata creata una classe Adapter che espone metodi CRUD. Comunicano con le Activities tramite oggetti DTO creati ad hoc (Pazienti, Protocolli, Rilevazioni), in modo tale che non siano le activities stesse a doversi preoccupare del trattamento dei Cursor restituiti dalle query e della conversione in ContentValues per l’inserimento/aggiornamento dei dati.

All’inserimento dei dati di un nuovo paziente, viene inserito, in maniera transizionale, anche il relativo protocollo di trattamento. Non vi sono invece particolari vincoli sull’inserimento di nuove rilevazioni (per esempio, dopo aver eseguito lo scan di un TAG e non aver trovato dati sul dispositivo relativi al paziente, l’anestesista potrebbe comunque procedere ad effettuare comunque la rilevazione associandola solo al codice a barre, e mandarla poi in seguito via wi-fi direct al dispositivo che possiede i dati personali del paziente).

Quando si chiude la pratica di un paziente, vengono eliminati contestualmente tutti i dati personali, il trattamento e le relative rilevazioni, e viene aggiunto il codice a barre in suo possesso alla tabella dei pazienti cancellati. In questo modo, in un secondo momento, scambiando dati via wi-fi direct, l’eliminazione delle vecchie entry verrà propagata anche agli altri dispositivi.

Passiamo ora ai veri aspetti cardine dell’attività progettuale: l’uso di NFC e Wi-Fi Direct.

NEAR FIELD COMMUNICATION

Introduzione

Near Field Communication (NFC) è un set di tecnologie wireless a corto raggio (tipicamente non più di 4 centimetri). I dispositivi Android dotati di tecnologia NFC supportano tipicamente tre modalità operazionali:

1. Reader/writer mode, consistente nella lettura e/o scrittura di oggetti passive (NFC tags).
2. P2P mode, che consente al device di scambiare dati con altri peers
3. Card emulation mode, nella quale il device si comporta come una NFC card.

Android dispone di un Sistema di dispatch dei tag che analizza i tag NFC scansionati, ne fa il parsing, e cerca le applicazioni alle quali potrebbero interessare. In particolare:

1. Fa il parsing del tag NFC, e ritrova il tipo MIME o l'URI che identifica i dati del payload del tag.
2. Incapsula il tipo MIME o l'URI e il payload in un intent.
3. Viene lanciato l'intent (e conseguentemente inizia di una activity interessata)

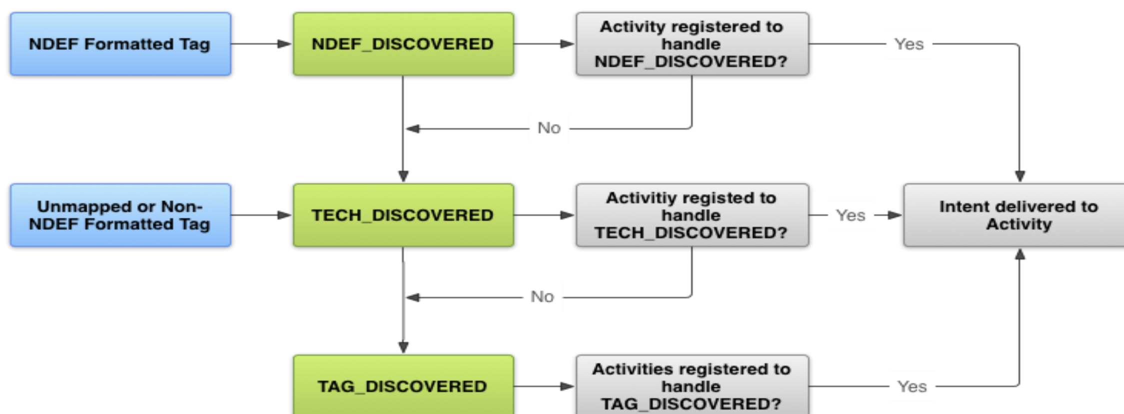
Lo standard largamente supportato da Android è NDEF. I dati NDEF vengono incapsulati in un NdefMessage, il qualche contiene uno o più NdefRecord. Ogni NdefRecord deve essere well-formed, secondo il tipo di record che si vuole creare. Android supporta anche altri tipi di tag, che non contengono dati NDEF, che possiamo gestire tramite le classi contenute nel package android.nfc.tech.

