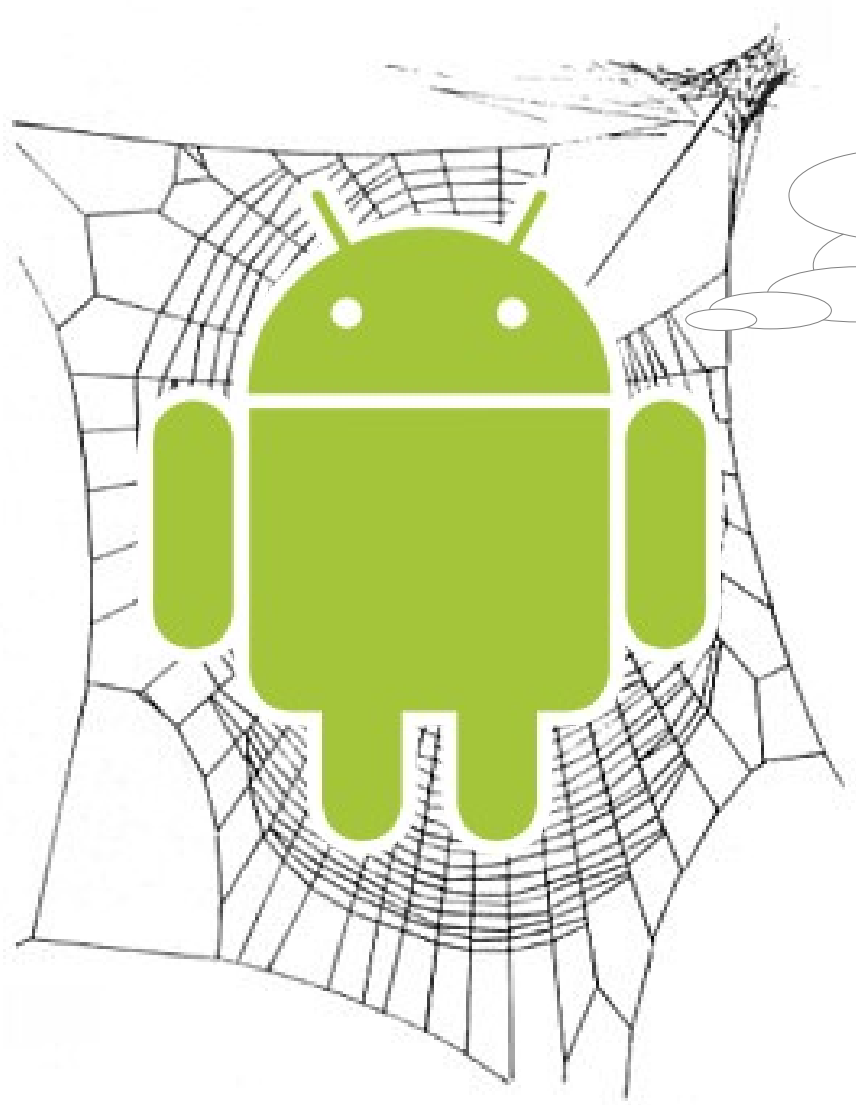
Communication réseau sous Android



Master 2 informatique 2012-2013

Communication réseau

- Implantation de l'API java.net du JDK --> cours applications réseau M1
- Nouvelle API android.net :
  - **Informations sur la connectivité réseau**
  - Interface pour services VPN
  - **Implantation de clients HTTP**
    - **Client java.net.HttpURLConnection** : à privilégier sur les versions récentes
    - Client Apache HTTP (android.http) : moins maintenu désormais
  - Implantation d'un client RTP (Real-time Transport Protocol)
  - Implantation d'un client SIP
  - **Gestionnaire Wi-Fi**
  - **Téléphonie (appels cellulaires, SMS, MMS)**
- Pour accéder à toute machine extérieure : *android.permission.INTERNET*

