Cloud Based Communication in B2B Model

Tamara Luarasi and Andi Domi
Department of IT, Mathematics and and Statistics
European University of Tirana
Tirana, Albania
tamara.luarasi@uet.edu.al

Tomi Thomo, Agim Kasaj and Erjon Baboci
Department of IT, Mathematics and and Statistics
European University of Tirana
Tirana, Albania
agim.kasaj@uet.edu.al

Abstract **- The new technology supports new business scenarios or new implementations for the existing ones. The paper describes an implementation of B2B scenario. The Google cloud platform as service makes possible a digital broker system between businesses linked by interests. Architecture of this digital broker consists of common Cloud SQL database, hosted on Google platform, and a Google App Engine application. The users will use the web application from a browser or from small devices through an Android application. In both cases the system will have a user interface for updating the common database and setting the queries over it. As example of a business pair is considered the pair farmer-market, which is very much in need for communication especially on difficult rural zones. The paper represents the methodology used for the digital broker creation**.

Keywords **- *digital broker, cloud computing, web services, mobile applications.***

I. INTRODUCTION

We observe a huge demand for situational and ad-hoc applications desired by the mass of business end-users that cannot be fully implemented by the IT departments[1]. A typical case is the solution to support the individual businesses in difficult rural zones and their need for communication and information about the market and its needs. These solutions would be specific to various ad-hoc B2B scenarios. The broker facilitates transactions between sellers and buyers, generally without ever owning what is being sold and profiting from a charge on top of the sale price [2]. The case study that is considered here is a digital broker system that serves to a pair peasant-market. The peasant needs to know the markets, and the best one to sell the products. The best way for him is to have this information by a smart-phone or iPhone. Darin Newsom says that nearly 20 percent of survey respondents said they use grain tracking software as part of their operation. Seventy-eight percent said they use a smartphone as their primary way of accessing market prices, and 50 percent said they use a tablet [3]. Fig. 1 represents a series of appropriate steps in android smart-phone, to be used by the peasant from one side (client 1) and by a market on the other side every day. A button on the screen will display a list of the available markets (for the peasant) associated by some characteristics, and clicking on each of them, some additional details will be shown. For each of them, the quantity of the product that can be sold is sent by the peasant and this request is asked to be approved by him. On the other side the market can see all the requests and approves some by pressing a button. The approvals from both sides update a common database, in the way that after each approval the information that both sides see, is updated.

II. CLOUD COMPUTING

Cloud computing is a new technology that allows users to process data over the Internet in real time. The combination of trade liberalization and technological innovation are used to be seen as mutual friends that stimulate economies to specialize more every time in higher-skilled labour exports. While technological innovation kills some jobs, it also creates new and better ones [4].

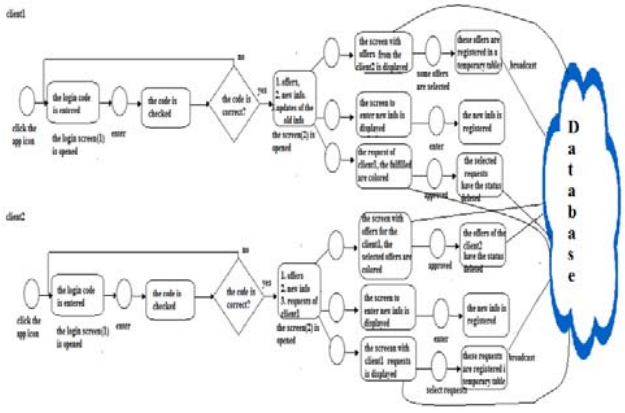

Fig. 1 The B2B scenario activity.