Un NdefRecord well-formed contiene:

- **3-bit TNF (Type Name Format):** indica come interpretare il campo type.
- **Type (di lunghezza variabile):** descrive il tipo di record. Se il TFN è settato a TFN_WELL_KNOWN, questo campo contiene un RTD.
- **Identificatore (di lunghezza variabile e opzionale):** un identificatore univoco per il record
- **Payload (di lunghezza variabile):** i veri e propri dati del record.

Se il parsing avviene correttamente, il sistema di dispatch dei tag inferisce il tipo di messaggio in base al tipo del primo record e incapsula il messaggio all'interno di un intent ACTION_NDEF_DISCOVERD. Se il sistema non riesce a determinare il tipo di tag, un intent ACTION_TECH_DISCOVERED ne incapsula le tecnologie e il payload.

Gli intent del Sistema di dispatch hanno priorità come da schema:



Dettagli pratici

Nel file manifest è necessario dichiarare:

- **<uses-permission android:name="android.permission.NFC" />** permesso necessario per l'utilizzo dell'hardware NFC.
- **<uses-sdk android:minSdkVersion="10"/>** per le API necessarie (almeno 10, per avere funzionalità più ampie meglio 14)

È stato omesso il tag <uses-feature …> in quanto ai fini dell'applicazione l'uso di NFC non è cruciale.

Ad ogni paziente viene associato un tag, nel quale si registrano il codice del paziente, data e ora dell'ultima rilevazione (un record diverso per ogni campo). Lettura e Scrittura del tag vengono effettuate in due activity differenti: la prima utilizza il sistema di dispatch dei tag intent, la seconda il sistema di foreground dispatch.

Inoltre, negli Ndefrecord di tipo testuale, il primo byte di payload non contiene dati veri e propri, ma informazioni riguardanti il formato del testo: in particolare, il primo bit indica se la codifica è UTF-8 o UTF-16, i restanti il codice della lingua.

Lettura dei tag

Innanzitutto nel manifest si dichiara per l'activity dedicata alla lettura l'intent filter:

```
<intent-filter>
                <action android:name="android.nfc.action.NDEF_DISCOVERED" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="text/plain" />
 </intent-filter>
```

In questo modo il sistema di dispatch lancerà l'activity ogni qualvolta che verrà effettuato lo scan di un tag conforme allo standard NDEF e di tipo testuale. (Nel caso ci siano altre activity registrate per ottenere questo intent, comparirà l'usuale Activity Chooser, lasciando all'utente il compito di decidere).

L'activity che riceve l'intent e presenta all'utente i dati letti è la classe ScanNFC, la quale utilizza una classe di utility (NFCReading.java), che fornisce metodi statici per la creazione dei record letti e il parsing del loro payload.

In ScanNFC si controlla l'intent ricevuto:

```
if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction()) &&
"text/plain".equals(intent.getType() ))
{
              NdefMessage[] messages = NFCreading.readNdefMessageFromTag(intent);

              String[] result = NFCreading.readRecordsFromMessage(messages[0]);
              if(result != null)
                     {
                            managing_intent = true;
                            new AsyncTaskRecuperaInfo(this).execute(result);
                     }
              else
             findViewById(R.id.textViewDatiNonPertinenti).setVisibility(View.VISIBLE);
}
```

Il metodo readNdefMessageFromTag tratta l'intent. Vengono recuperati i messaggi all'interno del tag (di fatto quello di interesse è sempre uno solo, in quanto le API di scrittura consentono la scrittura di un unico messaggio in un Tag). In caso di errore vieni restituito un array contenente un messaggio composto da un record di tipo TNF_UNKNOWN.

```
static public NdefMessage[] readNdefMessageFromTag(Intent intent)
	{
		NdefMessage[] message = null;
		Parcelable[] rawMessages = 
intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
		if (rawMessages != null) {
			message = new NdefMessage[rawMessages.length];
			for (int i = 0; i < rawMessages.length; i++) {
				message[i] = (NdefMessage) rawMessages[i];
			}
		} else {
			byte[] empty = new byte[] {};
			NdefRecord record = new NdefRecord ( NdefRecord.TNF_UNKNOWN, empty, 
empty, empty );
			NdefMessage msg = new NdefMessage( new NdefRecord[] { record } );
			message = new NdefMessage[] { msg };
		}

		return message;
	}
```

