MEGA PROJECT

GASP Programmer Tutorial
« How to design a GASP Game »

Romain PELLERIN
CNAM-INT

Summary:

This tutorial explains to the game programmer how to design a game for the GAming Services Platform (GASP) in terms of conception and publishing.

TABLE OF CONTENTS

1. Introduction

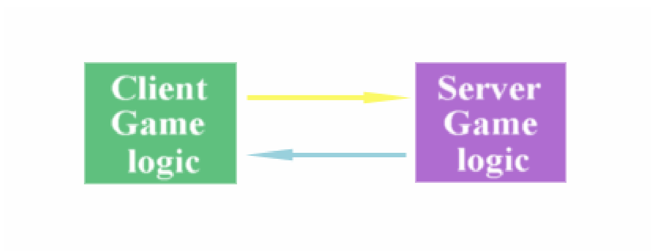
The Gaming Services Platform, **GASP**, is an open game services platform for mobiles. It is based on the Open Mobile Alliance specifications version 2.0 and improve its architecture while keeping its essence, in fact an OMA compliant game may running on GASP without massive code modifications.

This tutorial presents how to design a GASP compliant Game, how to use the GASP utils and finally how to publish a GASP game.

2. How to design a GASP compliant game

The game programmer who wants to create a game designed for the GAming Services Platform must follow some rules, these rules are described in this how-to.

The traditionnal architecture for multiplayer games is the client/server architecture with a direct communication between the client game logic and the server game logic.



Traditionnal communication architecture

To offer gaming services, for example lobby service, it is necessary to have a platform hosting the games. Also it is particularly important in the mobile communication way because the mobile cannot connect to an another mobile via GPRS or UMTS communication, a bridge between mobiles is necessary as GASP is.
To provide mobiles to communicate with GASP or between a mobile and a server hosted by the platform, it is necessary to have some communication interfaces.
For GASP, these interfaces are the abstract classes **GASPClient** and **GASPServer**.

GASPClient.java

This abstract class must be extended by the client game logic and provides all the methods to communicate with the platform.

- See: **org.mega.mgp.client.midp.GASPClient**
- or: **org.mega.mgp.client.doja.GASPClient**

GASPServer.java

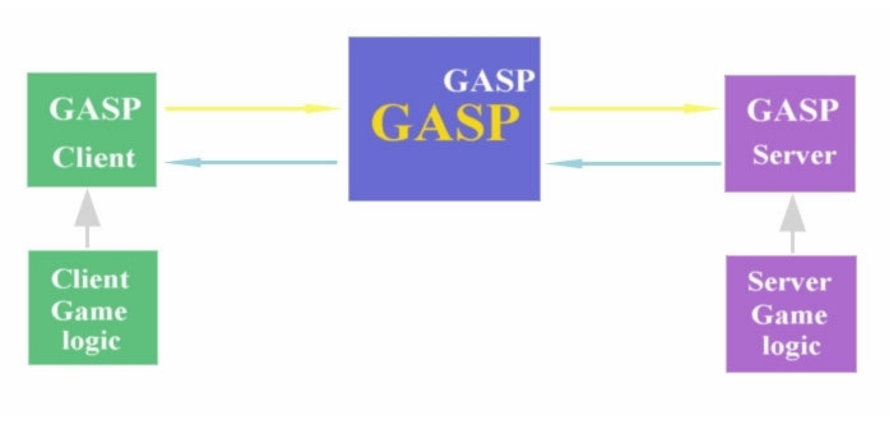
This abstract class must be extended by the server game logic and provides all the methods to communicate with GASP.

- See:**org.mega.mgp.server.GASPServer**

<u>Example</u>:

```
public class ClientGameLogic extends GASPClient {...}
public class ServerGameLogic extends GASPServer{...}
```

In these two classes, the programmer must implements the herited abstracts methods, onJoinEvent, onStartEvent, onEndEvent, onQuitEvent, onDataEvent, the listenners to the GASP events: join, start, end, quit and data.



GASP communication architecture

In-game communication between the client game logic and the server game logic must be optimized in term of data transfered size due to the mobile communication low bandwith.
It is out of the question to serialise java objects over that type of communication.
The MooDS Generator provide this optimisation in in-game data transfert.

