Elisa Gonzalez Boix (email:egonzale@vub.ac.be, office:10F731)
Jorge Vallejos (email:jvallejo@vub.ac.be, office:10F724)

4 Partial Failures in AmbientTalk: Mobile Music Player

Ambienttalk's tutorial and language reference are available at `http://soft.vub.ac.be/amop/`. The lab session material is available at `http://soft.vub.ac.be/amop/teaching/dmpp`

4.1 Idea

The purpose of this exercise is to gain insights into the behaviour of far references and failure handling by implementing a music player meant to be used on mobile devices. The music player contains a library of songs. When a music player discovers another player in its environment, they set up an ad hoc network and exchange their music library's list. After the exchange, the music player calculate the percentage of songs both users have in common. If this percentage exceeds a certain threshold, the music player warn the user that someone with a similar musical taste is nearby.

4.2 Implementing the mobile music player

We will implement the music player application starting from a skeleton code shown below (also available in the lab material):

```
// the library consists of simple song objects
def Song := object: { //... };
def THRESHOLD := 40; // when users share 40 % of their songs, we signal a match
def makeMusicPlayer(username){
  def myLib :=  Vector.new();// the local user's songs library
  // the remote interface object that will be exported
  def remoteFacade := object: {
    // returns a session object encapsulating the music library exchange
    def openSession(remoteUser) {  //...}
  };
  def localInterface := object: {
    // engage in a peer-to-peer service discovery to discover another music player
    def goOnline() { //... };
    def addSong(artist, title) {
      myLib.add(Song.new(artist, title));
    };
  };
  localInterface;
};
```

A music player application is created by invoking a function called `makeMusicPlayer`. The function defines two nested objects named `localInterface` and `remoteFacade` which respectively define the methods invoked locally and the methods of the distributed protocol that music players will use to exchange their library's list. Use the above skeleton to incrementally grow the application as follows:

(a) Complete the implementation of the skeleton code with the distributed protocol of a music player which works as follows:

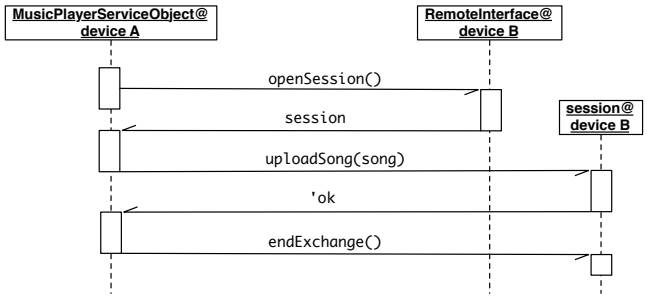

Figure 1: The music library exchange protocol

Once a music player discovers another one, it asks the remote player to open a library exchange session by sending it the `openSession` message which returns a session object. A session object implements methods to send song information (`uploadSong`) and to signal the end of the library exchange (`endExchange`). A music player uses the session object to send all of its own songs one by one. Once all songs are sent, the end of the exchange is signalled. After the exchange, the percentage of songs both users have in common is calculated and the application signals the presence of a user with similar taste if needed.

Note: Use the provided `testAsyncMusicPlayerDiscovery()` unit test to help you in this iteration.

(b) Adapt your implementation to add failure handling code to deal with transient and permanent failures. More concretely, add failure handling support to (a) the necessary asynchronous message sends, and (b) the session object.

Hint: The session object is clearly only relevant within the context of a single music library exchange. If the exchange cannot be completed (due to a persistent network partition or a crash), the session object and the resources it transitively keeps alive should be reclaimed. This indicates that the remote reference to this session object should be leased, such that both music players can gracefully terminate the exchange process if the lease expires. If the exchange is successfully completed (i.e the remote player signals the end of the exchange), the lease should also expire.

(c) Uncomment `testAsyncMusicPlayerExchange` unit test and adapt your implementation to pass it. You will need to change the `makeMusicPlayer` constructor function to receive as parameter a closure which is applied when a remote peer has a similar taste.

(d) Add a unit test to test the library exchange when one player disconnects permanently.