Il metodo readRecordsFromMessage effettua il vero e proprio parsing dei record contenuti nel messaggio ndef. I 3 record devono essere di tipo TNF_WELL_KNOWN e RTD_TEXT, per ogni record viene recuperata la stringa testuale (trattando l'encoding) e memorizzata in un array. Viene eseguito poi un controllo sulle stringhe riguardanti la data e l'ora.

```
static public String[] readRecordsFromMessage(NdefMessage message)
	{
		// 3 Record: codice a barre, data e ora
		String[] result = new String[3];
		for(int j=0; j<3 ; j++)
		{

			NdefRecord record = message.getRecords()[j];
			if(record.getTnf() != NdefRecord.TNF_WELL_KNOWN ||
!Arrays.equals(record.getType(), NdefRecord.RTD_TEXT))
				return null;
			byte[] payload = record.getPayload();
			//Text Encoding
			String textEncoding = ((payload[0] & 128) == 0) ? 
		   "UTF-8" : "UTF-16";
			//Language Code
			int languageCodeLength = payload[0] & 0063;

			try {
				result[j] = new String(payload, languageCodeLength + 1,
payload.length - languageCodeLength - 1, textEncoding);
			} catch (UnsupportedEncodingException e) {

				return null;
			}
		}

		if(CorrectParsing(result))
			return result;
		else return null;
	}
```

```
static private boolean CorrectParsing(String[] string_array)
	{
		String data = string_array[1];
		String ora = string_array[2];
		SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy" + " " + "HH:mm");

		try {
			formatter.parse(data + " " + ora);
			return true;
		} catch (ParseException e) {
			return false;
		}

	}
```

Una volta verificata la correttezza sintattica dei record, viene eseguito un thread nell'activity ScanNFC che si occupa di recuperare i dati del paziente (se registrato sul dispositivo) e le rilevazioni (se presenti).
È possibile effettuare nuove rilevazioni anche se il paziente non è registrato per poi mandarle a un altro dispositivo in un secondo momento.

```
private class AsyncTaskRecuperaInfo extends AsyncTask < String , Void , Boolean > {

		private ProgressDialog      progressDialog ;
		private Activity            targetActivity ;
		private Paziente p = null;
		private List<Rilevazione> rilevazioni = null;

		public AsyncTaskRecuperaInfo ( Activity activity ) {
			this.targetActivity = activity ;
			// this.needToShow = true;
			progressDialog = new ProgressDialog ( targetActivity ) ;
			progressDialog.setCancelable ( false ) ;
			progressDialog.setMessage ( getString(R.string.attendere) ) ;
			progressDialog.setTitle ( getString(R.string.recuperodati)) ;
			progressDialog.setIndeterminate ( true ) ;
		}

		@ Override
		protected void onPreExecute ( ) {
			progressDialog.show ( );
		}

		@ Override
		protected Boolean doInBackground ( String ... params) {

			codice_a_barre = params[0];
			data = params[1];
			ora = params[2];
			PazientiDBAdapter datasourcePazienti = new
PazientiDBAdapter(getApplicationContext());
			RilevazioneDBAdapter datasourceRilevazioni = new
RilevazioneDBAdapter(getApplicationContext());
			datasourcePazienti.open();
			p = datasourcePazienti.getPaziente(codice_a_barre);
			datasourcePazienti.close();

			datasourceRilevazioni.open();
			rilevazioni = datasourceRilevazioni.getRilevazioni(codice_a_barre);
			datasourceRilevazioni.close();
			if(p != null)
				return true;
			else
				return false ;
		}
```

```
            @ Override
            protected void onPostExecute (Boolean paziente_presente) {
                   if(progressDialog != null && progressDialog.isShowing()){
                          progressDialog.dismiss ( ) ;

                          textView_codice_a_barre.setText(codice_a_barre);
                          textView_data.setText(data);
                          textView_ora.setText(ora);

                          if(paziente_presente)
                          {
                   findViewById(R.id.LayoutPazientePresente).setVisibility(View.VISIBLE);
                          textView_nomecognome.setText(p.getNome() +  " " + p.getCognome());
                          textView_protocollo.setText(p.getNome_protocollo());
                          }
                          else{

       findViewById(R.id.textViewPazienteNonPresente).setVisibility(View.VISIBLE);
                          }

                          if(rilevazioni != null && !rilevazioni.isEmpty())
                          {

       findViewById(R.id.textViewNessunaRilevazione).setVisibility(View.GONE);
       ListView listviewRilevazioni = (ListView)findViewById(R.id.ListViewRilevazioni);
                                 Collections.sort(rilevazioni);
                                 Collections.reverse(rilevazioni);
                                 ArrayRilevazioneAdapter arrayrilevazioneadapter =new
ArrayRilevazioneAdapter(targetActivity, R.layout.rowrilevazione, rilevazioni);
                                 listviewRilevazioni.setAdapter(arrayrilevazioneadapter);
                                 listviewRilevazioni.setVisibility(View.VISIBLE);
                                 Rilevazione rilevazionescan = new Rilevazione(data, ora);
                                 Rilevazione rilevazioneheadlist = rilevazioni.get(0);
                                 if(rilevazionescan.compareTo(rilevazioneheadlist) < 0)
                                 {
                                        textView_data.setText(rilevazioneheadlist.getData());
                                        textView_ora.setText(rilevazioneheadlist.getOra());
                                 }
                          }
                          else
                   findViewById(R.id.textViewNessunaRilevazione).setVisibility(View.VISIBLE);
                   }
            }
       }
```

