solaiemes

REST OMA API – GSMA Innovation Accelerator Service Sample: Joyn access to saying service

DocVersion: 1.0 DRAFT
Type: Tech Description
Date: 26/11/12

REVISION

Version	Date	Author	Comments
1	26/11/2012	Francisco Carriedo (fcarriedo@solaiemes.com)	Version 1.0 DRAFT

INDEX

1. Code overview

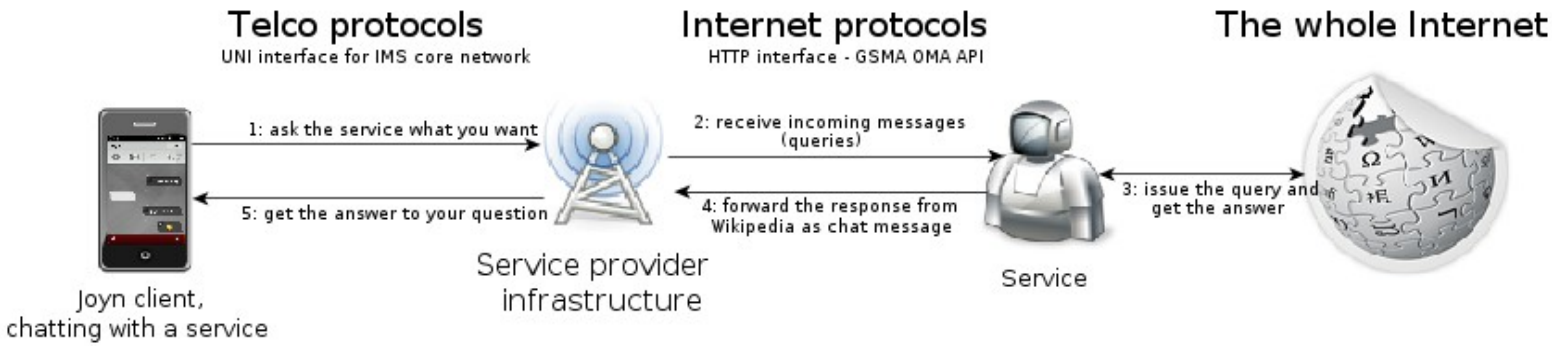
This project contains a very simple example of a SERVICE that is registered and reachable as a RCS user and capable of getting resources through HTTP.

The project is not optimized at all and does not consider all the possible error scenarios. Just serves as starting point for any developer to create his own service.

The program is written in Java and organized as described below.

- **com.solaiemes.serviceExample:**
  - ServiceExampleLauncher.java: just the checking of the parameters needed and creation of another thread that will run ServiceExample.java code.
  - ServiceExample.java: the main block of the service, getting requests from the incoming notifications (long polling ~20 seconds of waiting on each checking, and forwarding the request to the corresponding Internet service if there were such requests).
  - ServiceExampleActions.java, all the invocations needed to the HTTP OMA API to get requests and send responses (register, unregister, create channel notification, create chat subscription, get pending notifications).
- **com.solaiemes.serviceExample.control** just contains a MBean to control the service start / stop. If you want to know more about the MBeans check this, but long story short is a way of controlling a service running on a JVM using JConsole for example (check this).
- **com.solaiemes.serviceExample.datastructures** contains the classes that represent basically the JSON messages included in the HTTP requests issued to the OMA API (a chat message, a subscription to a channel...).
- **com.solaiemes.serviceExample.internetconnectors** contains the Internet-side of your service. For example the connector contained in QuoteGetter.java just receives the query coming from the RCS-side of the service and forwards it to a quote service through HTTP. Once the request from the service has come, it is parsed appropriately into JSON message and sent back by the service as RCS chat message to the user that asked initially. To create a new service you just need to create a new class on this package that communicates appropriately with the Internet.
- **com.solaiemes.serviceExample.utils** contains convenience classes that unify resources needed in different parts of the project such as the JSON parser (SimpleNotificationParser.java), the HTTP resources (HttpResources.java), a builder of HTTP clients (one for interacting with the API, another one to issue HTTP requests to the quote service or whatever you need when developing your own service) and finally a central point where to check the status code for all the HTTP responses.

