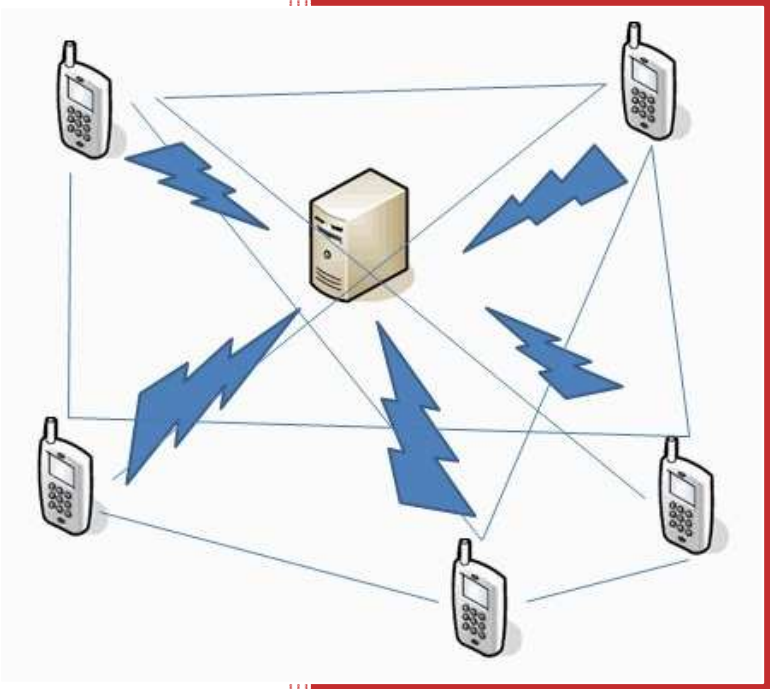
Sistema P2P ibrido per il File Sharing tra dispositivi Android



Gabriele Falace (127599)

Griglie e Sistemi di Elaborazione Ubiqui

A.A. 2009/2010

Sommario

Indice delle figure

1. Descrizione generale del progetto

L'obiettivo del progetto è la progettazione e realizzazione di un sistema peer to peer ibrido per il file sharing che consenta ai partecipanti di effettuare ricerche sui file che ciascun partecipante possiede sulla propria SD card. I partecipanti possono inoltre verificare che l'utente proprietario del file di interesse sia raggiungibile tramite Bluetooth. Tale funzione non è però funzionante sull'emulatore, in quanto questo non fornisce alcun supporto al Bluetooth.

Un componente server gestisce la volatilità del sistema: infatti in un sistema mobile immerso in un ambiente come ad esempio un parco o una piazza ha la proprietà di variare molto spesso in termini di utenti che lasciano o entrano nella rete. Il server ha però solo il ruolo di Discovery Server, in quanto è utilizzato dagli altri peer solo in fase di accesso alla rete e per aggiornare la propria tabella dei peer.

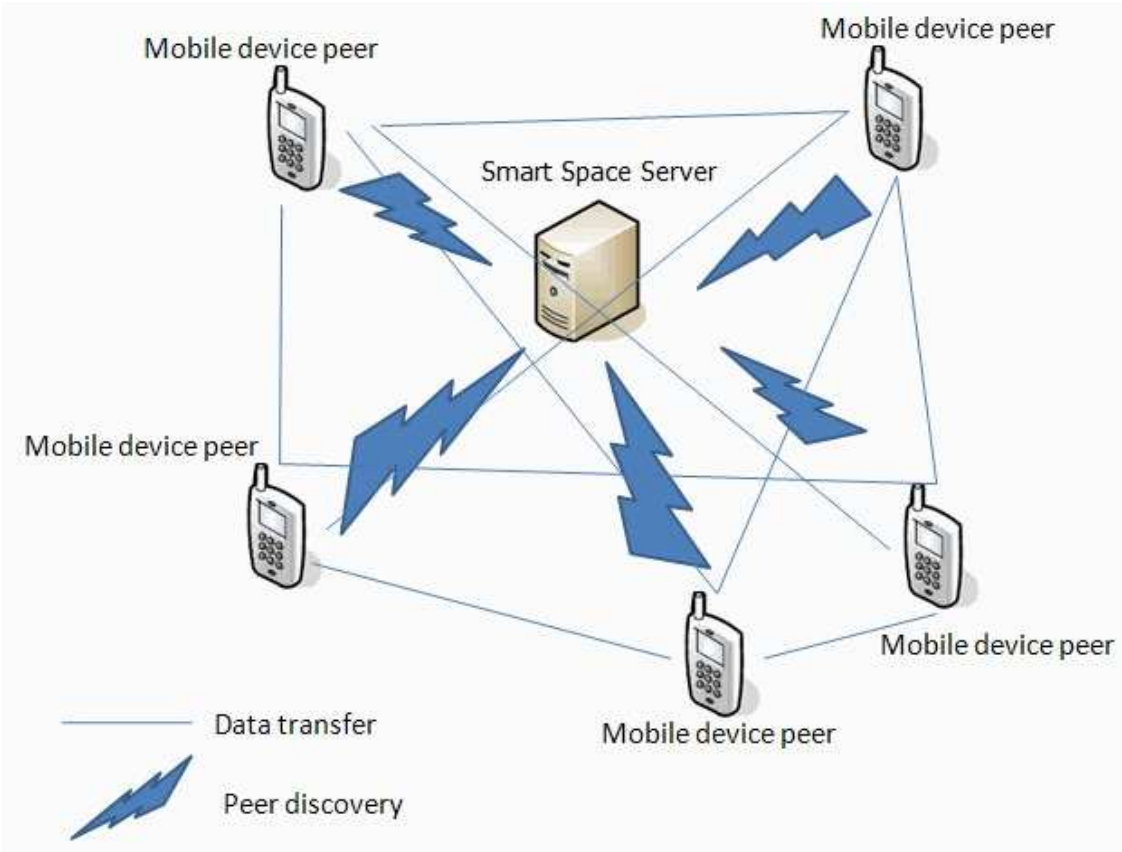


Figura 1-1 Vista generale dell'applicazione

2. Modello comportamentale del sistema

In questa sezione viene introdotto il modello concettuale dell'applicazione che porta – in seguito – alla definizione di un più dettagliato modello progettuale.

Di seguito è mostrato il diagramma di attività che riferisce le principali operazioni svolte dall'applicazione nella fase di avvio, comprendenti i setup di rete e le comunicazioni con il Discovery Server.

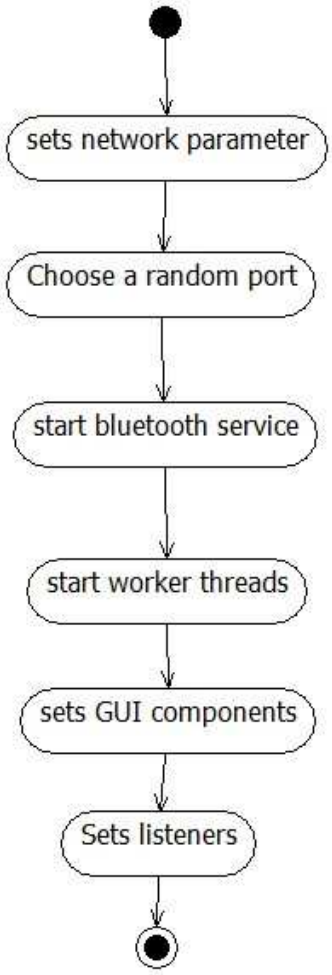


Figura 2-1 Avvio dell'applicazione

Nelle successive 3 sotto-sezioni sono invece presentati rispettivamente i thread attivati dall'applicazione, i listener designati a rispondere alle sollecitazioni dell'utente e i protocolli che definiscono le interazioni tra il dispositivo mobile ed il server, nonché le interazioni tra diversi dispositivi mobile.

Threads

L'applicazione fa uso di diversi background worker thread, al fine di ottenere una maggiore efficienza: infatti, contrariamente ai *Service*, i thread attivati da una *Activity* sono eseguiti separatamente dal Runtime e questo consente una esecuzione più fluida.

I principali thread presenti nell'applicazione mobile sono:

FileSearcher

Mostrato in Figura 2-2, è un thread creato dal listener del bottone di ricerca; esso riceve come input l'id di un peer target ed ha lo scopo di chiedere (a tale peer target) i file nei cui nomi vi siano occorrenze del termine di ricerca. Una volta ottenuti l'elenco di tali file, lo ritorna al listener.