Cloud computing is very supportive for an individual to do its own business without the need to purchase hardware and software. Cloud Computing is on-demand access to a shared pool of computing resources. It helps consumers to reduce costs, reduce management responsibilities and increase business agility [5]. A cloud service brokerage serves as an intermediary between you and your cloud service provider by aggregating multiple cloud services, integrating them with in-house apps and customizing them to meet your needs [6]

Several cloud platforms offer their infrastructure and their services platform freely, thus allowing the individuals to become digital brokers. This new technology brings a transformation in the way how the people work. An

978-1-4799-6487-1/14 $31.00 © 2014 IEEE
DOI 10.1109/AMS.2014.17

important part of the work is to be aware of this technology. There are several frameworks that we can choose to achieve our goal and we have to know them, to compare them and to integrate all the necessary components for the chosen framework. The integration of the existing platforms with the services, the accomplishment of the necessary configurations is the essential part of the work, because this part defines the development process of an application. In this study we have chosen the Google platform.

III. RESEARCH FRAMEWORK

The framework that is chosen is the Google framework, Fig. 2. Gone are days of manually configuring and load-balancing your servers based on traffic. A major advantage of App Engine is that it automatically scales your application for you [7]. An App Engine application is deployed on Google and we have an internet address to use it. This application can be used from an Android device or from a web browser. The App Engine application provides the connection with the cloud database.

The development of such a system has two aspects. One of them is a series of steps on Google Console side, and the other is the development of a Google Web Application on Eclipse and the deployment of this application on Google.

Another work on the Google side is the creation of a Cloud SQL instance, its configuration and the linkage of this instance with the Google App Engine identifier.

From the Google Console we create a starting project and define some settings by a Wizard. Two components are defined for this project: a Google App Engine component [8], and a Google Cloud SQL component [9]. The last one includes the Google App Engine Component identification in his configuration.

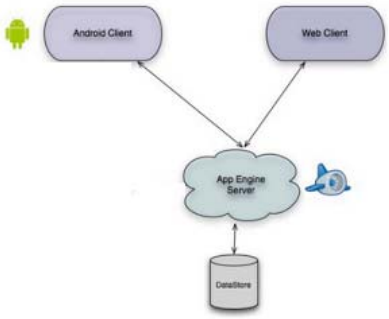

Fig. 2 The Framework

On the other side, that is, on the development environment, Eclipse in our case, some other configurations are done to provide the connection with the console Google project components and Google API. In Eclipse we have included first of all the Google plugins, which make possible the development of a Google web application which is part of Google App Engine.

IV. DATABASE MODEL

The database model is presented in the Fig 3. The main entities shown are the Offer (offers and requests), the Market-Info, the User-Info, the Operations, and some other entities are the Location (location for the user and for the market), Offer's Type, and the user's Activity. An operation is linked with an offer or request and with a user. These relations between the entities justify the following schema.

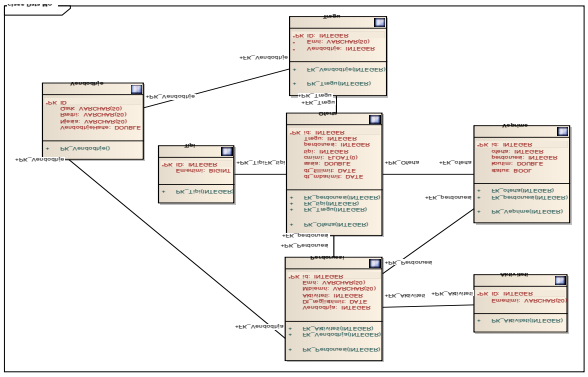

Fig. 3 The database model

V. RESEARCH METHOD

A. The AppEngine application

The servlet technology is used for the Google web Application, which is deployed later on Google. The **property** option of the Eclipse project, and **Google** option inside it, makes possible the inclusion of the Google App Engine Component ID inside the Eclipse project and the connection with the database. At the beginning we connect the web application with a local MySQL database. Inside the application the connection with the local database is provided by:

```
Class.forName("com.mysql.jdbc.Driver")url =
"jdbc:mysql://127.0.0.1:3306/database_name?user=root";
```

