JSR 82

Bluetooth :Historique

- 1998: Création du Bluetooth SIG (Ericsson, Nokia, IBM, Intel, Toshiba)
- 1998: Spécification 0.7 (Bande de base + LMP)
- 1999: Spécification 1.0
- 2001: Spécification 1.1
- 2003: Version 1.2
- 2004 : Version 2.0 EDR (3 Mbits/s)

Caractéristiques

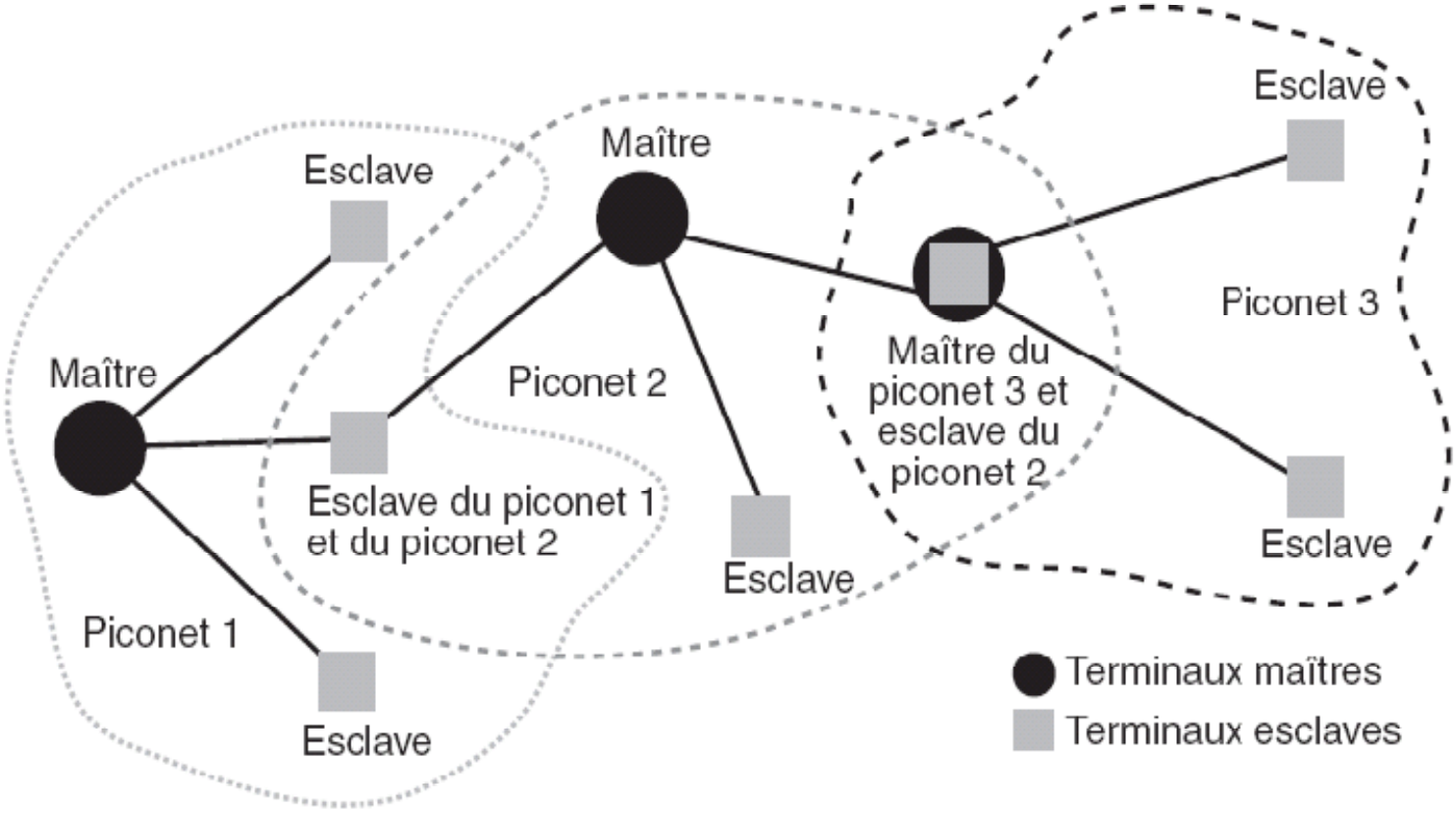
- Spectre de fréquences: 2.400-2.4835 GHz
- Canaux: 2402+k MHz, k=0,…,78

- Codage de l'information:
  - Période: 625 µS => 1600 slots
- Technique de transmission des données
  - saut de fréquences avec étalement de spectre (Frequency Hoping Spread Spectrum)

Classes

Classe	Puissance Maximale	Portée
1	100 mW	100 m
2	2,5 mW	15-20 m
3	1 mW	10 m

Topologie


Esclave
Maître
Esclave
Maître
Piconet 2
Maître du
piconet 3 et
esclave du
piconet 2
Piconet 3
Esclave du piconet 1
et du piconet 2
Esclave
Esclave
Piconet 1
Esclave
Terminaux maîtres
Terminaux esclaves