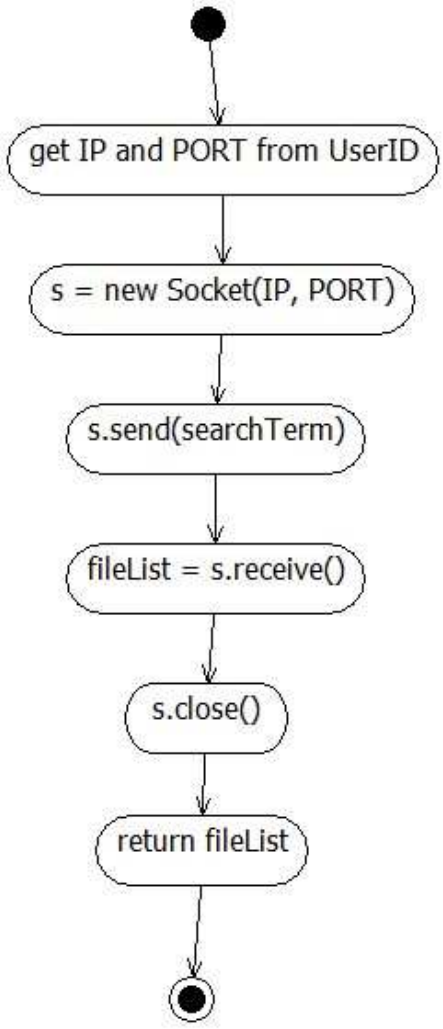


Figura 2-2 Attività del thread FileSearcher

SearchResponderAgent

Il SearchResponderAgent (mostrato in Figura 2-3) è il thread responsabile per la risposta di un dispositivo ad una ricerca di file da parte di tutti gli altri dispositivi. Si tratta di un server multi-thread che attiva un thread per ogni richiesta che riceve in modo da non bloccarsi in modo sequenziale;

I thread avviati sono gestiti con un Thread Pool di dimensione fissata e pari a 5, in modo da non servire troppe richieste concorrentemente. Limitare la concorrenza è importante al fine di non rallentare troppo l'applicazione e ridurre il consumo della batteria. Il thread è predisposto alla possibilità di effettuare un controllo sul livello di batteria per decidere se servire o meno una richiesta: se la batteria è sotto una determinata soglia, il thread decide di attendere 60 secondi la terminazione dei "SingleRequestManager" avviati e – trascorso questo tempo – li eventualmente termina bruscamente.

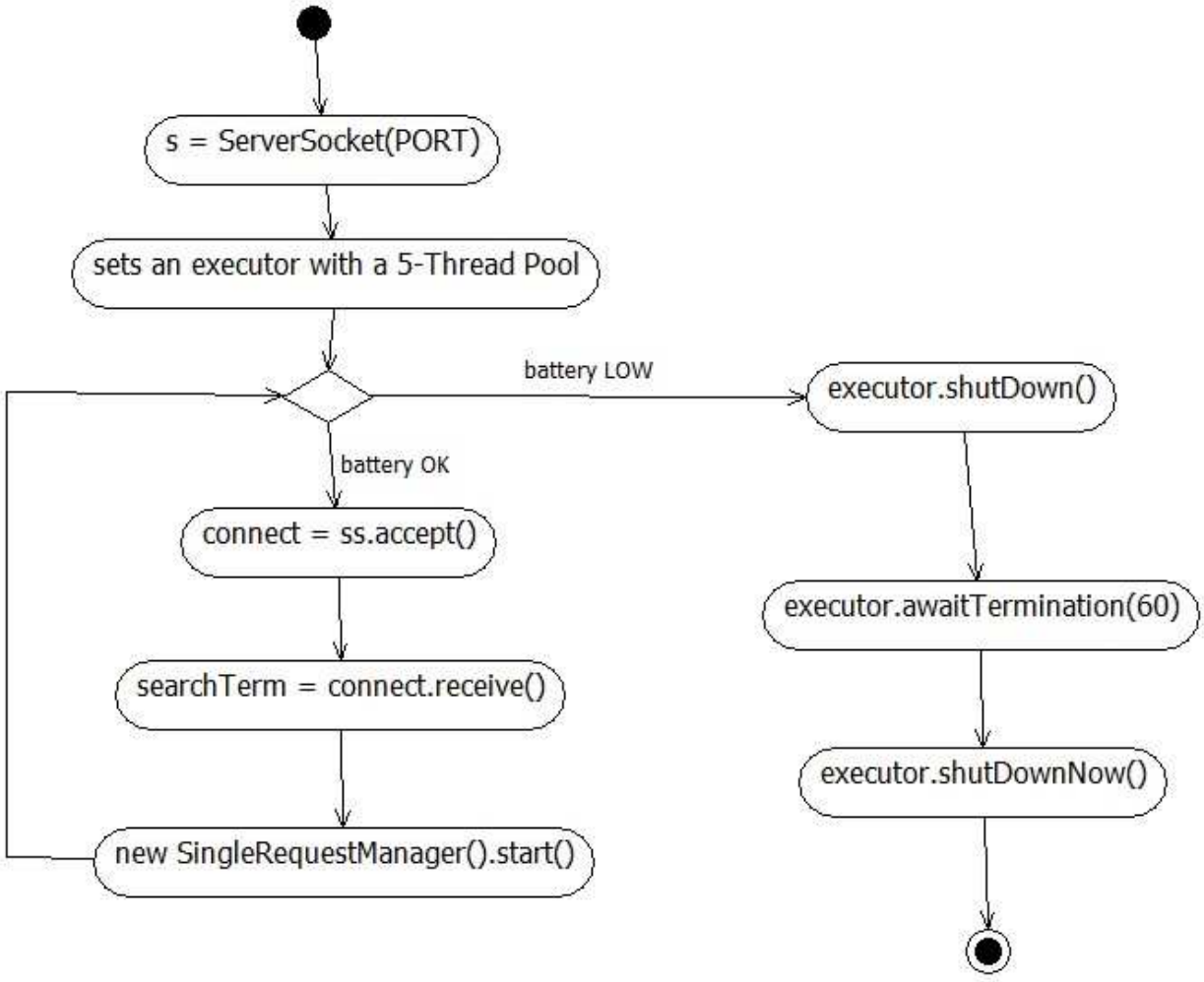


Figura 2-3 Attività del SearchResponderAgent

SingleRequestManager

Avviato dal SearchResponderAgent, consente di rispondere ad una singola richiesta di file da parte di un peer;

Questo thread (mostrato in Figura 2-4) accede alla SD Card del dispositivo per recuperare i file rilevanti per la ricerca. Un file è ritenuto rilevante se il titolo contiene come sottostringa il termine di ricerca inviato dal dispositivo richiedente. Una volta finita la ricerca il thread invia i risultati tramite la connessione al richiedente e termina.

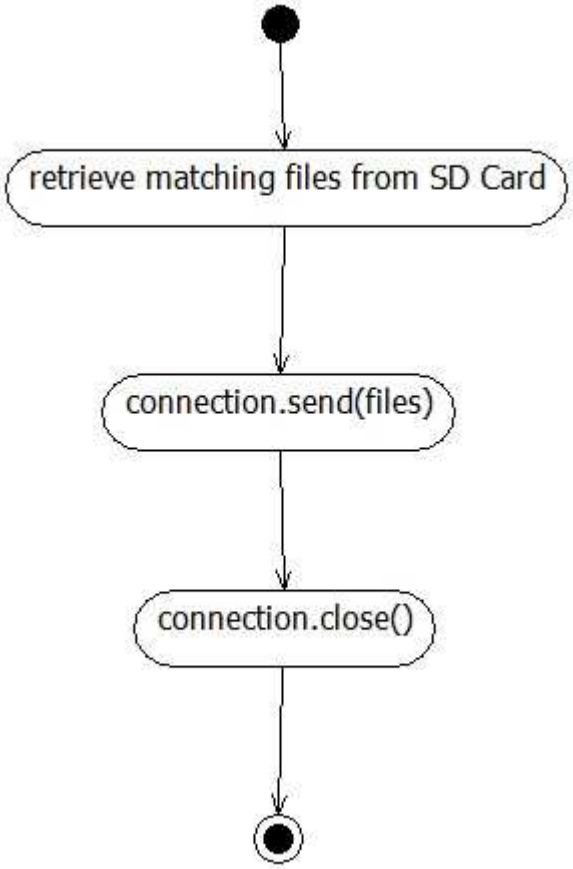


Figura 2-4 Attività del SingleRequestManager

UserAgent

È il thread lato mobile che si occupa di gestire lo stato complessivo del sistema tramite messaggi di PING e PEER_UPDATE inviati periodicamente al Discovery Server. Inoltre tale thread (mostrato in Figura 2-5) effettua tutte le comunicazioni di credenziali al server al momento della connessione.

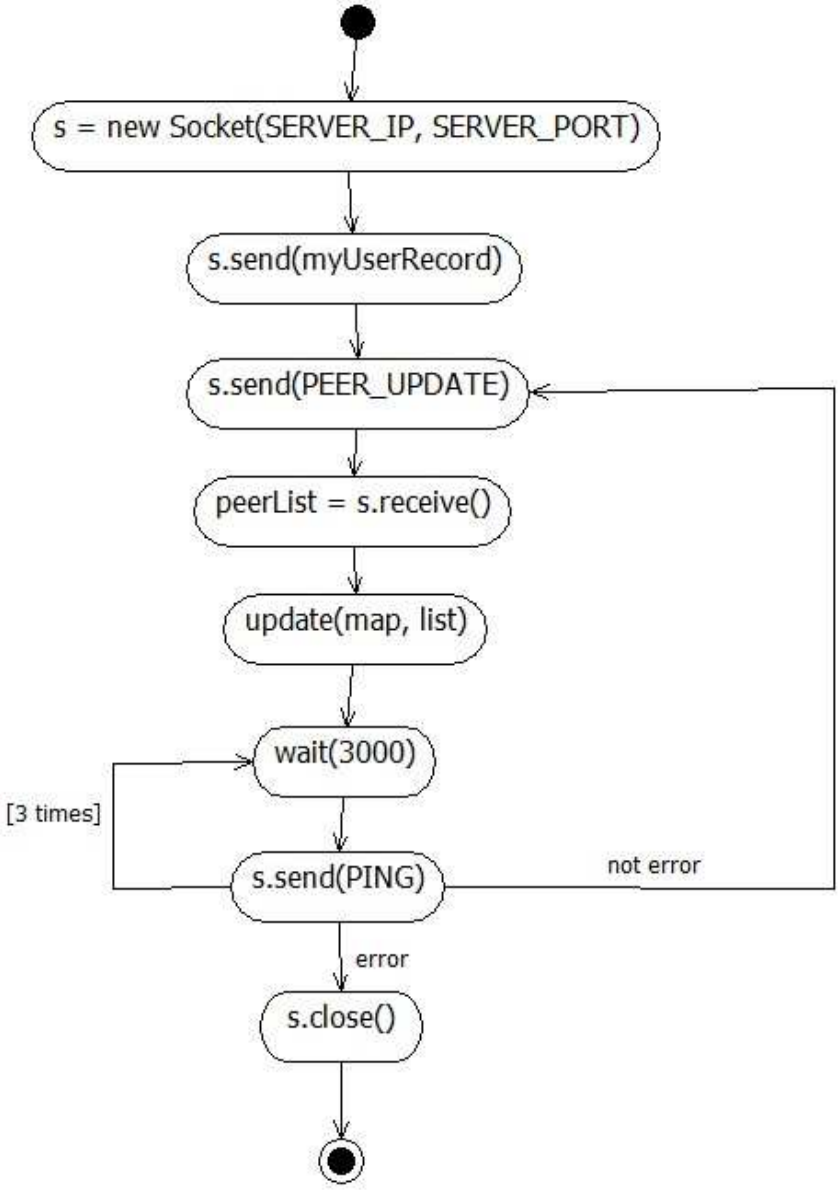


Figura 2-5 Attività del thread UserAgent

UserManager

Lato Discovery Server, invece, vi è un unico processo Server che avvia un nuovo thread *UserManager* per ogni richiesta di connessione.