java.net.HttpURLConnection

- Client HTTP simple instantiable depuis une URL :
  - ```
    URL url = new URL(address) ;
    HttpURLConnection conn = (HttpURLConnection)url.openConnection() ;
    InputStream is = new BufferedInputStream(conn.getInputStream()) ;
    ```
- Client HTTP bloquant : à employer dans une nouvelle thread. Timeout en lecture avec setReadTimeout(int millis) ; lève *SocketTimeoutException*
- Pour envoyer un fichier
  - ```
    conn.setDoOutput(true) ;
    conn.setChunkedStreamingMode(0) ;
    OutputStream out = new BufferedOutputStream(conn.getOutputStream()) ;
    ```
- Ne pas oublier de fermer la connexion : *conn.disconnect()*
- Gestion automatique d'un pool de connexions ouvertes (réutilisation de connexions) ; désactivation avec *System.setProperty("http.keepAlive", "false")*
- *android.net.http.HttpResponseCache* fournit un cache depuis ICS

Gestion des en-têtes

- En-têtes de la requête :
  - *setRequestMethod(String method)* : choix de la méthode (GET, POST, PUT...)
  - *addRequestProperty(String field, String value)*
- En-têtes de la réponse :
  - *Date getDate()* : date de la réponse
  - *Map<String, List<String>> getHeaderFields()* : champs d'en-tête de la réponse
  - *long getLastModified()* : date de dernière modification (timestamp Unix)

Gestion des cookies

- On initialise le *CookieHandler* de la VM :
```
CookieManager cm = new CookieManager()
CookieHandler.setDefault(cm) ;
```
- On ajoute des cookies :
  - Manuellement
```
HttpCookie cookie = new HttpCookie("key", "value")
cookie.setDomain("univ-mlv.fr")
cm.getCookieStore().add(new URI("http://www.univ-mlv.fr", cookie) ;
```
  - Automatiquement par HttpURLConnection
- Les cookies résident dans la mémoire du processus : pour un stockage persistent il faut fournir un *CookieStore* personnalisé à *CookieManager*

Exemple : POST en urlencoded

```
public static void postURLEncoder(HttpURLConnection conn, Map<String, ?> data)
    throws IOException
{
        // Encode data
        conn.setDoOutput(true);
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        conn.setChunkedStreamingMode(0);
        // We don't already know the total size, does not work with HTTP/1.0
        conn.connect();
        Writer w = new BufferedWriter(
            new OutputStreamWriter(conn.getOutputStream(), "ASCII"));
        boolean first = true;
        for (Map.Entry<String, ?> entry: data.entrySet())
        {
            if (! first) w.write("&"); // Add a separator
            else first = false;
            w.write(URLEncoder.encode(entry.getKey(), "UTF-8"));
            w.write("=");
            w.write(URLEncoder.encode(entry.getValue().toString(), "UTF-8"));
        }
        w.close();
}
```

Exemple : POST en form-data (supporte les fichiers)

```
public static void postFormData(HttpURLConnection conn, Map<String, Object> data) throws IOException
     {
          conn.setDoOutput(true);
          conn.setRequestMethod("POST");
          String boundary = Long.toHexString(new Random().nextLong()); // Random string for boundary
          conn.setRequestProperty("Content-Type", "multipart/form-data; boundary=" + boundary);
          conn.setChunkedStreamingMode(0);
          OutputStream os = new BufferedOutputStream(conn.getOutputStream());
          for (Map.Entry<String, Object> entry: data.entrySet())
          {
               if (entry.getValue() == null) continue;
               String header = "--" + boundary + "\r\n"
                    + "Content-Disposition: form-data; field=\"" + entry.getKey() + "\"\r\n"
                    + "Content-Type: " + ((entry.getValue() instanceof String)?
                         "text/plain; charset=UTF-8":"application/octet-stream") + "\r\n\r\n";
               os.write(header.getBytes("ASCII"));
               if (entry.getValue() instanceof String)
                    os.write(entry.getValue().toString().getBytes("UTF-8"));
               else if (entry.getValue() instanceof InputStream)
               {
                    InputStream is = (InputStream)entry.getValue();
                    byte[] buffer = new byte[BUFFER_LEN];
                    for (int r = is.read(buffer); r >= 0; r = is.read(buffer))
                         os.write(buffer, 0, r);
               } else
                    throw new IOException(
                         "Object of type " + entry.getValue().getClass() + " is not supported");
               os.write("\r\n".getBytes("ASCII"));
          }
          os.write(("--" + boundary + "--\r\n").getBytes("ASCII"));
          os.close();
     }
```