CustomTypes.java

This class provide encoding/decoding methods of the custom types used by the game programmer in the communication between the client game logic and the server game logic.
It is generated by the generator org.mega.moods.MooDSGenerator .
See: **org.mega.mgp.customTypes.CustomTypes**

MooDSGenerator.java

The programmer describes his custom types in an xml file corresponding to the XML Schema **types.xsd**, available in package **org.mega.moods**. In this package you also have an example of that kind of xml file**, types-describing-example.xml** .

Package structure:

To provide publishing of the game on GASP, the game must be packaged following the GASP Packaging:

```
app .
    | CustomTypes.java
    |  ...  and the custom types classes used in
    |     communication between the client
    |     game logic and the server game logic
    |
    | .client.
    |          |  GASPClient.java
    |          |  ... and all classes of the client game logic
    |          |      for example the midlet class
    |
    |
    | .server.
    |          |  GASPServer.java
    |          |  ... and all classes of the server game logic
```

Steps of GASP game programmation:

1. Create the Client and Server game logics extending the associated communication interfaces **GASPClient.java** and **GASPServer.java**

2. Determine all the custom types objects can be transfered between the client and server.

3. Use the generator to generate CustomTypes.java including the appropriate encoding/decoding methods.

   See: section 3 – "How to use the MooDS Generator"

4. Package the game following the GASP Packaging model.

5. Publish the game on GASP.

   See: section 4 – "How to publish a game on GASP"

3. How to use the MooDS Generator

The generation procedure steps are :

1. Making of the xml description of the custom types classes

   The programmer describes his custom types in an xml file corresponding to the XML Schema **types.xsd**, available in package **org.mega.moods**. In this package you also have an example of that kind of xml file**, types-describing-example.xml** .

2. Running the generator

   The programmer have to run the org.mega.mgp.customTypes.generator.Generator with the parameters as follows:

   - **MoodsGenerator** <types xml description> < game package name>

   It results the generation of the class CustomTypes.java, implementing the interface **org.mega.mgp.customTypes.CustomTypes**, providing the methods:

   - **void encodeData (Hashtable h, DataOutputStream dos)**

     this method provides to encode on the DataOutputStream dos, the custom objects (corresponding to the types described in the xml file) contained in the hashtable h.

   - **Hashtable decodeData(DataInputStream dis)**

     this method provides to decode the DataInputStream data objects and returns the decoded custom objects (corresponding to the types described in the xml file) in a hashtable.

3. Update the game package

   Then, the programmer places the generated class CustomTypes.java in the package of his game at root with the custom types object classes.

   <u>Example</u>:

   Considering a game programmer who wants to transfert two types of objects, Update.java and Delete.java, he describes his custom types in an xml file, types.xml (as describe in the xml-describing-example.xml), he runs the generator with the command line:

   ➢ MoodsGenerator types.xml app11

   After this step the programmer can place the generated CustomTypes.java and his custom types classes in the game package:
   app11/CustomTypes, app11/Delete and app11/Update

4. How to publish a game on GASP

After the game programmer has created his game and correctly package its, the next step is the publishing of the game on the GAming Services Platform.

There is two procedures associated with the two cases above:

1. Testing: the programmer wants to test his game and does not have already a GASP host provider who manages the applications.
2. Publishing: the programmer wants to publish his game on the GASP host provider

Let's describe the two procedures:

1. Testing:

   a. Choose a **unique** game application ID, **appID**, hard coded on the client game logic and the server game logic.
   b. Update the GASP database putting the choosed appID in the table applications.
   c. Rename your package as: **"app"+appID** Example: *app11*
   d. Create and place the two jar follows:
      - the **client jar** to place on the **mobile side**, containing the common classes and the client classes.

        Example:

        app11.jar, containing the packages *app11* and *app11.client* (not *app11.server*)

      - the **server jar** to place on **GASP side** in the web-inf/lib folder, containing the common classes and the server classes.

        Example:

        app11.jar, containing the packages *app11* and *app11.server* (not *app11.client*)

   e. Update the properties file **apps.properties** in the web-inf/lib folder, increment the number of apps on the plaform and write the line:

      "app"+ **appID** +".package="+your_package_name

      Example:

      app11.package=app11

2. Publishing:

   a. Contact the GASP host provider and require to him a **unique** application ID, **appID**.
   b. Hard-code this **appID** in your client and server game logic.
   c. Do the steps 1.c and 1.d, then send the **server jar** to your GASP host provider.

Thus the game is available on GAming Services Platform.