Tale thread riceve tutti i messaggi del dispositivo, che possono essere di due tipi:

1. PING
   Sono messaggi inviati al fine di segnalare la propria presenza al server: ogni 3 secondi un messaggio di PING viene inviato da ogni UserAgent al proprio UserManager press oil server. Questo si occupa di aggiornare la tabella dei peer con la timestamp del peer che ha inviato il PING. Qualora il dispositivo si disconnetta, il thread UserManager rilevando la disconnessione, rimuove il peer dalla tabella dei peer e termina per liberare le risorse presso il Discovery Server.

2. PEER_UPDATE
   Sono inviati al fine di richiedere al server l'elenco aggiornato dei peer attualmente connessi al sistema. Dato che tale messaggio richiede al server l'invio di una struttura dati serializzata, esso viene inviato ogni 12 secondi, meno di frequente rispetto ai messaggi PING.

Funzioni e listeners

Nell'applicazione sono definiti 3 listeners

1. Login Listener
2. SearchListener
3. BluetoothCheckListener

Login

Il *Login Listener* (in realtà è un listener anonimo agganciato direttamente al bottone di login) ha la responsabilità di far scegliere all'utente il nome logico da usare durante la sessione applicativa. Una volta che l'utente ha digitato un nome e premuto il bottone, il listener avvia l'Activity (android) principale dell'applicazione passando ad essa il nome logico come "extra" nell'Intent utilizzato. La relazione tra le due Activity è mostrata di seguito, in Figura 2-6.

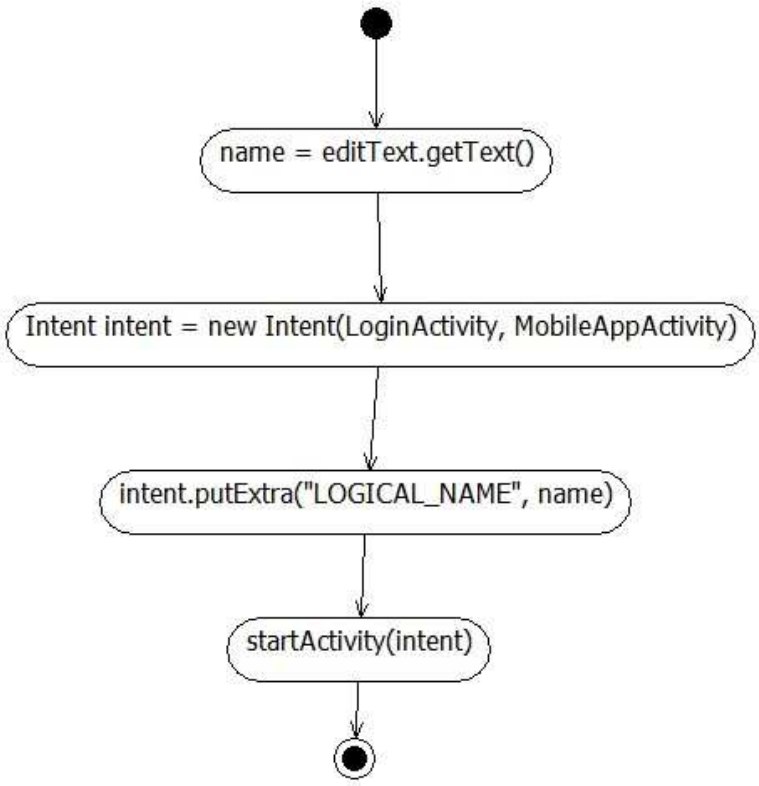


Figura 2-6 Attività della Login Activity

Ricerca file

Il *SearchListener* definisce la logica di ricerca dei file nei peer attualmente conosciuti dal dispositivo. Una volta inserito il searchTerm nella casella di testo e premuto il bottone che avvia la ricerca, viene istanziato un *Thread Pool* a taglia non prefissata. Questo è un aspetto importante al fine di effettuare la ricerca in parallelo, in quanto il dispositivo non può conoscere a priori il numero di altri peer che formeranno con lui la rete: è un aspetto dinamico. In particolare, il listener:

- crea una task-list di Callable, con un FileSearcher per ogni peer nella sua tabella;
- utilizza un Executor (con il thread pool variabile) per avviare i suddetti thread;
- Mediante una lista di Future raccoglie i risultati dei FileSearcher;
- Aggiorna la GUI mediante l'adapter.

Controllo di disponibilità con bluetooth

Il *BluetoothCheckListener* consente all'utente di verificare se il proprietario di un dato file presente nella lista dei risultati è raggiungibile mediante Bluetooth. Al fine di effettuare tale verifica l'applicazione attiva (*onCreate*) l'interfaccia bluetooth del dispositivo per rendersi visibile. Dopo aver effettuato la verifica il listener informa l'utente della raggiungibilità o meno del peer. Quando l'applicazione viene chiusa (*onDestroy*) il bluetooth viene spento. Lo stato della connessione bluetooth relativamente allo stato dell'applicazione è mostrato in Figura 2-7.

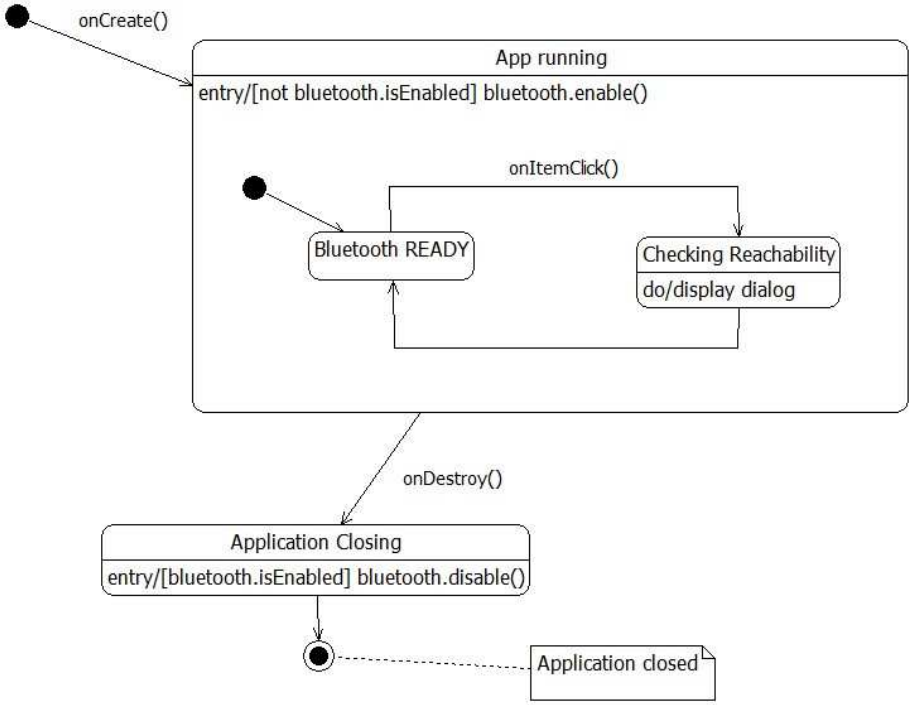


Figura 2-7 Statechart della gestione del Bluetooth

Protocolli ed interazioni

I principali protocolli utilizzati nell'applicazione sono:

1. La registrazione (*Registration*) di un nuovo dispositivo che entra nella rete;
2. La comunicazione tra dispositivo utente e Discovery Server per il mantenimento dello stato della rete (*Presence Management*);
3. La ricerca dei file (*File Search*) che coinvolge diversi componenti sul dispositivo che avvia la ricerca e sul dispositivo che risponde.

Nel seguito i tre protocolli sono illustrati mediante dei diagrammi di sequenza UML.

Registration e Presence Management

La registrazione ha luogo quando l'utente effettua la login; in tal caso il thread UserAgent invia un messaggio al server con le credenziali dell'utente:

- indirizzo IP;
- port TCP su cui si terrà in ascolto per ricevere le comunicazioni da parte degli altri dispositivi peer;
- nome logico scelto dall'utente.

Il server istanzia un thread UserManager per gestire tutta la comunicazione con il nodo mobile in ingresso e tale thread rimane attivo finché rimane attiva l'applicazione nel dispositivo mobile.