Gestion des connexions

- Récupération du *ConnectivityManager* : *Context.getSystemService(Context.CONNECTIVITY_SERVICE)*
- Permissions nécessaires : *android.permission.{ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE}*
- Informations réseau : *NetworkInfo[] getAllNetworkInfo()*
  - *NetworkInfo* indique le type de réseau (mobile, Wi-Fi, Wimax, Ethernet, Bluetooth), son état (demande DHCP, connecté, roaming...)
- Réseau de la route par défaut : *NetworkInfo getActiveNetworkInfo()*
  - Le terminal est "en ligne" si getActiveNetworkInfo().isConnected() est vrai
- Pour être informé des changements : réception d'Intent *ConnectionManager.CONNECTIVITY_ACTION* par un *BroadcastReceiver* (arguments *EXTRA_{EXTRA_INFO, IS_FAILOVER, NETWORK_INFO, NETWORK_TYPE, NO_CONNECTIVITY, OTHER_NETWORK_INFO, REASON}*)

Gestion du Wi-Fi

- Récupération du *WifiManager* : *Context.getSystemService(Context.WIFI_SERVICE)*
- Verrous Wi-Fi :
  - *createMulticastLock(String tag)* : verrou pour recevoir les datagrammes multicast
  - *WifiLock createWifiLock(String tag)* : verrou pour ne pas mettre en sommeil la pile Wi-Fi
    - wifiLock.acquire()
    - wifiLock.release()
- *WifiInfo getConnectionInfo()* : informations sur la connexion courante (BSSID, SSID, RSSI, adresse IP, bande passante...)

Scan de connexions Wi-Fi

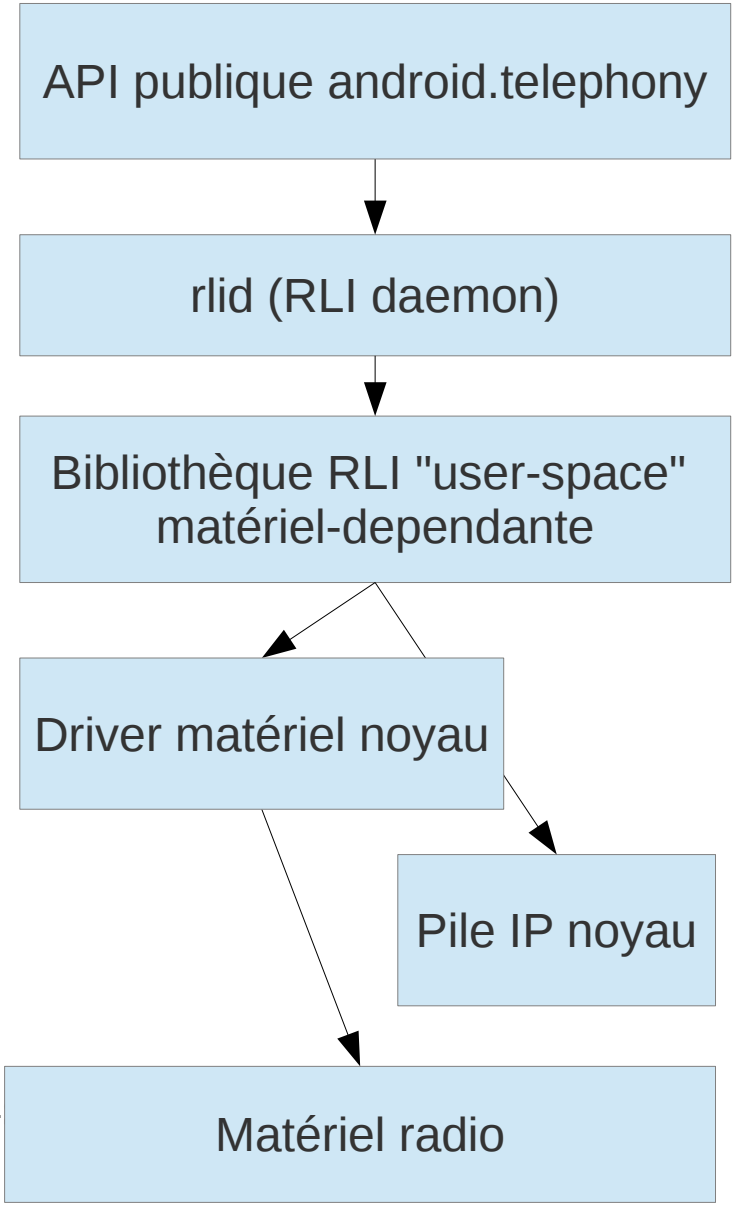
```
WifiManager manager = (WifiManager)getSystemService(Context.WIFI_SERVICE);
List<ScanResult> scanResult = null;

public void startScan()
{
    BroadcastReceiver receiver = new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {
            scanResult = manager.getScanResults();
            unregisterReceiver(this);
        }
    };
    registerReceiver(receiver,
        new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
    manager.startScan();
}
```