Scan NFC

Codice a barre	22999		
Rilevazione più recente	24/02/2014 11:49		
Pinco Pallino	ProtocolloA		
24/02/2014			11:49
SVS Riposo	1	Sforzo	1
VAS Riposo	0	Sforzo	0
Sede	1	Sedazione	1
FR	33	SO2	6
PA	44	FC	644
24/02/2014			09:49
SVS Riposo	1	Sforzo	1
VAS Riposo	0	Sforzo	0
Sede	1	Sedazione	1
FR	3434	SO2	43
PA	4	FC	4

Scan NFC

Codice a barre 1929291
Rilevazione più recente 24/02/2014 11:26
Paziente non registrato su questo dispositivo
Nessuna Rilevazione Presente

Scrittura dei tag

Dopo il salvataggio di una rilevazione, è possibile scrivere nel tag associato al paziente la data e l'ora di quest'ultima. Poiché ragionevolmente le due cose devono avvenire contestualmente, l'activity che si occupa di salvare la rilevazione, provvede anche della scrittura su tag. Quest'ultima funzione è fornita ovviamente ai soli device che dispongono dell'opportuno hardware, in caso contrario essa risulta totalmente mascherata. Inoltre, poiché è proprio l'activity in foreground che deve occuparsi della scrittura, viene usato il sistema di foreground dispatch.

Quest'ultimo, fa sì che, qualora venisse fatto il discovery di un tag conforme a certe caratteristiche di interesse per l'activity in stato running, l'intent viene rigirato direttamente a quest'ultima (che quindi è come se avesse la "priorità massima") e viene reperita l'informazione EXTRA_TAG; nel caso venga fatto il discovery di un tag non di interesse per l'activity, questo viene trattato dal normale sistema di dispatch degli intent.

```
@Override
	protected void onResume() {
		datasource.open();
		super.onResume();
		if( nfc_adapter != null)
		{
			nfc_adapter.enableForegroundDispatch(this, pending_intent,
writeTagFilters, null);
			if(!nfc_adapter.isEnabled())
				this.askUserToActivateNfc();
		}

	}
```

```
@Override
	protected void onPause() {
		datasource.close();
		if(nfc_adapter != null)
			nfc_adapter.disableForegroundDispatch(this);
		super.onPause();
	}
```

```
@Override
	protected void onNewIntent(Intent intent) {
		super.onNewIntent(intent);
		if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())) {
			// validate that this tag can be written
			detectedTag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

		}
	}
```

```
@Override
	protected void onCreate(Bundle savedInstanceState) {

...
nfc_adapter = NfcAdapter.getDefaultAdapter(this);

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

		IntentFilter ndefDetected = new
IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
```

```java
                try {
                        ndefDetected.addDataType("text/plain");
                }
                catch (MalformedMimeTypeException e) {
                        throw new RuntimeException("fail", e);
                }
                writeTagFilters = new IntentFilter[] { ndefDetected };

...
}
```

Anche in questo caso, per rendere il codice dell'activity meno pesante, è stata creata una classe, NFCwriting.java, che espone un metodo per la creazione dei record, e uno che si occupa della scrittura.