The testing process includes the running of the application in Eclipse as a web application, and from the browser we use the address http://localhost:8888/googleproject, where googleproject is the name that identifies our servlet in the web.xml file in our project. When we have tested everything with a local database, we replace the local database with the Cloud MySQL database. Meanwhile, the Google Cloud SQL instance console gives us the possibility to create our database, which was tested locally. Inside the code the connection will be as follows:

```
Class.forName("com.mysql.jdbc.GoogleDriver");
url="jdbc:google:mysql://project-ID:SQL-component-
ID/database_name?user=root";
```

Where project-ID is the project identification on Google side, and SQL-component-ID is the identification of Cloud SQL component on Google site too. From this moment we are able to deploy our web application in Google, and this means that we can use now our application from the browser with a URL address that Google provides us in the moment of the Google App Engine component creation.

All this process provides the web application which can be considered now as a Google service and make us able to use a Cloud database for which we are interested.

We can use this service with a URL address from a browser, or by a smart-phone. The architecture of the web application corresponds to the Fig 4.

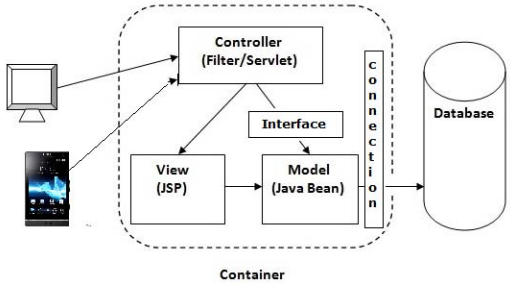

Fig. 4 The web application architecture

```
public class Parameters {
   public String[] parameters;
   public StringBuffer strError=null;

   public  Parameters (String[]parameters) {
      if (validate(parameters)){
     this.parameters = new
              String[parameters.length];
     for(int i=0; i< parameters.length;i++){
        this.parameters[i] =
                    parameters[i];}}
      }
   public boolean validate(String[] fields) {
     strError = new StringBuffer();
     for(int i=0;i<fields.length;i++){
        if(fields[i] == null){
          strError.append("parameter
               "+(i+1)+" is missing");
         return false;}
       if(fields[i].equals("")) {
            strError.append("parameter
                "+(i+1)+" is empty");
       return false;}    }
        return true;
  }
  public String[] getParameters() {
     return parameters;}
  public String getErrorString() {
     return strError.toString();}
  }
```

Fig. 5 The class Parameters

The model in Fig. 4 represents a set of classes, which correspond to the database tables and Android Screens. The last ones manage the different user operations. There is a communication by parameters between android and servlet, or controller. To generalize this communication we have used a separate class of Parameters, Fig. 5, where we deal with parameters that come from the Android. They are accepted and validated. All the screen classes will be extensions of the class Parameters.

The Android Screen classes manage the operations of each screen. For example the class that manages the login operation, or answers to the first screen, Fig. 6, just performs a selected statement in response to the parameter value like below, where User class is a class that corresponds to the database user table:

```
public class Screen1 extends Parameters {
   private User user;
   public Screen1(String[] parameters){
      super (parameters);}

   public User getUser() {
      return user;    }
   public boolean getInfo() {
      try {
       ConnectionDB.connection();
       String querySQL =
            "SELECT * FROM user WHERE ID =" +
       Integer.parseInt(parameters[0])+ "
            AND password = '"+ parameters[1]+"'";
       Statement stmt =
            ConnectionDB.conn.createStatement();
       ResultSet rs =
            stmt.executeQuery(querySQL);
       boolean querySuccessful;
       if(rs.next()) {
         user= new User(rs.getInt(1),. . .);
         querySuccessful = true;
       } else querySuccessful = false;
       ConnectionDB.conn.close();
       return querySuccessful;}
   catch (Exception e) {
       e.printStackTrace();
       return false;}   }}
```