La gestione della presenza nel sistema è effettuato mediante l'invio periodico di informazioni tra i dispositivi mobili ed il server di discovery (mostrato in Figura 2-8). Il Discovery Server ha il compito di mantenere una tabella aggiornata dei peer presenti nel sistema. I peer vengono aggiunti alla tabella in fase di registrazione e sono rimossi qualora l'interruzione delle comunicazioni con il rispettivo UserManager causa la terminazione di quest'ultimo.

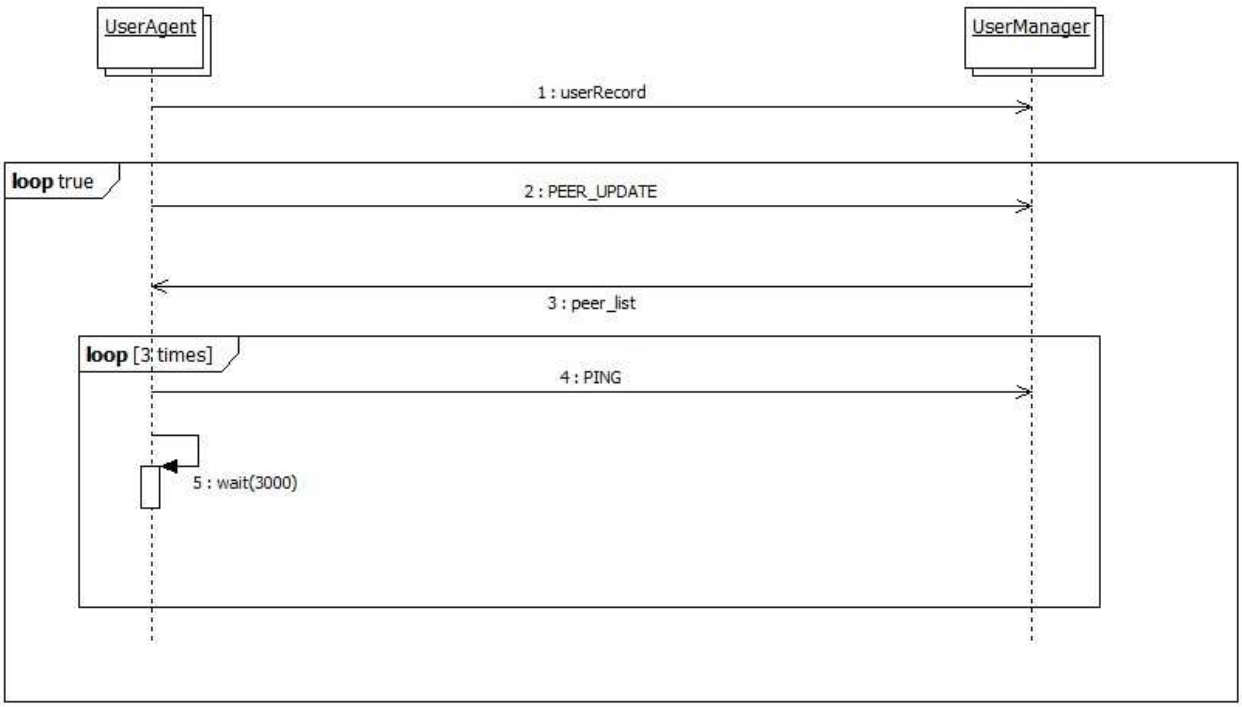


Figura 2-8 Interazione tra dispositivo mobile e UserManager

File Search

La ricerca dei file nei peer conosciuti è una interazione che coinvolge dal lato "richiedente" il listener della funzione di ricerca e dal lato "rispondente" il thread SearchResponderAgent. Questo, attivato nell'avvio dell'applicazione, resta in attesa (sulla porta specificata in fase di registrazione) di ricevere le richieste di ricerca file con lo scopo di rispondere. Per servire più richiedenti concorrentemente il SearchResponderAgent attiva a sua volta un SingleRequestManager il quale – accedendo alla SD Card del dispositivo – recupera i file rilevanti e li invia direttamente al FileSearcher verso il quale possedeva la connessione. Il seguente diagramma di sequenza (Figura 2-9) mostra le interazioni che vengono compiute per la risoluzione di una ricerca.

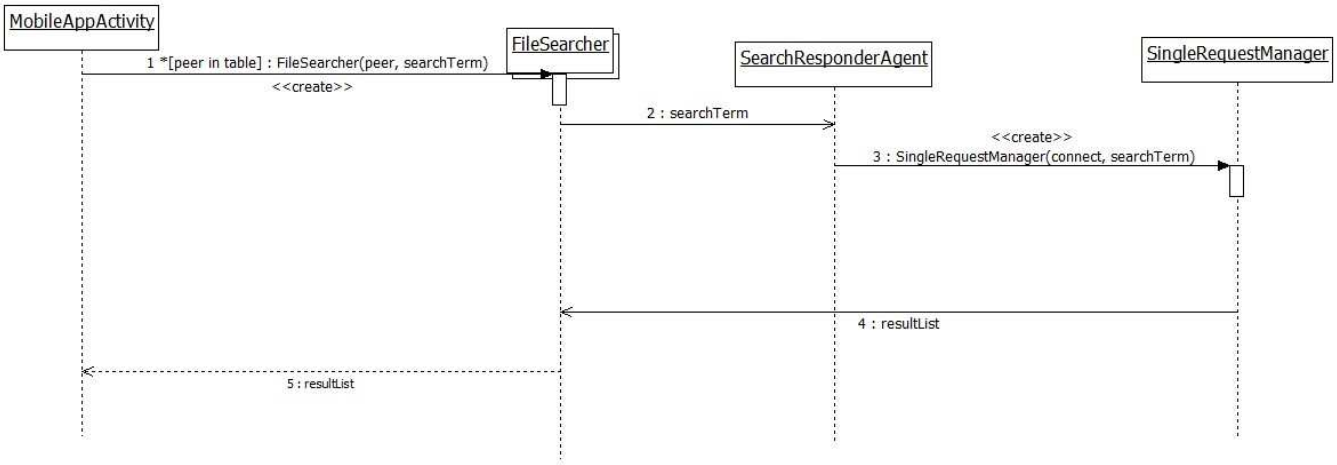


Figura 2-9 Interazione per la ricerca dei file tra peer

3. Struttura e componenti dell'applicazione

La struttura dell'applicazione è esplicata in termini di diagrammi di deployment – utili al fine di comprendere quali componenti software sono eseguiti su determinati nodi – e diagrammi di classe aventi lo scopo di illustrare la struttura interna delle classi che costituiscono l'applicazione.

Deployment dei componenti

L'implementazione dell'architettura software dell'applicazione consiste di

1. un componente "server" che cerca di mantenere una visione consistente dei peer attualmente connessi. Tale server, inoltre, aggiorna una GUI ogniqualvolta riceve un evento.
2. dai diversi "mobili" che si connettono dando vita alle interazioni peer to peer vere e proprie

Il seguente deployment diagram (Figura 3-1) mostra i tipi di nodi coinvolti nel sistema, mettendo in evidenza che presso il nodo mobile è presente il file "AndroidManifest.xml" utile a dichiarare i componenti dell'applicazione nonché i permessi per consentire l'accesso alla rete all'applicazione.

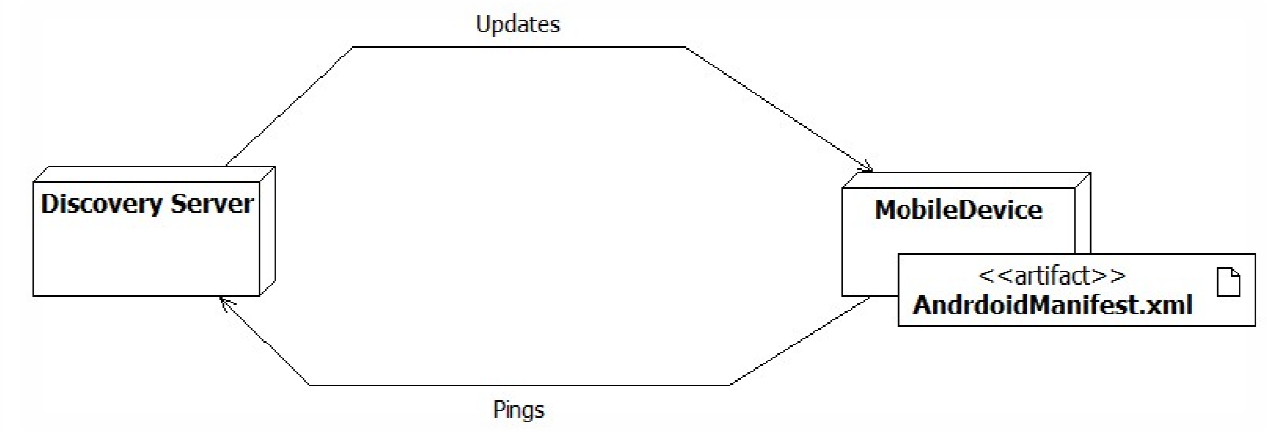


Figura 3-1 Diagramma di deployment mobile-server

I permessi di cui fa uso l'applicazione sono:

- INTERNET
  Per consentire all'applicazione l'accesso alla rete;

- BLUETOOTH
  Per consentire l'accensione del bluetooth;

- BLUETOOTH_ADMIN
  Per consentire la scansione di ricerca di dispositivi.

Il seguente deployment diagram (Figura 3-2) mostra le interazioni tra istanze di nodi dello stesso tipo, cioè dispositivi android.