Come già detto, nei record che contengono messaggi testuali il primo byte di payload dà informazioni sull'encoding e la lingua.

```java
static public NdefRecord createRecord(String string_to_write)
        {
                Locale locale= new Locale("it","IT");
                byte[] langBytes = locale.getLanguage().getBytes(Charset.forName("UTF-8"));
                boolean encodeInUtf8=false;
                Charset utfEncoding = encodeInUtf8 ? Charset.forName("UTF-8") :
        Charset.forName("UTF-16");
                int utfBit = encodeInUtf8 ? 0 : (1 << 7);
                char status = (char) (utfBit + langBytes.length);
                byte[] textBytes = string_to_write.getBytes(utfEncoding);
                byte[] data = new byte[1 + langBytes.length + textBytes.length];
                data[0] = (byte) status;
                System.arraycopy(langBytes, 0, data, 1, langBytes.length);
                System.arraycopy(textBytes, 0, data, 1 + langBytes.length, textBytes.length);
                NdefRecord textRecord = new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
NdefRecord.RTD_TEXT, new byte[0], data);
                return textRecord;
        }
```

Il metodo che si occupa della scrittura ritorna una stringa contenente l'esito dell'operazione. Per prima cosa cerca di creare un oggetto NDef a partire dal tag. Se questo non è null, significa che il tag è già formattato in NDef per cui, previo controllo che il tag sia scrivibile e di grandezza sufficiente, si abilitano le operazioni di I/O col tag, si scrive il messaggio e si chiude la comunicazione. Se il tag non risulta formattato in NDef, ma supporta tale tecnologia, si cerca di formattare tale tag e di scrivere il messaggio (tramite il metodo format di esposto dalla classe ndefFormatable). In caso contrario, l'operazione di scrittura risulta impossibile.

```java
static public String writeNdefMessageToTag(NdefMessage message, Tag detectedTag) {
                int size = message.toByteArray().length;

                try {
                        Ndef ndef = Ndef.get(detectedTag);
                        if (ndef != null) {
                                ndef.connect();
                                if (!ndef.isWritable()) {
                                        return "Il tag è read-only";
                                }
                                if (ndef.getMaxSize() < size) {
                                        return "Spazio insufficiente, la capacità è " +
ndef.getMaxSize() + " bytes, il messaggio è " + size + " bytes";

                                }
                                ndef.writeNdefMessage(message);
                                ndef.close();
```

```
                        return successo;
                } else {
                        NdefFormatable ndefFormat = NdefFormatable.get(detectedTag);
                        if (ndefFormat != null) {
                                try {
                                        ndefFormat.connect();
                                        ndefFormat.format(message);
                                        ndefFormat.close();
                                        return successo;
                                } catch (IOException e) {
                                        return "Formattazione fallita";
                                }
                        } else {
                                return "NDEF non supportato";
                        }
                }
        } catch (Exception e) {
                return "Operazione di scrittura fallita";
        }
}
```

Nel caso la scrittura non vada a buon fine, un alert dialog mostra la causa dell'errore e invita l'utente a riprovare.

```
private void askUserToRetryNFCWriting(String error)
        {

                AlertDialog.Builder builder = new AlertDialog.Builder(context);
                builder.setCancelable(false)
                .setMessage(getResources().getString(R.string.riprovare))
                .setTitle(error)
                .setPositiveButton(getResources().getString(R.string.si), new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                                String esito = NFCwriting.writeNdefMessageToTag(msg_nfc,
detectedTag);
                                if(esito.equals(NFCwriting.successo))
                                {
                                        finish();
                                }
                                else
                                        askUserToRetryNFCWriting(esito);
                        }
                })
                .setNegativeButton(getResources().getString(R.string.no), new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                                finish();
                        }
                });

                AlertDialog alert = builder.create();
                alert.show();
        }
```

WI-FI PEER-TO-PEER

Introduzione

Wi-fi p2p permette a due device dotate di interfaccia wi-fi di connettersi direttamente fra loro senza access-point. Le API di wi-fi p2p consistono in:

- Metodi che permettono di fare il discovery, richiedere e connettersi a un altro peer, definiti nella classe WifiP2pManager.
- Listeners che permettono di essere notificati dell'esito della chiamate ai metodi di Wifip2pManager. Tali metodi ricevono uno specifico listener come parametro della signature.
- Intents che notificano la presenza di eventi rilevati dal framework di wi-fi P2P, (connessione caduta, nuovo peer trovato ecc…)

