Creating an application for LaCOLLA 1.0.X

Author: *Enric Jaen*

Date: March 13th 2007

Table of Contents

1. Introduction

This document explains how to write a simple application. It is assumed that LaCOLLA is already installed. For a guide about how to install LaCOLLA see the download section in the web side.

The demo application described here is named **PutGo**, and the source code can be downloaded from the web side:

`http://lacolla.uoc.edu/cgi-bin/viewvc.cgi/LaCOLLA_applications/Applications/putgo/`

2. Application structure

An application written in LaColla 1.0.x contains at least the following four Java classes. Only the two first are application-dependent. The other two are explained for a better understanding of the application functioning.

- **Main applicacion class (application-dependent)**
  The main class creates an instance of the `ApplicationSideApiServer` class, and it must discover the endpoint of LaColla API by invoking the `resolveApi()` method. After that, the application can login and invoke operations on LaColla. See section 3 for a detailed description.

- **Api.ApplicationSideApiImpl (application-dependent)**
  This class implements the `ApplicationSideApi` interface, and must be written by the application developer. It handles the callback methods (i.e notifications) that LaColla sends

to the application during its lifetime. See section 5 for a detailed description.

- **Api.ApplicationSideApiServer (application-independent)**
  This is a helper application, therefore the application programmer won't need to modify it. The constructor of this class registers (i.e. binds) the `ApplicationSideApiImpl` class into the RMI registry. The name of the application has an URI as follows:

  `"/ApplicationSideApi" + <applicationPort>`

  Where `<applicationPort>` is given as a constructor parameter. The RMI registry may be running in another host, which is also given as a constructor parameter. The RMI registry must be listening on port 1099.

  This class also contains the `resolveApi()` method to discover the endpoint of LaCOLLA API. (`LaColla.Api.Api` class). The name of LaColla API has an URI as follows:

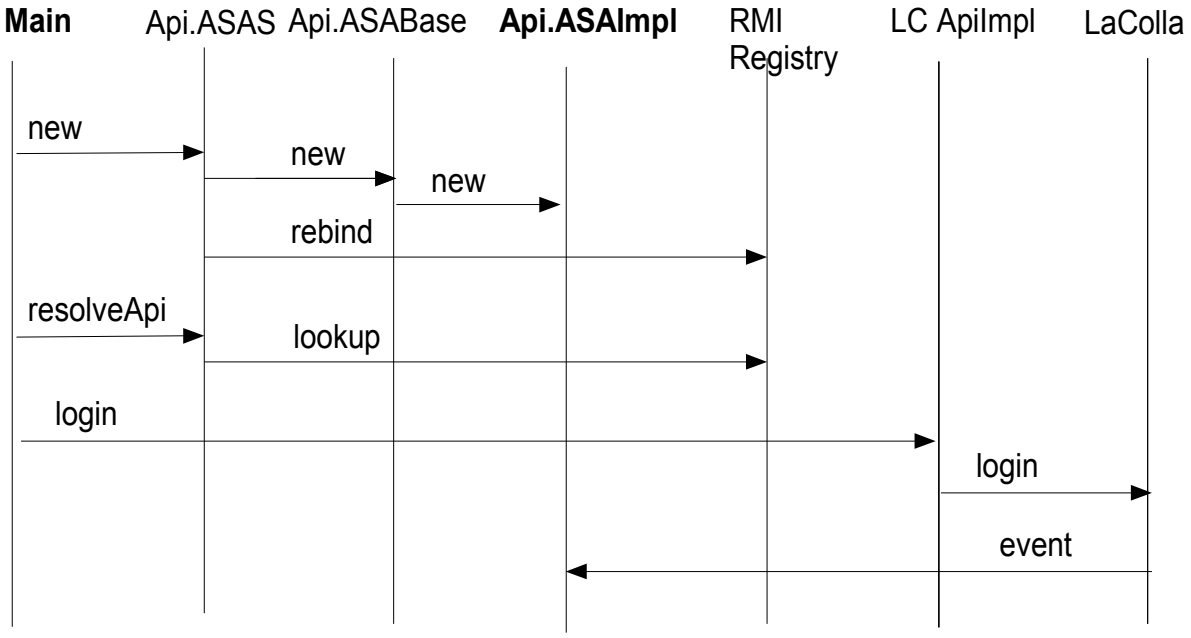
  `"/Api"+<apiPort>`

  Where <apiPort> is given as an application parameter, and it is defined by an UserAgent of LaColla [ref].

- **Api.ApplicationSideApiBase (application-independent)**
  This class is internally used by the Api.ApplicationSideApiServer class to handle the summary of events generated by LaCOLLA, and to instantiate the class that handles the events.

3. Initialization code

The following diagram summaries the startup of the application. The two bolded classes are the ones developed by the programmer.