The following picture contains all the actors involved and interactions needed to illustrate the service and please note that you will need a pair of joyn users to test it, one for the service to be reachable and the other one for the user that will chat with the service (you can get them following the explanation on the GSMA Innovation Accelerator website):


Telco protocols
UNI interface for IMS core network
Internet protocols
HTTP interface - GSMA OMA API
The whole Internet
1: ask the service what you want
5: get the answer to your question
2: receive incoming messages (queries)
4: forward the response from Wikipedia as chat message
3: issue the query and get the answer
Joyn client, chatting with a service
Service provider infrastructure
Service

2. Compilation and execution

The code needs to be compiled using Apache Maven (tested with version 2.2.1, available here). See the relevant lines highlighted in bold style to compare with your output:

```
$ cd <HOME_PATH_FOR_THE_PROJECT>

$ mvn clean install

[...]

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 5 seconds

[INFO] Finished at: Tue Nov 27 13:14:13 CET 2012

[INFO] Final Memory: 27M/431M

[INFO] ------------------------------------------------------------------------
```

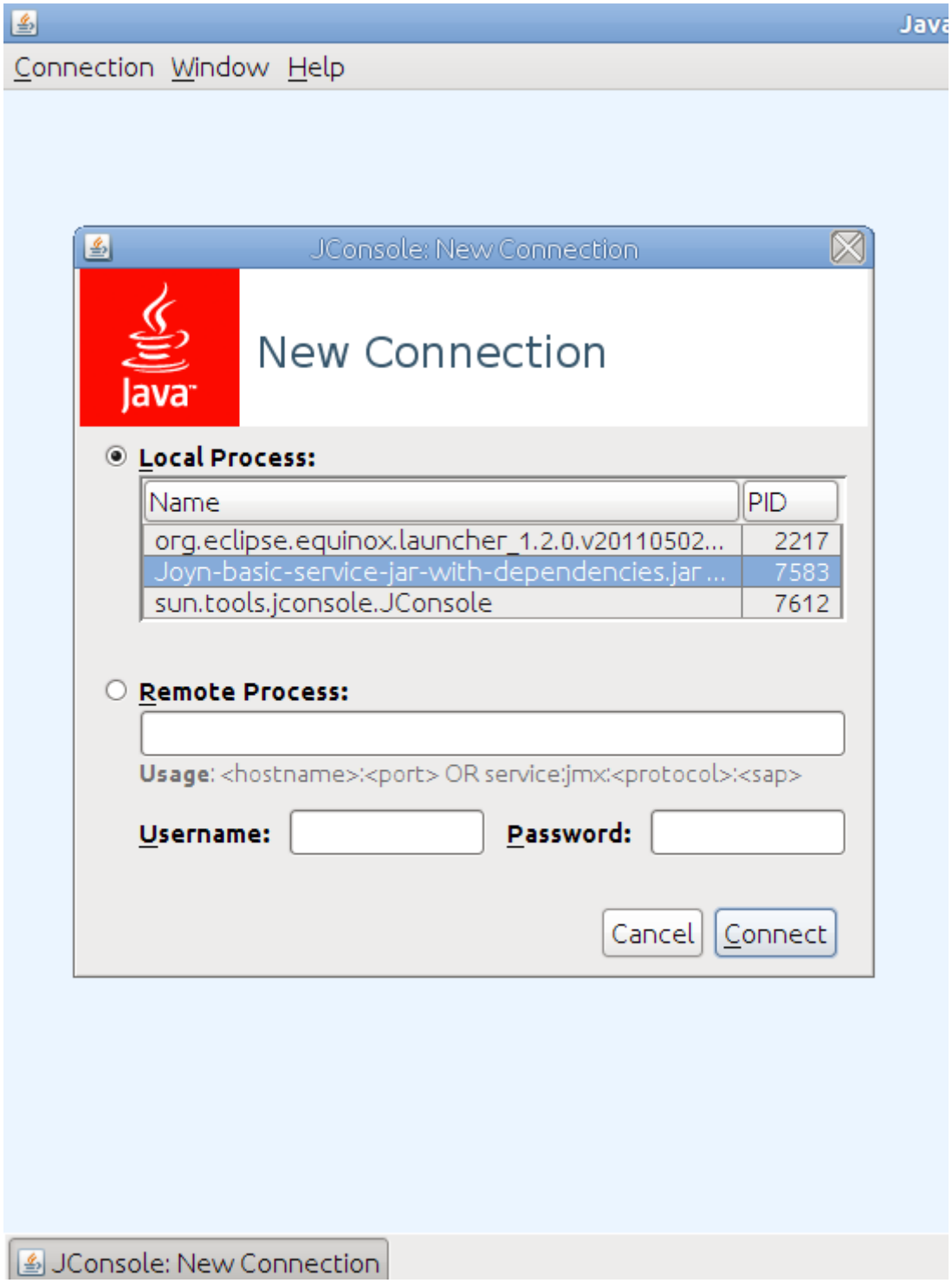
To execute the service you need to provide the username and the password corresponding to the service as parameter, that will be registered and listening to the queries submitted by other users through a chat message. Please note that the password may contain sensible characters for your shell, you should surround it with double quotes to be considered a literal (see the example below). The normal output of the service will display the following lines:

```
$ java -jar target/Joyn-basic-service-jar-with-dependencies.jar +34123456789
"<&/yourP455w0rd"

OK - Starting service as user +34123456789

OK - HTTP 201

OK - Session notification subscription created: 201

OK - HTTP 201

OK - Notification channel and subscriptions created correctly for user: +34123456789

OK - Registered successfully: 204

OK - Register ok as user +34123456789

OK - The service was re-launched...

OK - Listening for the commands coming from the MBean...

OK - No notifications for your service right now...

OK - No notifications for your service right now...
```