Innanzitutto, bisogna dichiarare nel manifest l'SDK minimo e i permessi per avere accesso l'hardware necesessario

```
<uses-sdk android:minSdkVersion="14" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

In secondo luogo, è necessario creare un broadcast receiver che riceva gli intent di interesse. Nel costruttore del broadcastReceiver tipicamente si passa anche l'activity sulla quale si va a lavorare, in modo tale da agganciare nel metodo onReceive() del broadcast receiver, i metodi opportuni dell'activity che verranno invocati in callback.

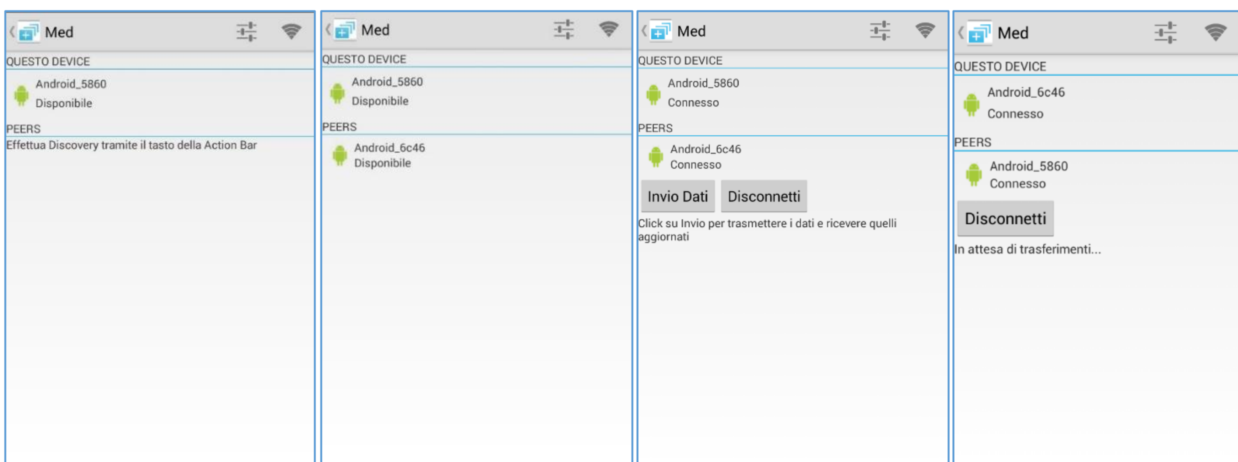
```
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {


