A GCM Solution for Leveraging Server-side JMS Functionality to Android-based Trading Application

Claudiu VINȚE
Bucharest University of Economic Studies
claudiu.vinte@ie.ase.ro

The paper presents our solution for a message oriented communication mechanism, employing Google Cloud Messaging (GCM) on the client-side, and Java Message Service (JMS) on the server-side, in order to leverage JMS functionality to Android-based trading application. Our ongoing research has been focused upon conceiving a way to expose the trading services offered by our academic trading system ASETS to a mobile trading application based on Android platform. ASETS trading platform is a distributed SOA implementation, with an original API based on JMS. In order to design and implement an Android based client, able to intercommunicate with the server-side components of ASETS, in a manner consistent with publisher/subscriber JMS communication model, there was particularly necessary to have object imbedded messages, produced by various ASETS services, pushed to the client application. While point-to-point communication model could be resolved on the client-side by employing synchronous HTTP socket connections over TCP/IP, the asynchronously generated messages from the server-side had to reach the client application in a push manner.
Keywords: *Trading Technologies, JMS, GCM, Android Bridge Servlet, MOM*

1 Introduction

Messaging and message oriented middlewares (MOM) have revolutionized, over the past decade, the software solutions that can be delivered through service oriented architectures. Java Message Service (JMS) interface, in particular, has proved to offer the most flexible and opened manner for designing complex distributed software systems, ranging from social intercommunication applications, to enterprise resource planning, internet banking, trading, and generic data dissemination [1].

In tandem with the messaging solutions employed on the enterprise server-side, Android powered mobile devices have become increasingly popular, and the diversity and the complexity of applications that the platform supports, open novel perspectives upon the way distributed computing and information technology impact our daily life and businesses.

Our research project intends to expand the reach of ASETS trading platform toward the users of mobile devices powered by Android OS.

ASETS trading platform features a distributed architecture, service orientated, being implemented entirely in Java, and having an original application programming interface (API) designed and built upon JMS [2], [3].

Since the JMS libraries *jms.jar* and *imq.jar*, required to access the messaging interface that facilitates the communication with the message provider (OpenMQ), are not currently available for the Android platform, we needed to explore a different approach that would provide a seamless communication mechanism between a trading client, residing on a mobile Android device, and the ASETS trading platform.

ASETS platform offers a sophisticated API for supporting rich Java applet trading clients, and provides services such as:

- *Order Management Server* (OMS);
- *Portfolio Management Server* (PMS);
- *Exchange Simulation Engine* (ESE), which contains the order matching algorithm;
- *Pseudo Random Order Generator* (PROG), to create liquidity within the simulation market;
- *Delayed Data Feed* (DDF), for feeding the trading simulation platform with real

DOI: 10.12948/issn14531305/17.3.2013.05

world pricing data, captured from Bucharest Stock Exchange (BSE).

Such a JMS based API could not be exposed out of the box to an Android based trading GUI. There have been researches and developments prior Google Cloud Messaging for Android introduction, for interconnecting applications running on mobile devices with JMS based systems [4], [5].

In the context of targeting to design and build a trading client application for Android OS, Google Cloud Messaging (GCM) service offers a messaging mechanism with the following main characteristics [6]:

- allows third party application servers to send messages to Android-based applications;
- offers the ability to deliver messages to an Android application on an Android device, even if the application is not running - as long as the Android based application is set up with the proper broadcast receiver and permissions, the system will wake up the application via Intent broadcast when a message arrives;
- the messaging service does not provide any built-in user interface or other handling for message data; GCM simply passes raw message data received from the producer straight to the Android application which, subsequently, has full control of how to handle it.

As for the ASESTS Android-based trading application, the requirements for accessing the GCM service are as follows:

- the Android application identifies itself for registering to receive messages by using an *Application ID* (the package name from the manifest);
- when the Android application use for the first time the messaging service, it calls the GCM method `register()`, which returns a *Registration ID*; the ID issued by the GCM servers allows the Android application to receive messages; the Android application should store the *Registration ID* for later use (for instance, to check in method `onCreate()` if it is already registered; once the Android application has the *Registration ID*, this is sent to the ASETS server-side, which uses it to identify each device that has registered to receive messages for a given Android application; a *Registration ID* is tied to a particular Android application running on a particular Android device;
- in order for the Android application to register to the ASETS service responsible with the generation of the messages to be sent to GCM service, it needs to know the *Sender ID*, which is the identification of the third party server within the GCM service cloud.