P2P en Wi-Fi direct

- Wi-Fi direct : protocole facilitant l'interconnexion de machines en mode ad-hoc
- Annonce et découverte de services avec Bonjour et UPnP
- Utilisation de *WifiP2pManager*
  - Initialisation
  - Enregistrement de services
  - Découverte de pairs et services
  - Connexion avec des pairs
- Exemples disponibles dans samples/WiFiDirectDemo

Téléphonie cellulaire

- API publique aux fonctionnalités limitées
- Permet d'accéder à l'état de la communication avec le réseau cellulaire et d'être informé de changements
- Ne permet pas d'accéder aux couches basses du Radio Layer Interface pour créer son dialer personnalisé



Numéroter un appel

- On demande les permissions dans le manifeste :

```
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

  - Numérotation indirecte sans permission avec *Intent.ACTION_DIAL*

- On utilise le numéroteur installé en lui envoyant un Intent :

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:+33160957500"));
startActivity(callIntent);
```

- On enregistre un PhoneCallListener pour être informé de l'état de l'appel

```
// We create the listener
PhoneCallListener phoneListener = new PhoneCallListener() {
      private boolean calling = false ;
      @Override public void onCallStateChanged(int state, String incomingNumber) {
            switch (state)
            {
                  case TelephonyManager.CALL_STATE_RINGING :
                        // Phone is ringing
                        break ;
                  case TelephonyManager.CALL_STATE_OFFHOOK :
                        // Phone call is starting
                        calling = true ;
                        break ;
                  case TelephonyManager.CALL_STATE_IDLE :
                        // Either the phone has not dialed yet (!calling), or the call is finished (calling == true)
                        break ;
            }
}
// We register the listener
TelephonyManager telephonyManager = (TelephonyManager) this.getSystemService(Context.TELEPHONY_SERVICE);
telephonyManager.listen(phoneListener,PhoneStateListener.LISTEN_CALL_STATE);
```

Informations sur le réseau cellulaire avec *TelephonyManager*

- *int getDataActivity()* : activité de la connexion data (DATA_ACTIVITY_{NONE, IN, OUT, INOUT, DORMANT})
- *int getDataState()* : état de la connexion data (DATA_{DISCONNECTED, CONNECTING, CONNECTED, SUSPENDED)
- *String getDeviceId()* : retourne le numéro IMEI (requiert la permission READ_PHONE_STATE)
- *boolean isNetworkRoaming()* : indique si l'on est connecté sur un réseau en itinérance
- *List<NeighboringCellInfo> getNeighboringCellInfo()* : information sur les cellules voisines (CID, RSSI)
- *String getNeworkOperator()* : retourne le code numérique MCC+MNC de l'opérateur
- *int getNetworkType()* : retourne le type de réseau utilisé pour les transmissions data (GPRS, EDGE, UTMS, LTE...)