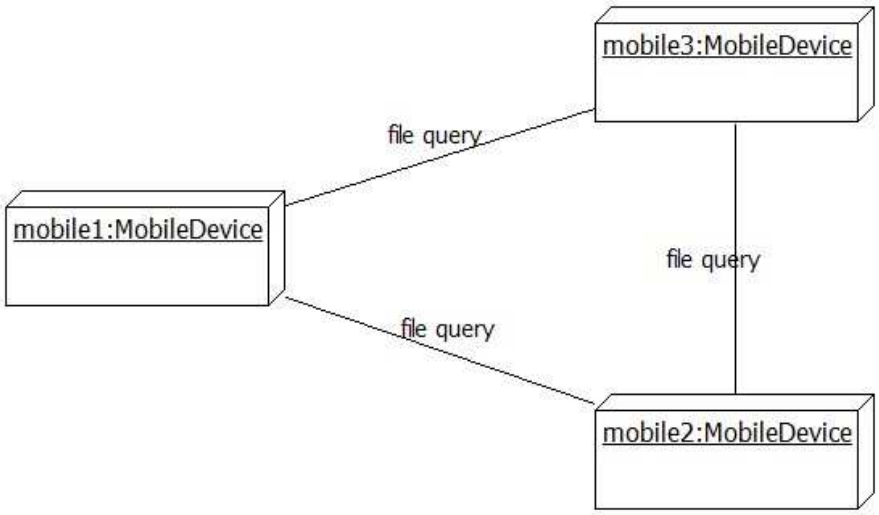


Figura 3-2 Diagramma di Deployment che mostra le interazioni tra peer

Classi dell'applicazione

Le classi più importanti che definiscono l'applicazione sono mostrate nei seguenti diagrammi di classe. In particolare, la Figura 3-3 mostra la relazione tra la principale Activity Android e diversi "background worker thread" che esplicano diverse funzioni nell'applicazione, in particolare la funzione di "ping" verso il Discovery Server e la funzione di risposta alle query degli altri peer.

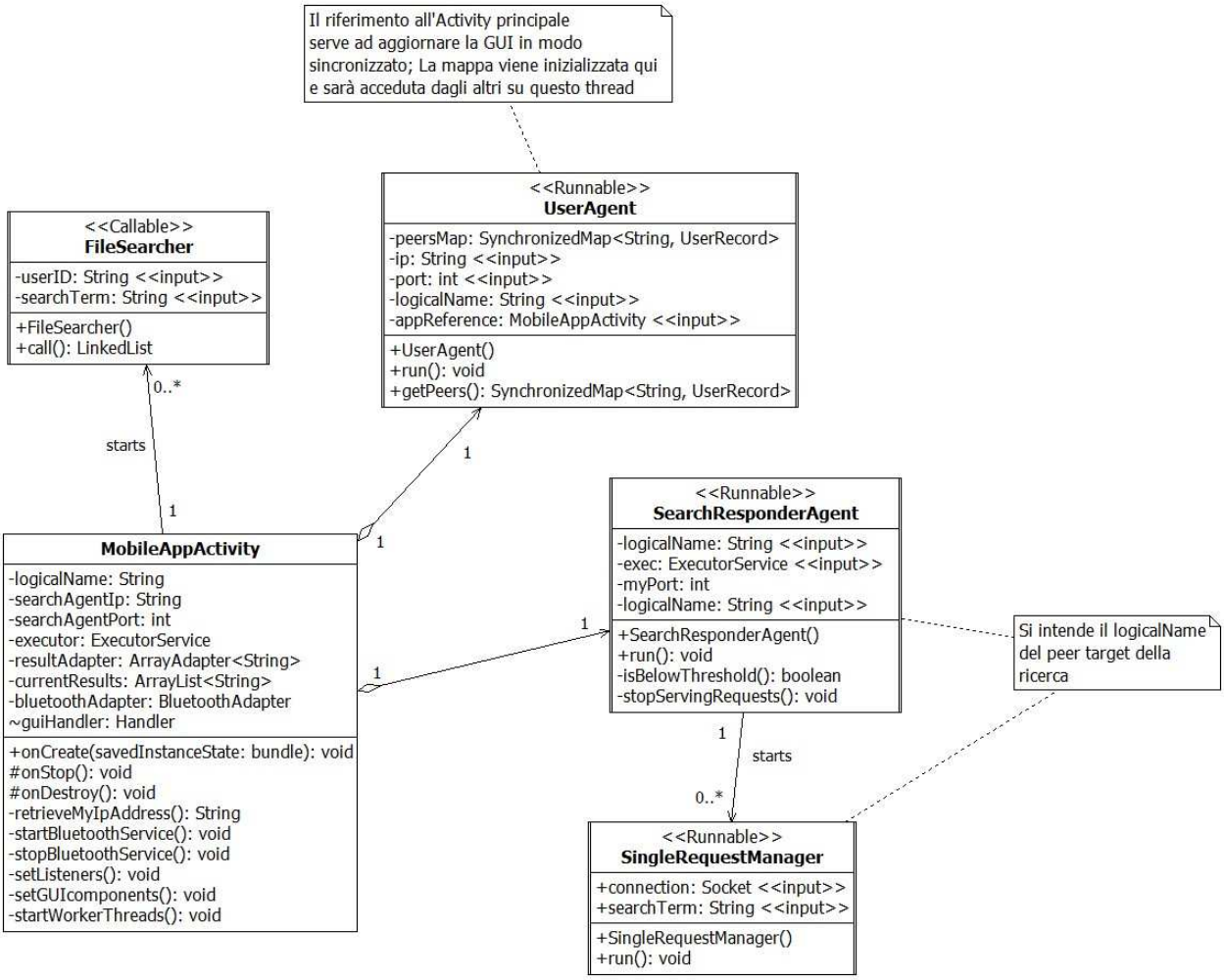


Figura 3-3 Class diagram dei principali thread sul dispositivo android

La Figura 3-4 mostra, invece, la relazione tra la MobileAppActivity e i due ascoltatori: quello della ricerca e quello della raggiungibilità tramite bluetooth. Inoltre è anche mostrato l'oggetto TextViewMessage che consente di sincronizzare la GUI con un thread di background, consentendo ad esso di aggiornare la GUI in modo thread-safe.

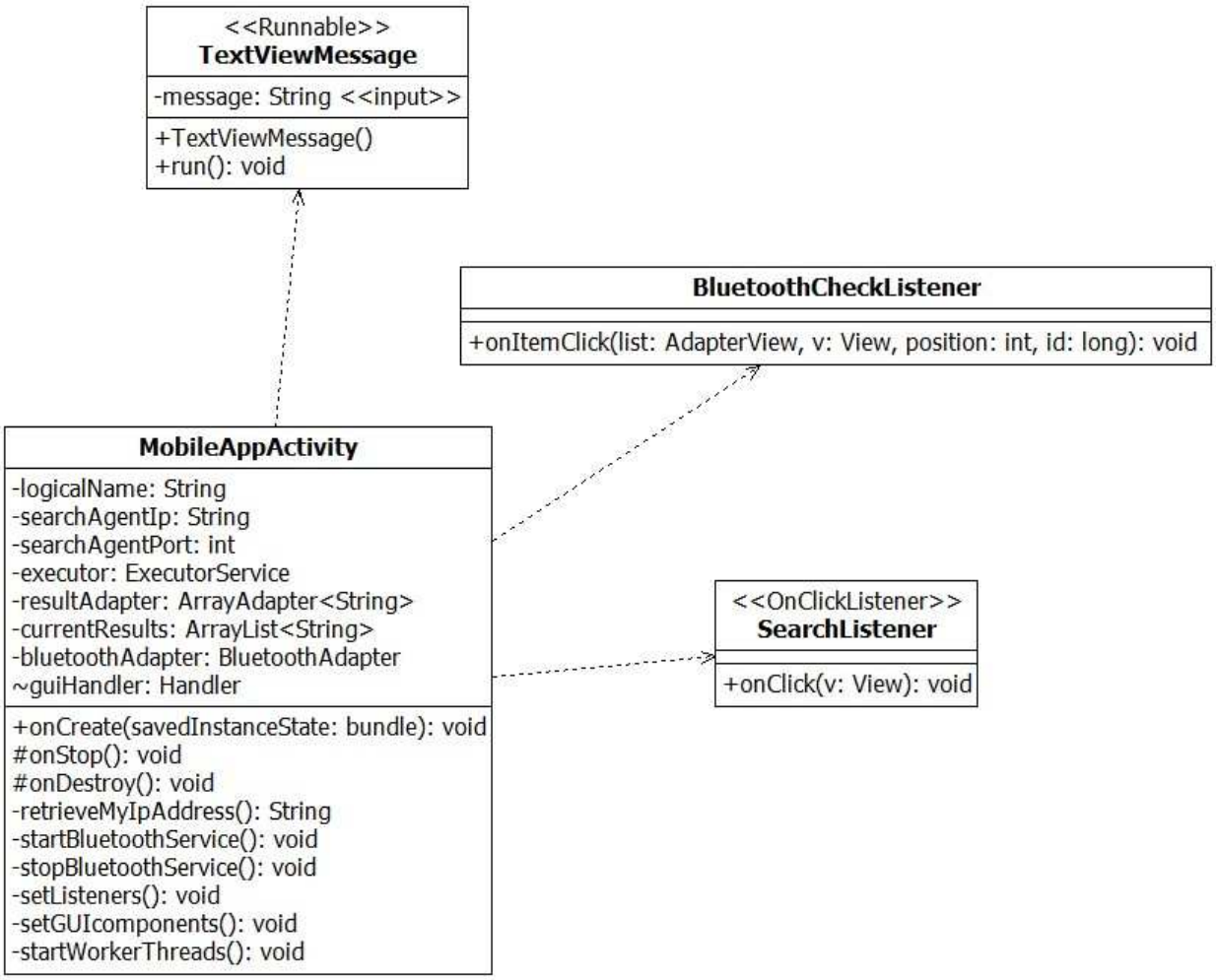


Figura 3-4 Class diagram dei principali ascoltatori dell'applicazione

Nella Figura 3-5 è mostrata la struttura delle classi del Discovery Server e dello UserManager utilizzato da esso per tenere sotto controllo la dinamica del sistema.