2 The Architecture of the Proposed Solution

Taking into account all the above briefly introduced elements, there arose the necessity for conceiving a software component that would act as a bridge between the Android based trading application and the ASETS server-side, along with application programming interface designed to support the communication between the software bridge and the Android application. In addition, the existing *asets.api* interface had to be reorganized, in order to make a separation between the classes that implement the business payload to be carried by the messages, and the classes that are dedicated to handle the communication part:

- *asets.android.api*
- *asets.jms.client.api*
- *asets.jms.server.api*
- *asets.data.api*

The first three APIs are communication oriented ones, and were to be built on the top of *asets.data.api*, which implements the business objects employed by ASETS trading platform, as we briefly introduced the approach in [7].

GCM specifications state that every message sent in GCM has by default the following characteristics:

- a data payload limit of 4096 bytes.
- it is stored by GCM for 4 weeks.

DOI: 10.12948/issn14531305/17.3.2013.05

In GCM terms, our solution employs messages with payload, or non-collapsible message.

Unlike a send-to-sync message or collapsible message, where each new message replaces the preceding one, every non-collapsible message, or a message with payload, it is delivered individually. The data payload that the message contains can be up to 4kb. In order to specify a non-collapsible message, all we need to do is actually to omit the *collapse_key* parameter in the send request to GCM servers. Consequently, GCM will send each message individually. It has to be noted that GCM does guarantee the order of delivery for the messages it handles.

The other characteristic of a GCM message, the time to live (TTL) feature, lets the sender specify the maximum lifespan of a message using the *time_to_live* parameter in the send request. The value of this parameter must be from 0 to 2,419,200 seconds, and it corresponds to the maximum period of time for which GCM will store and try to deliver the message. Requests that don't contain this field are assigned the default TTL, which is the maximum period of 4 weeks. Since ASETS trading system functions with daily sessions, starting up every day at hours 00:05, we considered useful to specify the *time_to_live* parameter, and set its value to the number of seconds remained till hours 23:55 hours of the current day, when ASETS shuts down.

As previously mentioned, a *Registration ID* (*regID*) represents a particular Android application running on a particular device. Once an application has a *regID*, this does not need to be changed. The un-registration should be done only in the case that the user wants the trading application to stop receiving messages. The *regID* is not associated with a particular logged in user, but it maps an application to a device. Therefore, to unregister the application is not a mechanism for logout user, or for switching between users.

The premises are that the server that sends the GCM messages to the Android devices has to manage the mapping between users, the *regIDs*, and individual messages:

- the server-side should maintain a mapping between the current user and the *regID*; this should include information about which user is supposed to receive a particular message; the mapping has to be persisted on disk, so in the eventuality of a server crash it can be retrieved, for operational continuity, once the server is restarted;
- the trading application running on the Android device should check to ensure that messages it receives match the logged in user.

Taking into account the messaging framework that GCM provides, in connection with messaging model employed by the ASETS platform, the solution that we propose as the outcome of our research relies on three key components:

- an API for supporting the Android client side communication;
- the ASETS Bridge Servlet (ABS) which acts as an ASETS JMS client, residing within the Apache Tomcat web server, and creates the link between the Android-based trading GUI and ASETS server-side; the servlet employs methods from both *asets.android.api* and *asets.jms.api* - *asets.jms.client.api* in particular - since the new bridge component behaves as a client of the ASETS server-side resources;
- Android-based ASETS trading GUI, the client application that provides trading capabilities from an Android powered mobile device to ASETS platform.

The overall architecture of our proposed solution is presented in Figure 1.

ASETS *Bridge Servlet* listens for the Android client HTTP requests, placed on the web server, and then converts these requests to JMS based requests that a forwarded to the corresponding ASETS services.