PhoneStateListener

- Enregistrement du listener avec *TelephonyManager.listen(PhoneStateListener listener, int events)*
- Evénements supportés :
  - *LISTEN_NONE* : aucun évènement
  - *LISTEN_CALL_FORWARDING_INDICATOR* : changements de l'état de redirection des appels
  - *LISTEN_CALL_STATE* : état d'appel
  - *LISTEN_CELL_INFO* : information de cellule
  - *LISTEN_CELL_LOCATION* : changement concernant la localisation obtenue d'après les cellules
  - *LISTEN_DATA_ACTIVITY* : activité data
  - *LISTEN_DATA_CONNECTIVITY_STATE* : état data
  - *LISTEN_SERVICE_STATE* : état de couverture (réseau dispnible, urgences seules...)
  - *LISTEN_SIGNAL_STRENGTHS* : changement de puissance du réseau reçu

Appels entrants

- Réception de l'évènement broadcast *TelephonyManager.ACTION_READ_PHONE_STATE_CHANGED* (requiert la permission *READ_PHONE_STATE*)
  - *EXTRA_STATE* : état de l'appel (*EXTRA_STATE_RINGING* nous intéresse)
  - *EXTRA_INCOMING_NUMBER* : ID de l'appelant
- Action possible :
  - Réponse en envoyant un Intent *ACTION_ANSWER*

Envoi de SMS

- Obtention du *SMSManager* : *SMSManager.getDefault()*
- Permission requise : *android.permission.SEND_SMS* (permission permettant d'envoyer vers n'importe quel numéro ; y compris surtaxés)
- Envoi de messages :
  - *sendTextMessage(String destination, String scAddress, String message, PendingIntent sentIntent, PendingIntent deliveryIntent)*
  - *sendDataMessage(String destination, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)*
  - *sendMultipartTextMessage(String destination, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntent, ArrayList<PendingIntent> deliveryIntent)* : permet d'envoyer un SMS en plusieurs morceaux ; on divise auparavant le message en morceaux avec *divideMessage*

1 : Activité d'envoi de SMS

```
public class SMSSender extends Activity
{
     // A list saving the statuses of the SMS parts
     private final ArrayList<String> sentStatuses = new ArrayList<String>();
     private ArrayAdapter<String> sentStatusesAdapter;
     private BroadcastReceiver br; private long sendTime;

     @Override protected void onCreate(Bundle savedInstanceState)
     {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_smssender);
          // Initialize the adapter displaying the statuses
          sentStatusesAdapter = new ArrayAdapter<String>(this, R.layout.simpletextview, sentStatuses);
          ((ListView)findViewById(R.id.smsSentStatus)).setAdapter(sentStatusesAdapter);
     }

     public void onSend(View v)
     {
          // Send a SMS using the recipient and message given in the form
          String recipient = ((EditText)findViewById(R.id.smsRecipient)).getText().toString();
          String message = ((EditText)findViewById(R.id.smsMessage)).getText().toString();
          send(recipient, message);
     }

     private static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz ";
     private static final int RANDOM_MESSAGE_LENGTH = 200;

     public void onGenerateRandomMessage(View v)
     {
          Random r = new Random();
          StringBuilder sb = new StringBuilder();
          for (int i = 0; i < RANDOM_MESSAGE_LENGTH; i++)
               sb.append(ALPHABET.charAt(r.nextInt(ALPHABET.length())));
          ((EditText)findViewById(R.id.smsMessage)).setText(sb.toString());
     }

     ...
}
```

2 : Envoi de SMS

```
public static final String SENT_ACTION = SMSSender.class.getName() + ".sentAction";
public static final String DELIVERY_ACTION = SMSSender.class.getName() + ".deliveryAction";

     public void send(String recipient, String message)
     {
          Log.v("smsSender", String.format("Sending message %s to %s", recipient, message));
          SmsManager sm = SmsManager.getDefault();
          ArrayList<String> split = sm.divideMessage(message);
          sentStatuses.clear();
          for (int i = 0; i < split.size(); i++) sentStatuses.add("sending part #" + i);
          sentStatusesAdapter.notifyDataSetChanged();
          // Register broadcast receivers to obtain the notifications
          if (br != null) unregisterReceiver(br);
          br = new BroadcastReceiver() { ... };
          IntentFilter filter = new IntentFilter();
          filter.addAction(SENT_ACTION);
          filter.addAction(DELIVERY_ACTION);
          registerReceiver(br, filter);
          ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>();
          ArrayList<PendingIntent> deliveryIntents = new ArrayList<PendingIntent>();
          for (int i = 0; i < split.size(); i++)
          {
               sentIntents.add(PendingIntent.getBroadcast
                    (this, 2*i, new Intent(SENT_ACTION).putExtra("chunkID", i), 0));
               deliveryIntents.add(PendingIntent.getBroadcast
                    (this, 2*i+1, new Intent(DELIVERY_ACTION).putExtra("chunkID", i), 0));
          }
          sendTime = System.nanoTime();
          sm.sendMultipartTextMessage(recipient, null, split, sentIntents, deliveryIntents);
     }
```