Bluetooth (JSR 82)

- API d'accès aux périphériques Bluetooth
  - Découvrir des peripheriques BT
  - S'enregistrer auprès de périphériques BT
  - Etablir une connexion BT, et échanger des données
- Package javax.bluetooth
  - Core bluetooth API
- Package javax.obex
  - OBject Exchange: standard d'échange de données entre deux points.
- Programmation d'une application Bluetooth
  - Initialisation stack BT
  - Device management
  - Device discovery
  - Service discovery
  - Communication (OBEX, RFCOMM, L2CAP-)

Logical Link Control and Adaptation Protocol

Bluetooth (JSR 82)

- Récupérer une instance du terminal dans lequel la Midlet est en cours d'exécution.
  - LocalDevice bt = LocalDevice.getLocalDevice();
  - En cas d'erreur l'exception : BluetoothStateException est déclenchée.

Découverte des périphériques et des services Bluetooth

- Deux modes de visibilité du terminal :
  - Mode LIAC (*Limited Inquiry Access Code)*: le terminal est visible pendant une période de temps déterminée (en général 1 minute)
    - bt.setDiscoverable(DiscoveryAgent.LIAC);
  - Mode GIAC (General *Inquiry Access Code)* :
    - bt.setDiscoverable(DiscoveryAgent.GIAC);
- Rendre:
  - Pour rendre le terminal non détectable:
    - bt.setDiscoverable(DiscoveryAgent.NOT_DISCOVERABLE);

Recherche de périphériques

- Etape 1: créer une classe (l'écouteur) qui implémente DiscoveryListener
  - L'interface DiscoveryListener définit 2 méthodes pour la découverte de périphériques
    - deviceDiscovered() : appelée lors de chaque découverte d'un périphérique, le périphérique est passé à la méthode comme un objet de type RemoteDevice.
    - inquiryCompleted() : invoquée lorsque la recherche de périphériques est terminée
- Etape 2: Récupérer l'agent de découverte lié à votre terminal bluetooth.
- Etape 3: Démarrer la recherche par appel de la méthode `da.startInquiry()`

```
//Etape 1 Soit dl une classe qui implémente DiscoveryListener

LocalDevice bt = LocalDevice. getLocalDevice();
//Etape 2
DiscoveryAgent da = bt.getDiscoveryAgent();
Etape 3
boolean succes = da.startInquiry(DiscoveryAgent. GIAC, dl);
```

Recherche de services

- Chaque service est identifié par un nombre hexadécimal de 128 bits unique (uuid: universally unique identifier)

```
UUID[] uuidListe = {uuid,…};
//RemoteDevice rd; récupéré lors de la découverte du
périphérique
da.searchServices( null, uuidListe, rd, dl);
```

- La méthode searchServices recherche si les services mentionnés dans la liste uuidListe sont supportés dans le périphérique rd.
- Le premier paramètre est un tableau d'entiers qui identifient les attributs qu'on veut récupérer sur le service (exemple:0x100 représente le nom du service).
- L'interface DiscoveryListener définit 2 méthodes pour la découverte de services:
  - servicesDiscovered() : appelée lors de la découverte des services, les services sont retournés comme un objet de type ServiceRecord[]
  - serviceSearchCompleted() :appelée lorsque la recherche des services est terminée
- La méthode getConnectionURL() de l'objet ServiceRecord retourne la chaîne de connexion au service.

- Remarque:
  - Les périphériques connus du terminal ou bien disponibles dans le cache, peuvent être récupérés de la manière suivante:
    - da.retrieveDevices(da.PREKNOWN ou da.CACHED)
    - da.selectService(uuid):retourne la chaîne de connexion correspondante au service.

Connexion d’un client

- L’API Bluetooth supporte deux types de connexion:
  - L2CAP: similaire aux datagrammes IP
    - Chaîne de connexion: btl2cap://
  - RFCOMM: protocol basé sur les flux (similaire au protocole TCP)
    - Chaîne de connexion: btspp://

Connexion du Serveur

- l2CACP
  - Chaîne de connexion l2cap:
  - btl2cap://localhost:3B9FA89520078C303355AAA694238F08 ;name=ExempleL2CAP
  - Connector.open() retourne L2CAPConnectionNotifier (lcn).
  - lcn.acceptAndOpen() :attend les demandes de connexion et retourne une connexion de type L2CAPConnection qui peut être utilisée pour échanger des données avec le client.
- RFCOMM (ou SPP)
  - Chaîne de connexion: btspp://localhost:3B9FA89520078C303355AAA694238F08 ;name=SPPEXemple
  - Connector.open() retourne un objet de type StreamConnectionNotifier (scn).
  - Scn.acceptAndOpen() attend les connexions entrantes et retourne un objet de type StreamConnection qui peut être utilisé pour échanger des données avec le client.