The entire request flow from the Android trading application to the ASETS Bridge Servlet is handled through HTTP socket connections over TCP [8].

DOI: 10.12948/issn14531305/17.3.2013.05

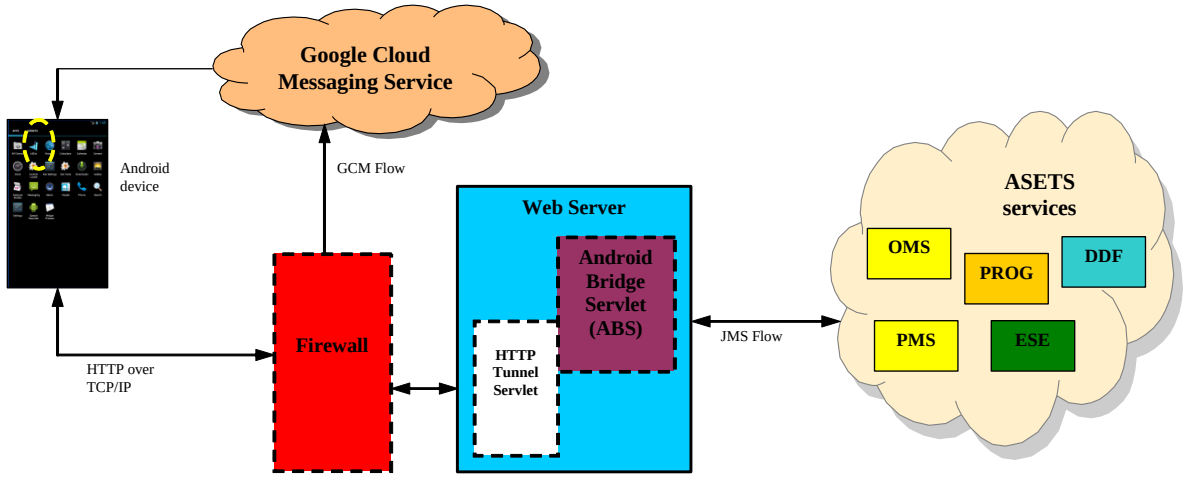

Fig. 1. The overall architecture of the proposed solution

The requests are not resolved by ABS synchronously, it returns to the Android application only a synchronous acknowledgment, informing it that the request was successfully received and process. The flow of replies from the ASETS Bridge Servlet is entirely asynchronous. The actual data reply is transmitted asynchronously to the Android client application via GCM service, by a *push* mechanism [9].

There are the following scenarios that we identified and conceived solutions for:

i. HTTP request with asynchronous GCM reply with message having a data payload under 4Kb – simulates a *point-to-point* (*p2p*) asynchronous request-reply;
ii. HTTP request with asynchronous GCM reply with message having a data payload over 4Kb;
iii. message push to the client Android application of published JMS message via GCM, with payload under 4Kb – simulates the JMS publishing mechanism for a previous subscription recorded on the server-side, via a HTTP request received from the Android application;
iv. message push to the client Android application of published JMS message via GCM, with payload over 4Kb.

We shall explore in details the steps that are to be followed in each scenario.

In the case of a simulated *p2p* asynchronous request-reply, when the reply message that has to be sent from ASETS Android Servlet has less than 4Kb of data payload, the process implies:

1. a HTTP request placed by the Android client application (JSON formatted message);
2. the JSON message received by ABS is un-marshalled, and a JMS request with reply it tis sent to the corresponding ASETS service on the server-side;
3. the JMS reply is received by ABS and the data is marshalled to a JSON formatted message;
4. if the JSON formatted message has presumably less than 4Kb of payload, then the entire message is delivered to GCM servers;
5. on the Android device, GCM service receives the message and passes it to the corresponding client application.

The mechanism is depicted in the Figure 2.