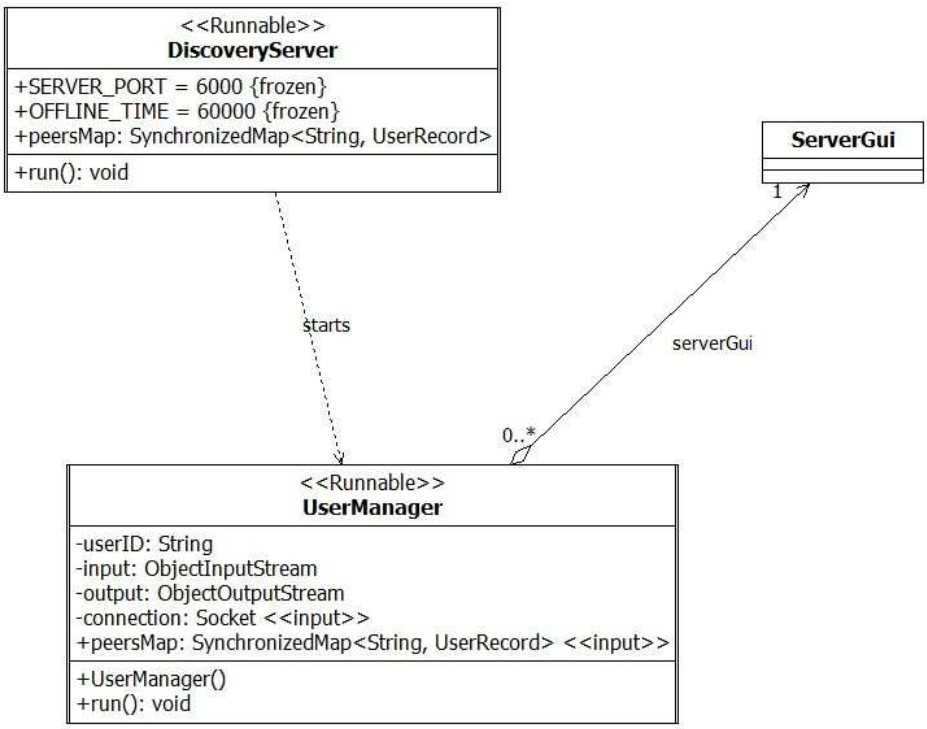


Figura 3-5 Class diagram del Discovery Server

4. Codice dell'applicazione

Il codice dell'applicazione è composto sostanzialmente da file XML e file Java.

File XML

File AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="mobile.filesharing"
     android:versionCode="1"
     android:versionName="1.0">

   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MobileAppActivity"
              android:label="@string/app_name">
              </activity>
      <activity android:name=".LoginActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>

      </activity>
   </application>

        <uses-permission
              android:name="android.permission.INTERNET">
        </uses-permission>
        <uses-permission
              android:name="android.permission.BLUETOOTH">
        </uses-permission>
        <uses-permission
              android:name="android.permission.BLUETOOTH_ADMIN">
        </uses-permission>
</manifest>
```

File list-item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<mobile.filesharing.ListItemView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dp"
  android:scrollbars="vertical"
  android:textColor="@color/text"
  android:fadingEdge="vertical"
/>
```

File login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">

<EditText
        android:id="@+id/name_field"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
/>

<Button
        android:id="@+id/login_button"
        android:text="Login"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
/>
</LinearLayout>
```

File mobile_app.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">

  <TextView
        android:id="@+id/connection_status"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
  />

  <EditText
        android:id="@+id/search_field"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
 />

  <Button
        android:id="@+id/search_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Search"
  />

  <ListView
        android:id="@+id/result_list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
  />

</TableLayout>
```

File Java

I file Java che costituiscono il core dell'applicazione sono riportati di seguito.

File UserReecord.java

```
package common;

import java.io.Serializable;

public class UserRecord implements Serializable {

	private String ip;

	private int port;

	private long lastTimestamp;

	private String logicalName;

	public UserRecord(){}

	public UserRecord(String ip, int port, String logicalName){
		this.ip = ip;
		this.port = port;
		this.logicalName = logicalName;
	}

	public void setIp(String ip) {
		this.ip = ip;
	}

	public String getIp() {
		return ip;
	}

	public void setPort(int port) {
		this.port = port;
	}

	public int getPort() {
		return port;
	}

	public void setLastTimestamp(long lastTimestamp) {
		this.lastTimestamp = lastTimestamp;
	}

	public long getLastTimestamp() {
		return lastTimestamp;
	}

	public void setLogicalName(String logicalName) {
		this.logicalName = logicalName;
	}

	public String getLogicalName() {
		return logicalName;
	}

}
```

File Message.java

```
package common;

import java.io.Serializable;

public class Message implements Serializable {

	private String messageType;

	public Message(String type){
		setMessageType(type);
	}

	public void setMessageType(String messageType) {
		this.messageType = messageType;
	}

	public String getMessageType() {
		return messageType;
	}
}
```

File SynchronizedMap.java

```
package common;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.Serializable;
import java.util.Collection;
import java.util.HashMap;
import java.util.Set;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

public class SynchronizedMap<K, V> implements Serializable {

	private HashMap<K, V> map;

	private transient ReadWriteLock rw_lock;
	private transient Lock readLock;
	private transient Lock writeLock;

	public SynchronizedMap(){
		map = new HashMap<K, V>();
		rw_lock = new ReentrantReadWriteLock();

		readLock = rw_lock.readLock();
		writeLock = rw_lock.writeLock();
	}

	public void put(K key, V value){
		writeLock.lock();
		map.put(key, value);
		writeLock.unlock();
	}

	public V get(K key){
		readLock.lock();
		V value = map.get(key);
```

```
            readLock.unlock();
            return value;
        }

        public int size(){
            readLock.lock();
            int size = map.size();
            readLock.unlock();
            return size;
        }

        public boolean containsKey(K key){
            readLock.lock();
            boolean res = map.containsKey(key);
            readLock.unlock();
            return res;
        }

        public boolean containsValue(V value){
            readLock.lock();
            boolean res = map.containsValue(value);
            readLock.unlock();
            return res;
        }

        public void remove(K key){
            writeLock.lock();
            map.remove(key);
            writeLock.unlock();
        }

        private void readObject(ObjectInputStream s) throws IOException,
                                                    ClassNotFoundException {
            s.defaultReadObject();
            rw_lock = new ReentrantReadWriteLock();
            readLock = rw_lock.readLock();
            writeLock = rw_lock.writeLock();
        }

        public Collection<V> getValues(){
            readLock.lock();
            Collection<V> values = map.values();
            readLock.unlock();
            return values;
        }

        public Set<K> getKeys(){
            readLock.lock();
            Set<K> keys = map.keySet();
            readLock.unlock();
            return keys;
        }
}
```

File ListItemView.java

```
package mobile.filesharing;

import android.content.Context;
import android.content.res.Resources;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.widget.TextView;

public class ListItemView extends TextView {

	private Paint marginPaint;
	private Paint linePaint;
	private float margin;

	public void init(){
		Resources myResources = getResources();

		marginPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
		marginPaint.setColor(myResources.getColor(R.color.margin));

		linePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
		linePaint.setColor(myResources.getColor(R.color.lines));

		margin = myResources.getDimension(R.dimen.margin);
	}

	public ListItemView(Context context, AttributeSet as, int ds) {
		super(context, as, ds);
		init();
	}

	public ListItemView(Context context, AttributeSet as) {
		super(context, as);
		init();
	}

	public ListItemView(Context context) {
		super(context);
		init();
	}

	@Override
	public void onDraw(Canvas canvas){
		canvas.save();
		canvas.translate(margin, 0);

		super.onDraw(canvas);
		canvas.restore();
	}

}
```

File LoginActivity.java

```
package mobile.filesharing;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class LoginActivity extends Activity {



	public void onCreate(Bundle savedInstanceState){
		super.onCreate(savedInstanceState);
		setContentView(R.layout.login);
		Button login = (Button)findViewById(R.id.login_button);
		final EditText field = (EditText)findViewById(R.id.name_field);

		login.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				String logicalName = field.getText().toString();
				if(logicalName==null || logicalName.equals(""))
					return;
				Intent intent = new Intent(LoginActivity.this,
									MobileAppActivity.class);
				intent.putExtra("LOGICAL_NAME", logicalName);
				startActivity(intent);
			}
		});

	}
}
```

File FileSearcher.java

```
package mobile.filesharing;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.LinkedList;
import java.util.concurrent.Callable;

public class FileSearcher implements Callable<LinkedList> {

	private String userID;
	private String searchTerm;

	public FileSearcher(String userID, String searchTerm){
		this.userID = userID;
		this.searchTerm = searchTerm;
	}

	@SuppressWarnings("unchecked")
	public LinkedList call(){
		Socket connect = null;
		LinkedList searchResult = new LinkedList();
		// used to receive search results
		ObjectInputStream input = null;

		// used to send a file search
		ObjectOutputStream output = null;

		try{
			String[] tokens = userID.split(":");
			String ip = tokens[0];
			int port = Integer.parseInt(tokens[1]);
			System.out.println("Cerco di contattare IP = "+ip+" e port =
"+port);
			connect = new Socket(ip, port);
			output = new ObjectOutputStream(connect.getOutputStream());
			output.writeObject(searchTerm);
			output.flush();
			input = new ObjectInputStream(connect.getInputStream());
			searchResult = (LinkedList)input.readObject();
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			try {
				input.close();
				output.close();
				connect.close();
			}
			catch(Exception e){
				e.printStackTrace();
			}
		}
		return searchResult;
	}
}
```