Fig. 6 The screen1 class

The servlet, Fig. 7, plays the role of a controller that manages all the communications with the other classes sending there the parameters from the Android application. The results that come from the operations of screen classes will be sent into Android Application.

```
public class BrokerwebappServlet extends
                              HttpServlet {
   String[] parameters;
   Screen1 screen1;
   User user;

   public void doGet (HttpServletRequest request,
                HttpServletResponse response) {
   char screen =
       request.getParameter("screen").charAt(0);
   parameters =
       request.getParameterValues("parameters");
   switch(ekrani){
   case '1': // screen 1
     screen1=
       new Screen1(parameters);
     if (screen1.validate(parameters)){
    if(screen1.getInfo())
      user = screen1.getUser();
       writeObject(user, request,response);
     } else{
    writeObject(screen1.getErrorString(),
       request,response);}
     break;
     . . .
     default:
   }
}
void writeObject(Object obj, HttpServletRequest
   request, HttpServletResponse response){
   try {
      ObjectOutputStream oos = new
         ObjectOutputStream(
            response.getOutputStream());
      oos.writeObject(obj);
      oos.flush();
      oos.close();
   } catch (Exception e) {
      e.printStackTrace();  }
   }
   public void doPost (HttpServletRequest
      request, HttpServletResponse response) {
     doGet(request, response);}
}
```

Fig. 7 The controller (the servlet)

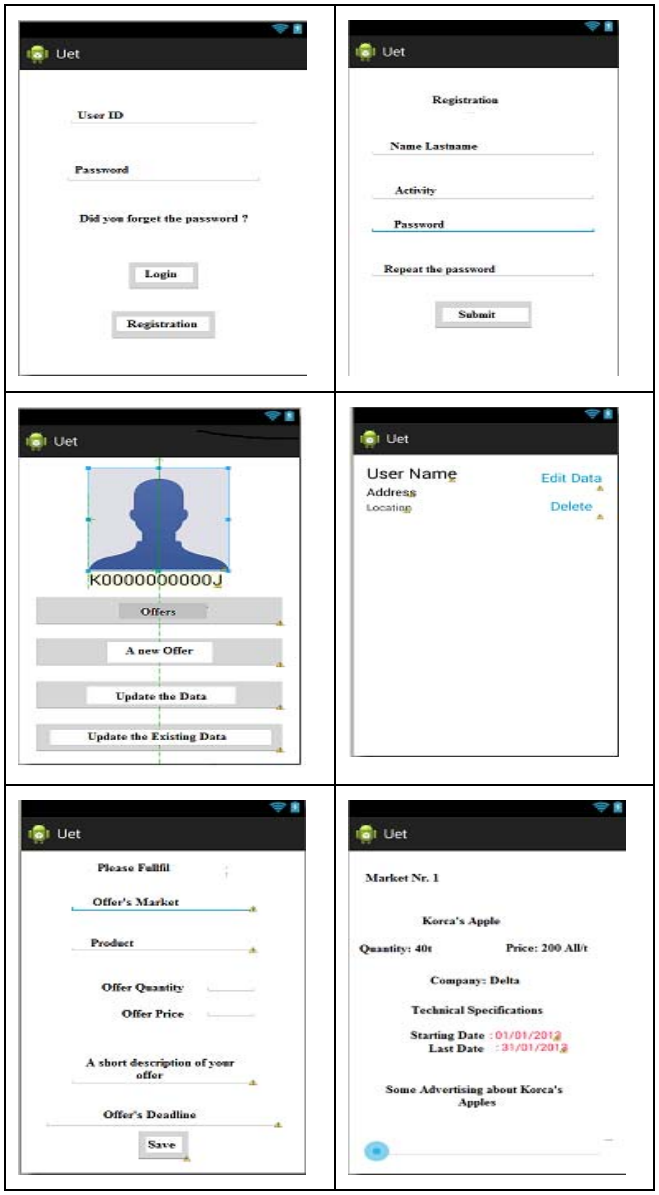

Fig. 8 Android Screens

The connection is managed by a separate class `ConnectionDB`

```
public class ConnectionDB {
   public static String url = null;
   public static Connection conn=null;
   public static void connection(){
      try {
       if (SystemProperty.environment.value()
       ==SystemProperty.Environment.
                    Value.Production) {
       Class.forName(
             "com.mysql.jdbc.GoogleDriver");
          url="jdbc:google:mysql:
           //uet-project-2014:
           uet-project-sql-2014/
           database_name?
           user=root&password=xxxx";
          } . . .
    } catch (Exception e) { . . .}

public static void close(){
   try {
     conn.close();
   } catch (SQLException e) {. . .}}}
```