DOI: 10.12948/issn14531305/17.3.2013.05

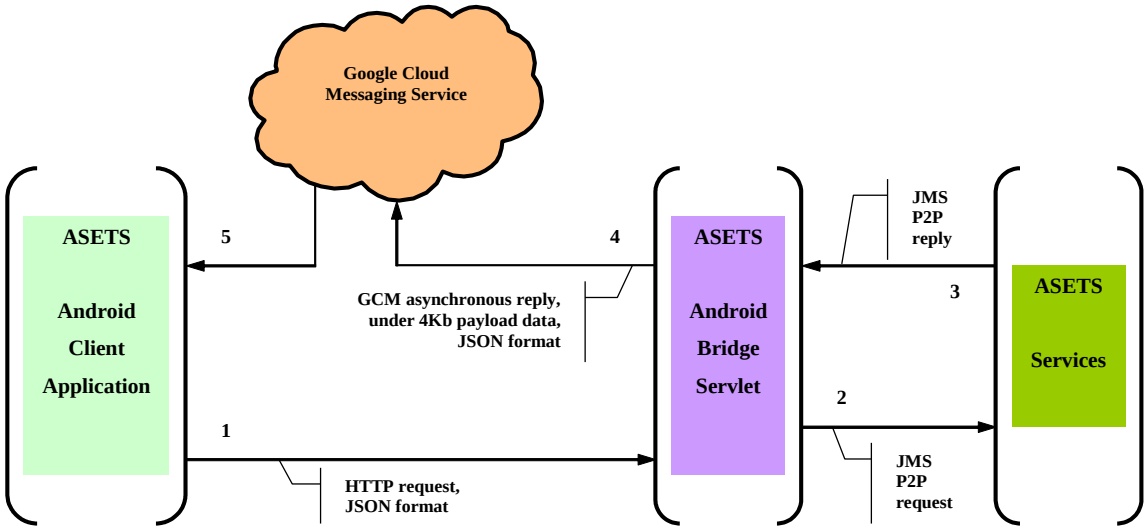

Fig. 2. Asynchronous reply through GCM for data payload under 4Kb

If the reply message that has to be sent from ASETS Android Servlet has more than 4Kb of data payload, the process implies the following additional steps, Figure 3:

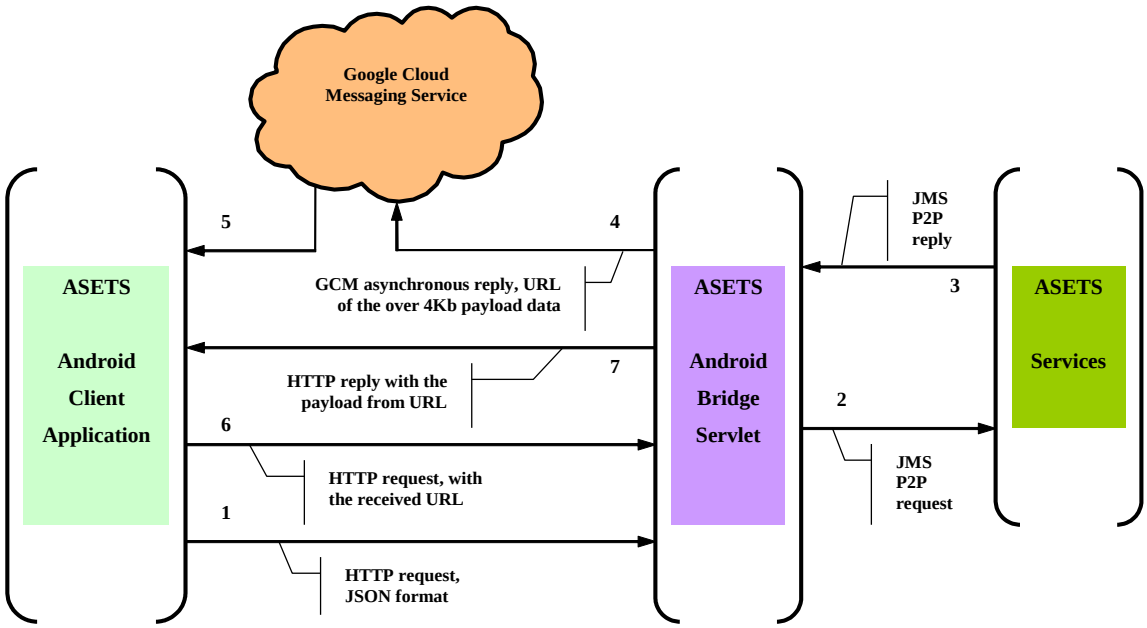

Fig. 3. Asynchronous reply through GCM for data payload over 4Kb, followed by a synchronous HTTP request-reply