File UserAgent.java

```
package mobile.filesharing;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.LinkedList;
import android.util.Log;
import common.Message;
import common.SynchronizedMap;
import common.UserRecord;



public class UserAgent implements Runnable {

	private volatile SynchronizedMap<String, UserRecord> peersMap;
	private int port;
	private String logicalName;
	private String ip;
	private MobileAppActivity appReference;

	public UserAgent(String ip, int port, String logicalName, MobileAppActivity
                                                                    reference){
		this.ip = ip;
		this.port = port;
		this.logicalName = logicalName;
		this.appReference = reference;
	}

	public SynchronizedMap<String, UserRecord> getPeers(){
		return peersMap;
	}

	@SuppressWarnings("unchecked")
	public void run(){
		Socket s = null;
		ObjectInputStream input = null;
		ObjectOutputStream output = null;

		try {
			s = new Socket("10.0.2.2", 6000);
			UserRecord record = new UserRecord(ip, port, logicalName);
			output = new ObjectOutputStream(s.getOutputStream());
			output.writeObject(record);
			output.flush();
			Log.d(logicalName+"::UserAgent", "inviato la richiesta di
                                                registrazione "+ip+":"+port);

			Thread.sleep(500);

			input = new ObjectInputStream(s.getInputStream());

			while(true){
				Message updateMsg = new Message("PEERS_UPDATE");
				output.writeObject(updateMsg);
				output.flush();

				Log.d(logicalName+"::UserAgent", "in attesa della peer-
                                                        list update");
				appReference.guiHandler.post(appReference.new
					TextViewMessage("waiting for peer-list update"));

				// Receive a list of UserRecord -> Creates a SynchroMap
```

```
                        LinkedList list = (LinkedList)input.readObject();

                        SynchronizedMap<String, UserRecord> peersMapReceived =
                                    new SynchronizedMap<String, UserRecord>();

                        appReference.guiHandler.post(appReference.new
                                  TextViewMessage("peer-list update received"));

                        for(Object iter: list){
                             UserRecord rec = (UserRecord)iter;
                             peersMapReceived.put(rec.getIp()+":"+rec.getPort(),
                                                                        rec);
                        }

                        peersMap = peersMapReceived;

                        if(peersMap != null)
                               appReference.guiHandler.post(
                                      appReference.new TextViewMessage(
                                             logicalName+" online -
                                             "+(peersMapReceived.size() - 1)+"
                                             peers connected"));
                        else
                               appReference.guiHandler.post(
                                      appReference.new TextViewMessage(
                                                    logicalName+" online - no
                                             other peers connected"));

                        for(int i=0; i<3; i++){
                          Message pingMsg = new Message("PING");
                          output.writeObject(pingMsg);
                          output.flush();
                          System.out.println("Ho inviato un messaggio di PING");
                          Thread.sleep(4000);
                        }


                        }
                  }
                  catch (Exception e) {
                        Log.e(logicalName+"::UserAgent",
                        "Errore:\n"+e.getStackTrace());
                  }
                  finally{
                        try {
                              s.close();
                              output.close();
                              input.close();
                              Log.d(logicalName+"::UserAgent", "Chiusi gli stream di
                                                                  I/O e la socket");
                        }
                        catch (IOException e) {
                              e.printStackTrace();
                        }

                  }
            }

}
```

File SingleRequestManager.java

```
package mobile.filesharing;

import java.io.File;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.LinkedList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import android.os.Environment;

public class SingleRequestManager implements Runnable {

	private Socket connection;
	private String searchTerm;
	private String logicalName;

	public SingleRequestManager(Socket connection, String searchTerm, String
                                                                    logicalName)
	{
		this.connection = connection;
		this.searchTerm = searchTerm;
		this.logicalName = logicalName;
	}

	@Override
	public void run() {
		ObjectOutputStream output = null;
		LinkedList fileList = new LinkedList();

		File rootDirectory = Environment.getExternalStorageDirectory();
		File[] rootContent = rootDirectory.listFiles();

		LinkedList results = new LinkedList();

		Pattern searchTermPattern = Pattern.compile(searchTerm,
                                                Pattern.CASE_INSENSITIVE);
		Matcher matcher =
                    searchTermPattern.matcher(rootContent[0].getName());

		for(File file: rootContent){
			String fileName = file.getName();
			matcher.reset(fileName);
			if(matcher.find()==true)
				results.add(fileName+" ("+logicalName+")");
		}

		// invia la lista dei risultati
		try {
			output = new ObjectOutputStream(connection.getOutputStream());
			output.writeObject(results);
			output.flush();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
		finally{
			try {
				output.close();
				connection.close();
			}
			catch(Exception e){
				e.printStackTrace();
```

```
                    }
                }
            }

        }
```

File SearchResponderAgent.java

```
package mobile.filesharing;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class SearchResponderAgent implements Runnable {

	private int myPort;
	private String logicalName;
	private ExecutorService exec;

	public SearchResponderAgent(int myPort, String logicalName){
		this.myPort = myPort;
		this.logicalName = logicalName;
		// accepted maximum of 5 simultaneous requests
		exec = Executors.newFixedThreadPool(5);
	}


	@Override
	public void run() {
		ServerSocket ss = null;
		Socket connection = null;
		ObjectInputStream input = null;
		try {
			ss = new ServerSocket(myPort);
			while(!isBelowThreshold()){
				// used to receive a term search
				connection = ss.accept();
				input = new ObjectInputStream(
							connection.getInputStream());
				String searchTerm = (String)input.readObject();
				exec.execute(
					new SingleRequestManager(
						connection, searchTerm, logicalName));
			}
			stopServingRequests();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
		catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
		finally{
			try {
				input.close();
				connection.close();
			}
			catch(Exception e){
				e.printStackTrace();
			}
		}
	}

	private void stopServingRequests(){
		try {
			exec.shutdown();
			exec.awaitTermination(60, TimeUnit.SECONDS);
```

```
                        exec.shutdownNow();
                }
                catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        private boolean isBelowThreshold(){
                /*
                 * TODO Battery level from OS should be retrieved
                 * (batteryLevel < THRESHOLD);
                 */
                return false;
        }
}
```