The source code of the main application class is listed next. Notice the name of the application in bold.

```
package Apps.PutGo;

import Apps.PutGo.Api.ApplicationsSideApiServer;
import Apps.PutGo.Api.ApplicationsSideApiImpl;
import LaColla.Api.Api;


public class PutGo {

  private static final int RMIport = 11099;
  private static String appIP;      //application IP
  private static int appPort;       //application port
  private static int apiPort;       //LaCOLLA API port
  private static String username="root";
  private static String passwd="root";
  private static String groupId="hola";


   public static void main(String[] args) {

            try {
                   appIP =InetAddress.getLocalHost().getHostAddress();
            } catch (UnknownHostException e1) {
                    e1.printStackTrace();
            }

            apiPort=Integer.parseInt(args[0]);
            appPort=Integer.parseInt(args[1]);


           ApplicationsSideApiServer asas=new
                              ApplicationsSideApiServer(appIP, appPort);

           Api api = asas.resolveApi(apiPort);


           …
  } //end main
 } //end class
```

4. User Login

Before using LaCOLLA, the user must first login into LaCOLLA. The code is as follows:

```
           int gapaIP = appIP;
           int gapaPort = 2001;
           String apiid = api.login(
                                groupId,
                                username,
                                passwd,
                                gapaIP,
                                gapaPort,
                                appIP,
                                appPort,
                                asas.readSummary());
```

5. Application callback code

The developer must implement the code of the LaCOLLA callback operations in a file named `ApplicationsSideApiImpl.java`. Its structure is as follows. Notice that the package includes the application name.

```
package Apps.PutGo.Api;

import LaColla.Api.ApplicationsSideApi;
import LaColla.core.data.Event;

public class ApplicationsSideApiImpl implements ApplicationsSideApi{
      private String memberId;
      public ApplicationsSideApiImpl() throws RemoteException {
          super();
      }

      public void newConnectedMember(
                        String groupId, String userId, String memberId)
                  throws RemoteException {
          this.memberId=memberId;
         ... //your code
      }

      public void newEvent(
                  String groupId, Event evt)
                  throws RemoteException {
         ... //your code
      }


      //rest of callback operations..
} //end class
```

6. Execution of the PUTGO application

We assume with respect LACOLLA that:

- Is already running (`ant lacolla`),
- The RA has a user named `admin` registered. (see `member1.ini`)
- The UA has an API listening in port 19001 (see `UA1.ini`)

The **putgo ANT target** has the following two parameters:

```
<target name="putgo" depends="build">
      ...
    <arg value="19001"/>  <-- API PORT
```

```
    <arg value="4321"/>   <-- APPLICATION PORT
</target>
```

To run the application type:

```
$ant putgo
     [java] PUTGO:              127.0.0.1
     [java] Host= 127.0.0.1 appPort=4321 LaCOLLAport= 19001
     [java] creada instancia de AplicationsSideApi
     [java] rmiport=11099
     [java] name published /ApplicationsSideApi4321
     [java] ******************
     [java] *List of Commands*
     [java]
     [java] exit
     [java] login
     [java] logout
     [java] putobject
     [java] getobject
     [java] devent
     [java] erelated
     [java] rmobject
     [java] addgrp
     [java] addusr
     [java] instmsg
     [java] getinfo
     [java] listcmd
     [java] please insert a command:
```

Then type `login`:

```
login
     [java] log4j:WARN No appenders could be found for logger
(LaColla.core.util.Identificator).
     [java] log4j:WARN Please initialize the log4j system properly.
     [java] Starting Tyrex  Version 1.0.3
     [java] Original code is Copyright (c) 1999-2001, Intalio, Inc. All Rights
Reserved. Contributions by MetaBoss team are Copyright (c) 2003-2005, Softaris
Pty. Ltd. All Rights Reserved.

     [java] your command: login
     [java] LOGIN
(1)  [java] username=admin
     [java] java.io.FileNotFoundException: lc_events.out (No such file or
directory)
     [java] tss=
     [java] allReceived: {}
     [java] nonConsecutive: {}
     [java] java.io.FileNotFoundException: lc_events.out (No such file or
directory)
(2)  [java] Login correcte: App#28b0f780c21a10048c06e7e96d09e474#
(3) newConnectedMember hola admin member#0#
(4) newEvent <<UA#bd327eb0bf3110048958f52c8803bee2#,1>,null,201,hola,new
connected Member member#0#,0>
     [java] memberid=member#0#
     [java] please insert a command:
```

Notice in the previous output that (1) the user `admin` does the login. As a response the application (2) gets an Application ID, (3) receives a `newConnectedMember` event (see section 5), and (4)

receives a `newEvent` also indicating a new connected member (see section 5). Should LaCOLLA have more pending events, the application will also receive them.

Note: don't worry about the FileNotFoundException; it is a logged exception that indicates that there is no previous events stored. This summary is stored in the file `lc_events.out`.

After playing with the application type `logout` and/or `exit`.

```
logout
     [java] your command: logout
     [java] please insert a command:
exit
     [java] your command: exit
```

7. API status

You can obtain the current status of the LaCOLLA API from this URL:

`http://lacolla.uoc.edu/lacolla/releases/1.0.x/doc/api-status.html`