4. if the JSON formatted message has more than 4Kb of payload, then the data is saved at a temporary URL, and the link is placed in the GCM message instead of the actual reply data; the JSON formatted message containing the URL it is sent to GCM servers;
5. on the Android device, GCM service receives the message and passes it to the corresponding client application;
6. the Android trading application uses the URL received via GCM service for creating a new HTTP request to the ASETS Bridge Servlet, this time expecting to receive synchronously the data stored at the transmitted link;
7. ABS replies synchronously at the HTTP request with the data stored at the received URL; once the reply is sent to the

DOI: 10.12948/issn14531305/17.3.2013.05

Android client, the servlet deletes the data and the temporary created URL.

One of the particularities of a trading system is that, for example, when a client order is placed on the market it may not be executed right away and, therefore, the mechanism for acquiring updates in the client trading application regarding the state of the order cannot rely on the *p2p* request-reply model. A pure JMS client of the ASETS platform can simply subscribe at a certain topic, in order to receive the subsequent order updates from the ASERT market. In the case of an Android trading client, we need first to subscribe to the JMS topics of interest, following a process which involves:

- a HTTP request from the Android application to the Android Bridge Servlet, regarding the topic it intends to subscribe to;
- ABS will convert the Android client request into a JMS subscription to the topic received in the request.

From that point on, the mechanism of sending ASETS server-side updates to the Android trading application it is presented in Figure 4.

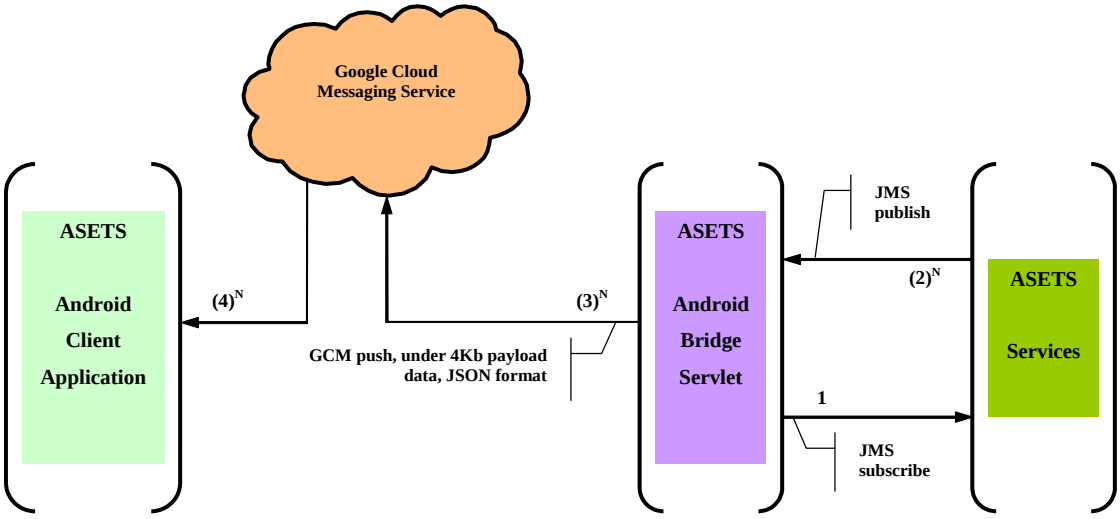

Fig. 4. Message pushing through GCM for data payload under 4Kb, followed by a synchronous HTTP request-reply

1. ASET Bridge Servlet subscribe to the topic of interest, specified in the HTTP request received from the Android trading application;
2. whenever a updated is generated by the market and published at the topic concerned, the ABS captures the JMS published message and marshals it to a JSON format message to be sent to GCM servers;
3. if the JSON formatted message has presumably less than 4Kb of payload, then the entire message is delivered to GCM servers;
4. on the Android device, GCM service receives the message and passes it to the corresponding client application.