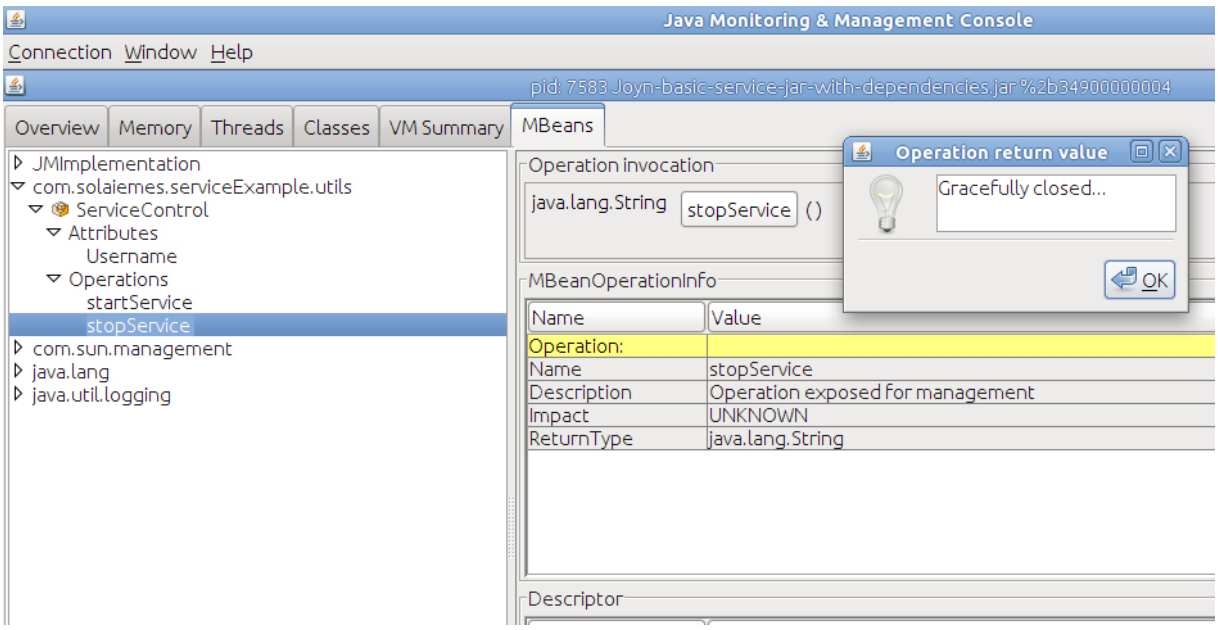
As a convenience, an Mbean that is capable of starting and stopping the service is provided. To use it you will need to use JConsole (check this). The behaviour of the service pretends to be pretty similar to that of an application server: it keeps running standalone and is managed from an external interface (JConsole in this