3 : Réception d'avis d'émission et de réception

```
new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent)
            {
                 Log.v("smsSender", "Retrieved intent=" + intent + ", extras=" + intent.getExtras());
                 int chunkID = intent.getIntExtra("chunkID", -1);
                 String message = null;
                 if (intent.getAction().equals(SENT_ACTION))
                 {
                      switch (getResultCode())
                      {
                      case Activity.RESULT_OK:
                           message = "sent"; break;
                      case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                           message = "send error #" + intent.getIntExtra("errorCode", -1); break;
                      case SmsManager.RESULT_ERROR_RADIO_OFF:
                           message = "radio off"; break;
                      default:
                           message = "unknown send error"; break;
                      }
                 } else if (intent.getAction().equals(DELIVERY_ACTION))
                 {
                      message = SmsMessage.createFromPdu(intent.getByteArrayExtra("pdu"))
                           .getDisplayMessageBody();
                 }
                 if (message != null && chunkID >= 0)
                 {
                      sentStatuses.set(chunkID, sentStatuses.get(chunkID) + ", "
                           + message + " after " + (System.nanoTime() - sendTime) / 1000000 + "ms");
                      sentStatusesAdapter.notifyDataSetChanged();
                 }
            }
        };
```

Réception de SMS

- Permission requise : *android.permission.RECEIVE_SMS*
- Réception d'un Intent *android.provider.Telephony.SMS_RECEIVED* diffusé par broadcast ordonné
  - Le Bundle communiqué contient un byte[][] (getExtras().get("pdus")) avec les PDUs (données binaires du SMS)
  - Conversion d'un PDU en SmsMessage : *SmsMessage.createFromPdu(pdu)*
  - Pour ne pas propager le boradcast ordonné : *BroadcastReceiver.abortBroadcast()*

Exemple de réception de SMS

```
/**
 * Intercept SMSs containing a given regular expression.
 */
public class SMSInterceptor extends BroadcastReceiver
{
	@Override
	public void onReceive(Context context, Intent intent)
	{
		Log.v("SMSInterceptor", "Receiving intent " + intent);
		Object[] pdus = (Object[])intent.getExtras().get("pdus");
		for (Object pdu: pdus)
		{
			SmsMessage message = SmsMessage.createFromPdu((byte[])pdu);
			Log.v("SMSInterceptor", "Content of the message: " + message.getDisplayMessageBody());
			if (isFiltered(message))
			{
				Log.v("SMSInterceptor", "Filtering message " + message);
				// The message is not propagated: it is lost
				abortBroadcast();
			}
		}
	}

	public static Pattern FILTER_PATTERN = Pattern.compile("^filter:.*$");

	public boolean isFiltered(SmsMessage message)
	{
		return FILTER_PATTERN.matcher(message.getDisplayMessageBody()).matches();
	}
}
```

Manifeste

```
<receiver android:name="fr.upemlv.helloworld.SMSInterceptor">
	<intent-filter android:priority="1000">
		<action android:name="android.provider.Telephony.SMS_RECEIVED" />
	</intent-filter>
</receiver>
```

Autres moyens de communication

- Bluetooth (android.bluetooth) :
  - permet de rendre découvrable, de découvrir d'autres périphériques, de les associer
  - propose des sockets client et serveur pour la communication en flux
  - supporte les profiles headset et A2DP pour la communication audio
- NFC (android.nfc) :
  - propose de scanner des tags NFC et de lancer l'activité la plus appropriée pour les traiter ; permet l'échange de petits volumes de données à faible distance entre appareils
- USB (android.usb) :
  - permet d'échanger des flux de données en tant qu'hôte ou accessoire
- Par ondes sonores (implantation d'un modem logiciel ;)