It has to be noted that there may be multiple published messages and hence the presence of **N** superscript at the steps that correspond to the pushing mechanism and flow.

If published JMS message marshalled JSON format is has a data payload over 4Kb, then pushing process involves some additional steps, as Figure 5 shows.

If the JSON formatted message has more than 4Kb of payload, then the data is saved at a temporary URL, and the link is placed in the GCM message instead of the actual reply data.

DOI: 10.12948/issn14531305/17.3.2013.05

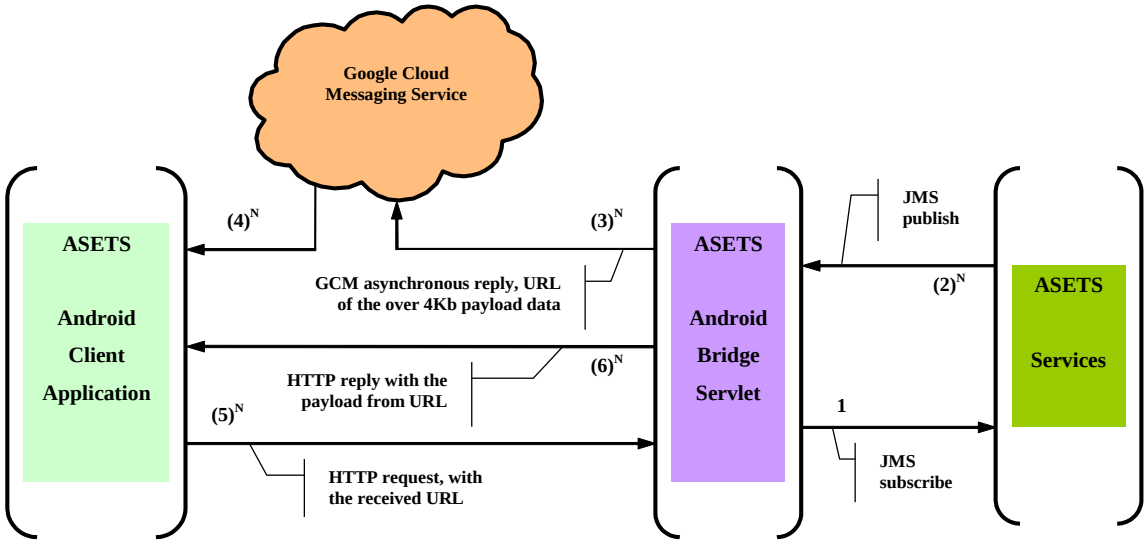

Fig. 5. Message pushing through GCM for data payload over 4Kb, followed by a synchronous HTTP request-reply

Later on, the Android trading application uses the URL received via GCM service for creating a new HTTP request to the ASETS Bridge Servlet, this time expecting to receive synchronously the data stored at the transmitted link.

As noted above, there may be multiple published messages and hence the presence of **N** superscript at the corresponding steps.

3 The Android-based Trading Application

ASETS Android application is designed and implemented using the *model-view-controller* architectural pattern. The graphical interface of the application supplies direct access to the main screens that put the user in control over his or her trading activity:

- *Orders* tab – offers access to the Orders and Executions panels, which list the investor's orders placed during the current trading session, along with their associated executions;
- *Instruments* tab – provides the tradable financial instruments on the ASETS market;
- *Portfolio* tab – displays the portfolio of financial instruments currently owned by the investor;
- *Trades* tab – offers access to the list of trades generated by the trading platform based on the investor's activity.

In order to offer the invertors a as close as possible trading experience that they enjoy using the ASETS GUI implemented as a Java applet which can be launched from the portal www.bursa.ase.ro, we designed the graphical interface for being rich enough in features, yet intuitive and simple to be used. Since on the Android platform the user interface is entirely built around a set of gestures executed on a touch screen, we had to design and implement a new trading application from the ground up.

In Figure 7 are shown the screens which are accessible through dedicated tabs, for offering information regarding: the list of tradable financial instruments on the ASETS market, the list of trades generated by the trading platform based on the investor's activity, and the portfolio of financial instruments currently owned by the investor.