case). To quit the execution the service has to be unregistered (through JConsole) and the process killed (CTRL+C for example will do the job).

On the images below appears how to access the MBean of the Java process with JConsole, first of all one must to fire up the program using the `jconsole` command within a shell on your system, then connect to the process:



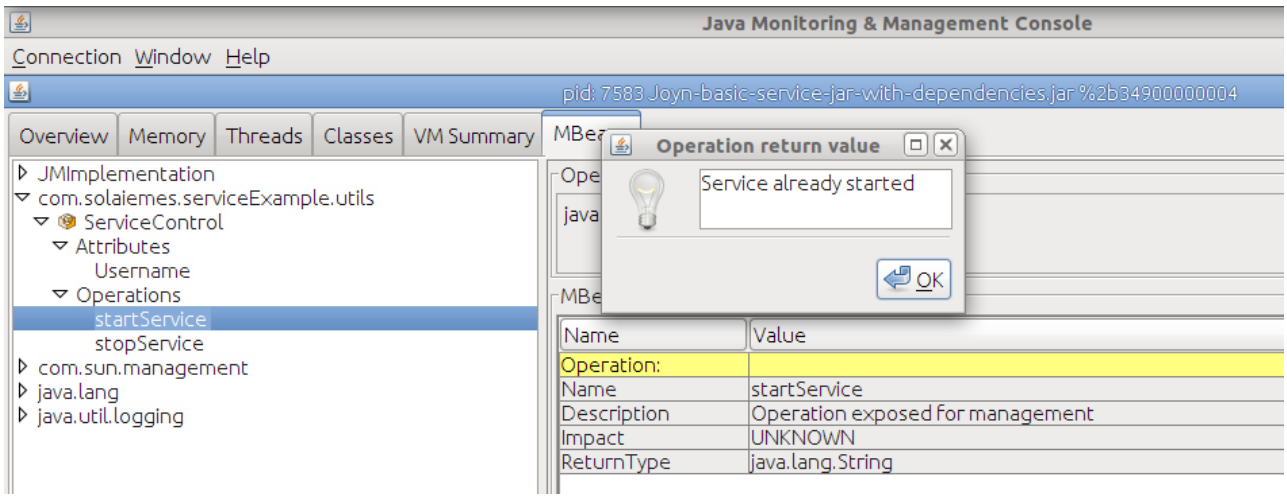
Once the process is connected, you can expand the tree on the left side to access the Mbean:



As the service is autostarted when you initiate the process from the shell, you can stop clicking the stopService button and you will get a message about how the operation resulted. You should get the following lines in the shell you used to fire up the process:

```
OK - No notifications for your service right now...

OK - Shuting down the service.

OK - Unregistered successfully: 204

OK - Unregister ok as user +34123456789
```

Now you can re-start the service again:



You should get the following output on the console:

```
OK - HTTP 201

OK - Session notification subscription created: 201

OK - HTTP 201

OK - Notification channel and subscriptions created correctly for user: +34123456789

OK - Registered successfully: 204

OK - Register ok as user +34123456789

OK - The service was re-launched...
```

That's all!

3. Further questions for developers

If you find issues while playing with this sample you can check the developer document send your questions to **JoynIC_Developer_Support@gsm.org**.