    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
            MyWifiActivity activity) {
        super();
        this.mManager = manager;
        this.mChannel = channel;
        this.mActivity = activity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            //controllo se wi-fi è abilitato o meno
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
            //si richiede una lista dei peers e tipicamente si invoca in callback un metodo
dell'activity per aggiornare la lista dei peers disponibili
        } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
            //lo stato della connessione a un peer è cambiata. Tipicamente Nel caso di
connessione si richiedono le info di connessione per identificare il groupOwner. In caso di
disconnessione l'activity torna nel suo stato iniziale
        } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
            // Lo stato della connessione di questo device è cambiato
        }
    }
}
```

Nell'activity principale viene registrato il broadcast receiver, dichiarandone anche gli intent filters, e vengono esposti due fragment: uno che mostra lo stato del device e dei peer disponibili, l'altro per effettuare l'operazione di scambio dati una volta stabilita una connessione.
L'actionbar presenta un tasto che rimanda alle impostazioni wi-fi (in modo da attivarlo se risulta non ancora abilitato) e un tasto che attua il discovery (invoca il metodo discoverPeers() del WifiP2PManager).
Se il processo di discovery ha successo, il sistema manda in broadcast l'intent WIFI_P2P_PEERS_CHANGED_ACTION, che verrà ricevuto dal broadcast receiver e provvederà a mostrare i peers presenti (è possibile ottenere già una lista dei peers trovati tramite il metodo requestPeers()).
A questo punto, un click su un peer mostrerà l'opportuno bottone che avvierà la connessione (invocazione del metodo connect()del WifiP2PManager. In maniera analoga a prima, il broadcast receiver sarà registrato agli intent WIFI_P2P_CONNECTION_CHANGED_ACTION in modo tale da aggiornare il fragment dei tasti a connessione avvenuta (verrà mostrato il tasto di invio dei dati per il device client, mentre per il device passivo solo il tasto di disconnessione).



Trasferimento dei dati

Una volta stabilita una connessione, un device avrà il ruolo di GroupOwner (server), l'altro di client.
Il GroupOwner istanzierà su un thread (o meglio un AsyncTask) un ServerSocket in attesa di connessione del client su una porta. Il client, da lato suo, creerà una socket utilizzando l'IP del GroupOwner e la porta su cui è in ascolto.

Il flusso della comunicazione è il seguente:

1. Il Client manda al Server tutti i dati relativi a pazienti, protocolli, rilevazioni e pazienti cancellati.
2. Il Server riceve i dati dal client e ne fa il merge coi dati presenti nel proprio DB (vedi seguito), e rimanda indietro al Client i dati aggiornati se tutto è andato a buon fine.
3. Il Client riceve i dati aggiornati e li carica nel proprio DB

Elaborazione dei dati inviati al device Server

Per quanto riguarda il merge dei dati ricevuti dal device che opera da Server:

- Pazienti: per ogni paziente ricevuto, innanzitutto si controlla se il corrispettivo codice si trova fra quelli dei pazienti cancellati. In caso positivo, la entry viene ignorata e si passa alla successiva, in caso negativo si cerca il codice fra i pazienti già registrati nel DB: se non è presente, la entry viene aggiunta al DB del Server altrimenti, in caso contrario, viene confrontato il timestamp dell'ultima modifica: se i dati ricevuti risultano essere più recenti, i dati del DB vengono aggiornati.
- Protocolli: nel caso il codice a barre di un protocollo corrisponda a un paziente cancellato, questo non viene inserito nel DB. In caso contrario, viene inserito nel DB del Server (non vi sono problemi

di aggiornamento, in quanto, per le ipotesi fatte all'inizio, un protocollo non varia per tutta la durata del trattamento).

- Rilevazioni: caso analogo ai protocolli, si controlla che il codice a barre delle rilevazioni non corrisponda a un paziente cancellato, e si procede all'inserimento nel DB.
- Pazienti cancellati: per ogni entry ricevuta, si procede alla cancellazione di tutti i dati corrispondenti nel DB. Questi codici a barre vengono poi aggiunti a quelli già presenti nel Server, per eventuali future propagazioni ad altri dispositivi.

La parte di elaborazione vera e propria è tutta a carico del device Server. Nel caso l'elaborazione del device Server vada a buon fine, il device Client non farà che il drop di tutte le proprie tabelle e ricaricherà di colpo tutti dati già aggiornati dal device Server.

Pulizia della table 'Pazienti Cancellati'

Un "problema" che a questo punto potrebbe sorgere è: fino a quando vengono conservati i codici a barre relativi agli utenti non più presenti? Per far sì che la pulizia di questa tabella non sia a carico dell'utilizzatore, e comunque non risulti essere un'operazione invadente, si è deciso di avviarla in modo automatico quando il device viene messo sotto carica o quando viene riavviato.
Introduciamo nell'applicazione due componenti molto semplici: un Service e un broadcast Receiver.

Il Broadcast Receiver registrato nel manifest, sarà interessato a ricevere gli Intent aventi action di tipo BOOT_COMPLETED e ACTION_POWER_CONNECTED e avrà il semplice compito di scatenare il Service, che in modo molto semplice, si occuperà di cancellare dalla tabella 'PAZIENTI_CANCELLATI' tutte le entry più vecchie di un mese.

```
public class ChargingAndBootReceiver extends BroadcastReceiver {

	@Override
	public void onReceive(Context context, Intent arg1) {
		Intent service = new Intent(context, DeleteOldDataService.class);
		context.startService(service);
	}
}
```

```
public class DeleteOldDataService extends Service{

	@Override
	public IBinder onBind(Intent intent) {

		return null;
	}

    @Override
    public void onStart(Intent intent, int startid)
    {
		SimpleDateFormat dateFormat =
	       new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault());
		Calendar cal = Calendar.getInstance();
		cal.add(Calendar.MONTH, -1);
		Date date = cal.getTime();
		String whereclause = dateFormat.format(date);

		PazientiDBAdapter datasource = new PazientiDBAdapter(this);
		datasource.open();
		datasource.deletePazientiCancellati(whereclause);
		datasource.close();
    }
}
```