For instance, the logging in procedure involves the transmission of a HTTP request, containing the previously registered user ID and password required for securely accessing the ASETS system [10]. Since the users' data is cashed by the ASETS *Bridge Servlet*, there correctness of the user credentials can be confirmed synchronously thought the same HTTP channel [11].

DOI: 10.12948/issn14531305/17.3.2013.05

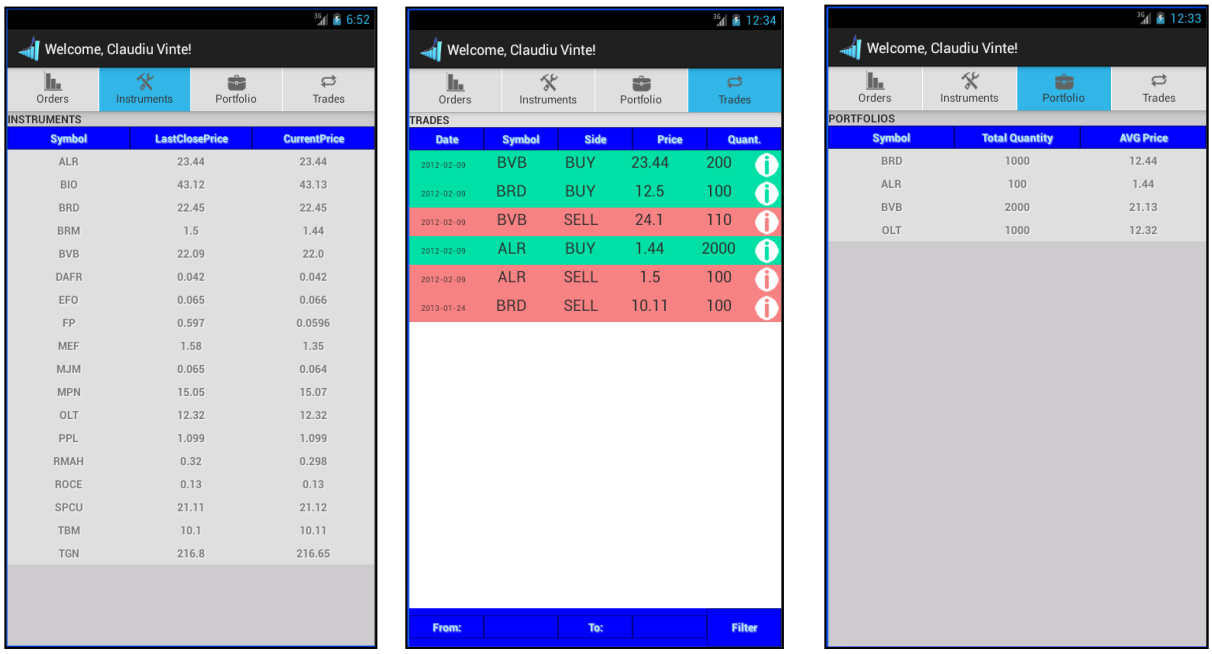

Fig. 7. Lists of instruments, generated trades, and investor's portfolio

In the Figure 6 below is shown the identification icon of the ASETS Android based trading GUI, along with the succession of screens associated with logging in procedure.