D:\NetBeans\Bt1\src\BClient.java

```
import java.util.Vector;
import javax.bluetooth.*;
import javax.microedition.io.*;
import java.io.*;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author pc
 */
public class BClient implements DiscoveryListener {
//
    LocalDevice bt = null;//Gestionnaire BT
    DiscoveryAgent agent = null;/*Agent de recherche*/
    Vector periphs = null;
    ServiceRecord[] services = null;

    public void deviceDiscovered(RemoteDevice rd, DeviceClass dc) {
        periphs.addElement(rd);
    }

    public void inquiryCompleted(int i) {
        switch (i) {
            case DiscoveryListener.INQUIRY_COMPLETED:
                /*Recherche terminée normalement
                 *
                 */
                if (periphs.size() > 0) //Rechercher les services du premier périphérique découvert.
                {
                    rechercheServices((RemoteDevice) periphs.firstElement());
                }
                break;
            case DiscoveryListener.INQUIRY_ERROR:
//Erreur durant le processus de recherche
                break;
            case DiscoveryListener.INQUIRY_TERMINATED:
                /*Recherche annulée par appel de la méthode agent.cancelInquiry()
                 *
                 */
                break;
        }

    }

    public void serviceSearchCompleted(int TransId, int codeReponse) {
        switch (codeReponse) {
            case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
```

```
            /*
             * Recherche des services terminée complétement avec succès
             *
             */
            ServiceRecord sr=services[0];
           String scn= sr.getConnectionURL(TransId,true);
        try {
            StreamConnection sc = (StreamConnection) Connector.open(scn);
            DataInputStream in =sc.openDataInputStream();
DataOutputStream out =sc.openDataOutputStream();
//Echange ....
in.close();
out.close();
sc.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
                break;
            case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
// périphérique non accessible
                break;
            case DiscoveryListener.SERVICE_SEARCH_ERROR:
// Erreur
                break;
            case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
// Aucun service disponible
                break;
            case DiscoveryListener.SERVICE_SEARCH_TERMINATED:

            /*
             * Recherche annulée par appel de la méthode agent.cancelServiceSearch()
             *
             */
            break;
        }
    }

    public void servicesDiscovered(int TransId, ServiceRecord[] srs) {
        services = srs;
    }

//Démarrer la recherche de périphériques
    private void recherchePeriph() {
        try {
            bt = LocalDevice.getLocalDevice();
        } catch (BluetoothStateException bse) {
        }
        agent = bt.getDiscoveryAgent();
        periphs = new Vector();
        try {
            if (!agent.startInquiry(DiscoveryAgent.GIAC, this)) {
//Recherche non démarrée
            }
        } catch (BluetoothStateException bse) {
```

```
// Code de gestion de l'erreur
        }
    }

    private void rechercheServices(RemoteDevice rd) {
        /*
         * La recherche de service retourne les attributs par défaut:
         * ServiceRecordHandle (0x0000), ServiceClassIDList (0x0001),
         * ServiceRecordState (0x0002), ServiceID (0x0003) et
         * ProtocolDescriptorList (0x004).
         *
         * nous voulons aussi répupérer les attributs suivants; ServiceName (0x100),
         * ServiceDescription (0x101) et ProviderName (0x102).
         *
         *
         */

        int[] attributs = {0x100, 0x101, 0x102};

        UUID[] uuids = new UUID[1];
// l'id 0x1002 représente tous les services publiques
        uuids[0] = new UUID(0x1002);
        try {
            //retourne un numéro de transcation
            agent.searchServices(attributs, uuids, rd, this);
        } catch (BluetoothStateException e) {
//Gestion des erreurs
        }
    }
}
```

D:\NetBeans\Bt1\src\BServeur.java

```
import java.io.*;
import java.io.IOException;
import javax.bluetooth.*;
import javax.microedition.io.*;


public class BServeur {

   LocalDevice local = null;
StreamConnectionNotifier server = null;
StreamConnection conn = null;
String connectionURL =
"btspp://localhost:393a84ee7cd111d89527000bdb544cb1;"
+ "authenticate=false;encrypt=false;name=RFCOMM Server";
/*
* Rendre le terminal visible
*
*/
public BServeur(){
try {
local = LocalDevice.getLocalDevice();
local.setDiscoverable(DiscoveryAgent.GIAC);
} catch (BluetoothStateException e) {
//

}
}
public void demarrer(){
try {
server = (StreamConnectionNotifier)
Connector.open(connectionURL);
} catch (IOException e1) {

}

try {
conn = server.acceptAndOpen();
DataInputStream in =conn.openDataInputStream();
DataOutputStream out =conn.openDataOutputStream();

//....
in.close();
out.close();
conn.close();
} catch (IOException e2) {
// Error handling code here
}

}
}
```