File MobileAppActivity.java

```
package mobile.filesharing;

import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import java.util.Set;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import android.app.Activity;
import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;
import common.SynchronizedMap;
import common.UserRecord;


public class MobileAppActivity extends Activity {

       private String searchAgentIp;
       private String logicalName;
       private int searchAgentPort;
       private UserAgent userAgent;
       private SearchResponderAgent searchAgent;
       private ExecutorService executor;

       private Bundle bundle = null;
       private EditText searchField = null;
       private TextView textView;
       private Button searchButton = null;
       private ListView resultListView = null;
       private ArrayAdapter<String> resultAdapter;
       private int resourceId;
       private ArrayList<String> currentResults;

       Handler guiHandler;

       private BluetoothAdapter bluetoothAdapter;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
             setContentView(R.layout.mobile_app);
```

```
        resourceId = R.layout.list_item;
        currentResults = new ArrayList<String>();
        bundle = this.getIntent().getExtras();
        logicalName = bundle.getString("LOGICAL_NAME");
        searchAgentIp = retrieveMyIpAddress();
        // codice eseguito solo su emulatore
        if(searchAgentIp == null || searchAgentIp.equals("10.0.2.15"))
              searchAgentIp = "10.0.2.2";

        searchAgentPort = new Random().nextInt(50000)+6001;

        //     startBluetoothService();

        startWorkerThreads();
        guiHandler = new Handler();
        setGUIcomponents();
        setListeners();
}


private void startWorkerThreads(){
   userAgent = new UserAgent(searchAgentIp, searchAgentPort, logicalName,
                                                                this);
        searchAgent = new SearchResponderAgent(searchAgentPort, logicalName);
        executor = Executors.newFixedThreadPool(2);
        executor.execute(userAgent);
        executor.execute(searchAgent);
}

private void setGUIcomponents(){
   textView = (TextView)findViewById(R.id.connection_status);
        textView.setText("Online @ "+searchAgentIp+":"+searchAgentPort);
        searchField = (EditText)findViewById(R.id.search_field);
        searchButton = (Button)findViewById(R.id.search_button);
        resultListView = (ListView)findViewById(R.id.result_list);
        resultAdapter = new ArrayAdapter<String>(this, resourceId,
                                                    currentResults);
        resultListView.setAdapter(resultAdapter);
}

private void setListeners(){
   searchButton.setOnClickListener(new SearchListener());
   // resultListView.setOnItemClickListener(new BluetoothCheckListener());
}

/*
 * Enables the bluetooth adapter and sets the logical name. Also starts the
 * discovery.
 */
private void startBluetoothService() {
   Log.d(logicalName, "starting Bluetooth");
   bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if(!bluetoothAdapter.isEnabled())
              bluetoothAdapter.enable();
        bluetoothAdapter.setName(logicalName);
        bluetoothAdapter.startDiscovery();
}

private void stopBluetoothService() {
   if(bluetoothAdapter.isEnabled())
        bluetoothAdapter.disable();
}

protected void onStop(){
   super.onStop();
}
```

```
protected void onDestroy(){
   stopBluetoothService();
}

class SearchListener implements OnClickListener {

   @Override
   public void onClick(View v) {
          currentResults.clear();
          resultAdapter.notifyDataSetChanged();
          String searchTerm = (String)searchField.getText().toString();
          if(searchTerm.equals(""))
                return;
          ExecutorService execService = Executors.newCachedThreadPool();
          SynchronizedMap<String, UserRecord> peers = userAgent.getPeers();
          if(peers == null)
                return;

          // peers
          Set<String> peersId = peers.getKeys();
          if(peersId.isEmpty())
                return;

          // remove itself from the set of peers
          peersId.remove(searchAgentIp+":"+searchAgentPort);
          if(peersId.isEmpty())
                return;
          /*
           * Starting one FileSearcher thread per known peer
           * Results from these threads will be directly shown in the GUI.
           *
           */
          Collection<Callable<LinkedList>> taskList =
                                      new LinkedList<Callable<LinkedList>>();
          for(String peerId: peersId){
              Callable<LinkedList> fileSearcher = new FileSearcher(peerId,
                                                                   searchTerm);
              taskList.add(fileSearcher);
          }

          try {
              List<Future<LinkedList>> futureList =
                 (List<Future<LinkedList>>)execService.invokeAll(taskList);
              for(Future<LinkedList> handler: futureList){
                 LinkedList<String> peerFiles =
                                            LinkedList<String>)handler.get();
                 //visualizzazione risultati
                 for(String it:peerFiles){
                        Log.d(logicalName, it);
                        currentResults.add(it);
                 }
              }
              resultAdapter.notifyDataSetChanged();
          }
          catch (InterruptedException e) {
                e.printStackTrace();
          }
          catch (ExecutionException e) {
                e.printStackTrace();
          }
   }
}


private String retrieveMyIpAddress() {
   try{
```

```
        for(Enumeration<NetworkInterface> en =
             NetworkInterface.getNetworkInterfaces(); en.hasMoreElements();)
        {
           NetworkInterface net = en.nextElement();

           for(Enumeration<InetAddress> enumIPAddress = net.getInetAddresses();
               enumIPAddress.hasMoreElements();)
           {
             InetAddress inetAddress = enumIPAddress.nextElement();
             if(!inetAddress.isLoopbackAddress())
                 return inetAddress.getHostAddress().toString();
           }
        }
   }
   catch(SocketException exc){
          Log.e("MobileAppActivity", exc.toString());
   }
   return null;
}


/*
 * Classe che consente ad altri thread di cambiare il messaggio di stato
 * nella textview utilizzando il metodo "post" dell'oggetto Handler.
 */
public class TextViewMessage implements Runnable{

   String message;

   public TextViewMessage(String aMessage){
          message=aMessage;
   }

   public void run(){
          textView.setText(message);
   }
}


public class BluetoothCheckListener implements OnItemClickListener{

          @Override
          public void onItemClick(AdapterView<?> listView, View arg1, int
                                                          position, long id)
          {
            String listEntry = (String)listView.getSelectedItem();
            String[] tokens = listEntry.split("(");
            String peerName = tokens[1].substring(0, tokens[1].length()-1);
            Set<BluetoothDevice> peerDevices =
                                       bluetoothAdapter.getBondedDevices();

            AlertDialog.Builder dialog = new
                                AlertDialog.Builder(MobileAppActivity.this);
            String title, message;

            if(peerDevices.contains(peerName)){
                 title = "Connection OK";
                 message = "You can connect to "+peerName;
            }
            else{
                 title = "Bad connection";
                 message = "cannot connect to "+peerName;
            }
            dialog.setCancelable(true);
            dialog.show();
          }
}
```

```
}
```

File Server.java

```
package discovery;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import common.SynchronizedMap;
import common.UserRecord;

public class Server implements Runnable{

	public static final int SERVERPORT = 6000;
	public static final int OFFLINE_TIME = 60000;

	private SynchronizedMap<String, UserRecord> peersMap = new
					SynchronizedMap<String, UserRecord>();

	public void run(){
		try {
			ServerGUI serverGui = new ServerGUI();
			int serverIndex = serverGui.addNewLog("Server main process");
			System.out.println("Index = "+serverIndex);

			ServerSocket ss = new ServerSocket(SERVERPORT);
			serverGui.writeOnLog(serverIndex, "Server Started.");

			while(true){
				serverGui.writeOnLog(serverIndex, "Listening on port:
					"+SERVERPORT+" ...");
				Socket clientConnection = ss.accept();
				serverGui.writeOnLog(serverIndex, "New client accepted:
					starting new server thread.");

				UserManager manager = new UserManager(peersMap,
					clientConnection, serverGui);
				manager.start();
				serverGui.writeOnLog(serverIndex, "New User Manager
					Started");
			}
		}
		catch (IOException e) {
			e.printStackTrace();
		}
	}

	public static void main(String[] args){
		Thread mainThread = new Thread(new Server());
		mainThread.start();
	}
}
```

File ServerGUI.java

```
package discovery;

import java.awt.Font;
import java.util.LinkedList;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;

public class ServerGUI extends JFrame{


	private static final long serialVersionUID = -3900509973504297511L;
	private JTabbedPane tabbedPanel;
	private LinkedList<JTextArea> areasList;

	public ServerGUI(){
		setFont(new Font("Arial", Font.PLAIN, 20));
		setTitle("Server Log");
		setSize(600, 600);
		setDefaultCloseOperation(EXIT_ON_CLOSE);

		tabbedPanel = new JTabbedPane();
		tabbedPanel.setFont(new Font("Arial", Font.PLAIN, 16));
		areasList = new LinkedList<JTextArea>();
		setContentPane(tabbedPanel);

		setVisible(true);
	}

	public int addNewLog(final String loggerName){
		JTextArea area = new JTextArea();
		area.setFont(new Font("Arial", Font.PLAIN, 16));
		JScrollPane scroll = new JScrollPane(area);
		SwingUtilities.invokeLater(new LogAdder(loggerName, scroll));
		areasList.addLast(area);
		return areasList.indexOf(area);
	}

	public void writeOnLog(final int index, final String message){
		SwingUtilities.invokeLater(new LogWriter(index, message));
	}

	public class LogAdder implements Runnable {

		private String loggerName;
		private JScrollPane scroll;

		public LogAdder(String loggerName, JScrollPane scroll){
			this.loggerName = loggerName;
			this.scroll = scroll;
		}

		public void run(){
			tabbedPanel.addTab(loggerName, scroll);
		}
	}

	public class LogWriter implements Runnable {

		private int index;
		private String message;
```

```
		public LogWriter(int index, String message){
			this.index = index;
			this.message = message;
		}

		public void run(){
			areasList.get(index).append(message+"\n");
		}
	}
}
```

File UserManager.java

```
package discovery;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Collection;
import java.util.LinkedList;

import common.Message;
import common.SynchronizedMap;
import common.UserRecord;

public class UserManager extends Thread {

	// socket già disponibile per la comunicazione
	private Socket connection;
	private ObjectInputStream input;
	private ObjectOutputStream output;
	private String userId;
	private volatile SynchronizedMap<String, UserRecord> peersMap;

	private ServerGUI serverGui;

	public UserManager(SynchronizedMap<String, UserRecord> map, Socket
                                            connection, ServerGUI serverGui)
	{
		peersMap = map;
		this.connection = connection;
		this.serverGui = serverGui;
	}

	public void run(){
		UserRecord record = null;
		int index = -1;
		// Register new user
		try{
			input = new ObjectInputStream(connection.getInputStream());
			output = new ObjectOutputStream(connection.getOutputStream());
			System.out.println("UM::Ho aperto gli stream di in e di out");
			record = (UserRecord)input.readObject();
			userId = record.getIp()+":"+record.getPort();
			peersMap.put(userId, record);

			index = serverGui.addNewLog(
                                record.getLogicalName()+"@"+userId);
			serverGui.writeOnLog(index, "["+System.currentTimeMillis()+"]
                                                        Registration OK");
		}
		catch (IOException e) {
			e.printStackTrace();
		}
		catch (ClassNotFoundException e) {
			e.printStackTrace();
		}

		// listen for PING and PEERS_UPDATE messages
		try {
			connection.setSoTimeout(Server.OFFLINE_TIME);
			while(true){
				Message message = (Message)input.readObject();
				if(message.getMessageType().equals("PING")){
					long currentTime = System.currentTimeMillis();
					record.setLastTimestamp(currentTime);
```

```
                                serverGui.writeOnLog(index, "["+currentTime+"]
                                                Received a PING MESSAGE");
                                serverGui.writeOnLog(index, "new time_stamp is
                                                "+record.getLastTimestamp());
                        }
                        if(message.getMessageType().equals("PEERS_UPDATE")){
                                serverGui.writeOnLog(index, "SERVER-THREAD::La
                                size della mappa lato Server è"+peersMap.size());

                                LinkedList list = new LinkedList();
                                Collection<UserRecord> peersRecords =
                                                        peersMap.getValues();
                                for(UserRecord rec: peersRecords)
                                        list.add(rec);

                                output.writeObject(list);

                                output.flush();
                                serverGui.writeOnLog(index,
                                        "["+System.currentTimeMillis()+
                                        "] Received an UPDATE MESSAGE");
                        }
                }
        }
        catch(Exception e){
                // timeout is expired
                peersMap.remove(userId);
                serverGui.writeOnLog(index, "DISCONNECTED!");
        }
        finally{
                try {
                        input.close();
                        output.close();
                        connection.close();
                }
                catch(Exception e){
                        e.printStackTrace();
                }
        }
    }

}
```