Fig. 9 The class that provides the connection with database

VI. THE ANDROID APPLICATION

The web application can be used from a browser or from small devices like smart-phones, but as we mentioned, the smart-phone will be the most used by the end-user. We need to have specific Android applications that will be used by the both side of the B2B scenario. The Fig. 8 represents the Android screens of one side. We have used a few simple Android screens, considering the fact that the daily use of them to be as easy as possible for a user. A key point of the Android application is the communication with the web application. For this we use the class `AsyncTask` [10], which makes possible the handling of background threads, such as the communication with the internet.

The Fig. 10 shows the communication part of the Android application with and web application.

```
. . .
private class WebOperations extends
     AsyncTask<String, Void, String> {
   protected String doInBackground(
         String... urls){
      String response = "";
      for(String url : urls) {
        try{
          HttpClient client =
                new DefaultHttpClient();
          HttpContext context = null;
          String[]parameters =
                    new String[2];
               parameters[0]=
                    String.valueOf(
                       obj.getField1());
                             String.valueOf(
                       obj.getField1());
              . . .
          String URLparam=
                    parameters[0].toString()+"?";
          for(int i=1;i<
                    parameters.length-1;i++)
          URLparam+=" parameters=
                    "+ parameters[i]+"&";
          URLparam+= "parameters ="+
                   parameters[parameters.length-1];
          HttpPost post =
                    new HttpPost(URLparam);
          HttpResponse response =
                    client.execute(post);
          ObjectInputStream content=
                     new ObjectInputStream(
                       response.getEntity().
                           getContent());
          try {obj =
             (Entity1)content.readObject();
             } catch(ClassNotFoundException e)
                              {. . .}
      } catch(IOException e){. . .}}
          return response; }
  protected void onPostExecute(String result)
     {   txtText.setText(result);}}
. . .
```

Fig. 10 The communication part of the Android application with web application

VII. CONCLUSION

The situational and ad-hoc applications give solution to various B2B scenarios. A digital broker system would be a helpful business to establish connections between businesses linked by interests and which support various B2B scenarios. A methodology is given in this paper to create a digital broker system. This methodology is based on the recent technology. It is sufficient for one computer today to provide such a system. One App Engine application, one Cloud database and an Android application are the basic components of this system. This would be a helpful experience for every ID individual to create his/her own business.

REFERENCES

[1] Robert G. Siebeck, …, 2009, Cloud-based Enterprise Mashup Integration Services forB2B Scenarios

[2] Noren,E. (2013 January 14) Digital Business Models How Companies Make Money Online. *Ten Digital Business Models*

[3] http://openmarkets.cmegroup.com/7468/dtn-survey-a-glimpse-at-what-farmers-are-planning-for-2014

[4] Evert-jan Quak (2014 February 26) Revaluing labour, Views on job creation within economics
[5] Zaigham Mahmood, Richard Hill (2011) Cloud Computing for Enterprise Architectures
[6] John Moore, December 10, 2012
[7] By Kyle Roche, Jeff Douglas, 2009 Beginning Java Google App Engine
[8] https://developers.google.com/appengine/
[9] https://developers.google.com/appengine/docs/java/cloud-sql/
[10] Lars Vogel, 2013 Android Background Processing with Handlers and AsyncTask and Loaders
[11] http://developer.android.com/about/index.html
[12] https://developers.google.com/+/mobile/android/sign-in