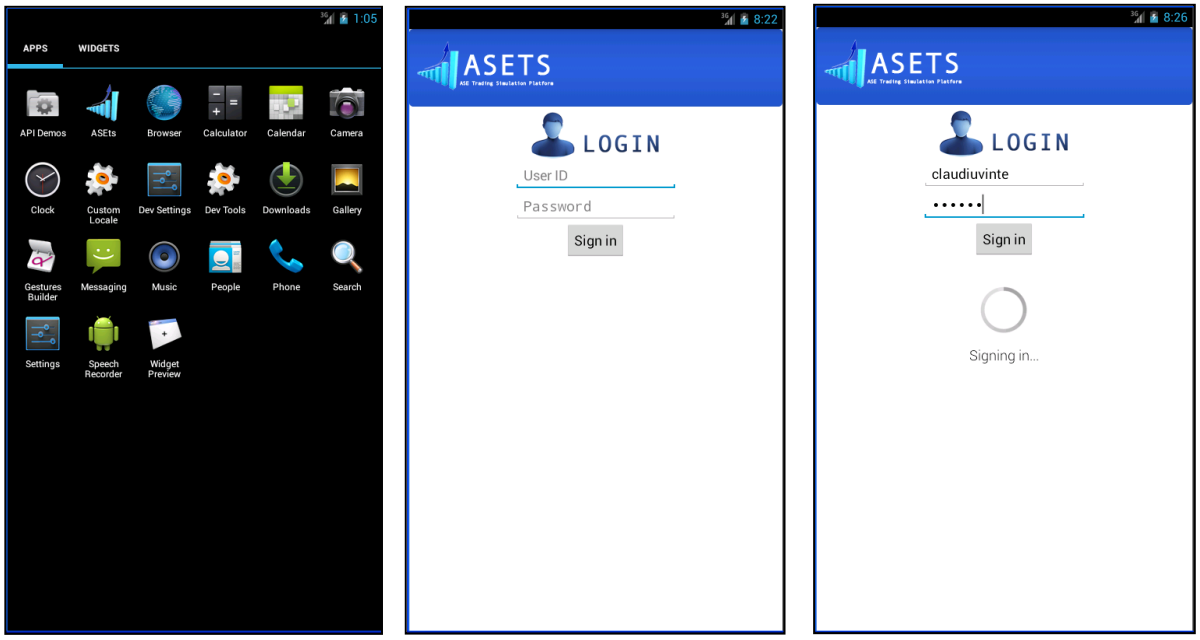

Fig. 6. ASETS Android based GUI identification icon, along with the logging in procedure

On the Android client-side application, both HTTP requests/replies and GCM messages are formatted using the lightweight data-interchange format JSON (JavaScript Object Notation). The Java objects created within the Android application go through a marshalling process and JSON formatted messages are generated prior being sent as requests to ASETS Bridge Servlet over HTTP. On the servlet side, the JSON formatted messages are un-marshalled, and the original Java object are recomposed and subsequently sent to ASETS services via methods provided by *asets.jms.client.api*.

In the context of a trading application in particular, one of the very useful features that

DOI: 10.12948/issn14531305/17.3.2013.05

GCM service provides, is the ability to deliver messages to an Android application on an Android device even if the application is not running. As long as the Android based application is set up with the proper broadcast receiver and permissions, the system will wake up the application via Intent broadcast when a message arrives.

In order for the ASETS Android application to register to the ASETS *Bridge Servelet*, it needs to know the *Sender ID*, which is the identification of the servlet within the GCM service cloud. The Android application identifies itself for registering to receive messages by using an Application ID (the package name from the manifest).

The following sequence of code shows how these constants, part of the GCM API, are to be employed by the Android client application.

```
public class AppConstants {
   public static final String
   GOOGLE_API_PROJECT_NUMBER_USED_AS
   _SENDER_ID = "460443047907";
   public static String DE-
   VICE_GCM_ID = "";
}

final String regId = GCMRegis-
trar.getRegistrationId(this);
if (regId.equals("")) {
   GCMRegistrar.register(this, Ap-
   pCon-
   stants.GOOGLE_API_PROJECT_NUMBER_
   USED_AS_SENDER_ID);
} else {
   System.out.println("Device al-
   ready registered");
   AppConstants.DEVICE_GCM_ID =
   regId;
}
```

The *String* returned by **GCMRegistrar.*getRegistrationId*()** it is an ID issued by the GCM servers to the Android application that allows it to receive messages. Once the ASETS Android application has obtained the registration ID, it sends it to the ASETS *Bridge Servelet*, which uses it to identify each device that has registered to receive messages for a given Android application [10]. In other words, through a registration ID is identified a particular ASETS Android application running on a particular device, and implicit a certain investor that uses ASETS trading platform.

As we briefly mentioned in the requirements for our solution, the ASETS Bridge Servlet has to maintain a mapping between the user of the request and the *regID*. The mapping has to be persisted on disk, so in the eventuality of server crash it can be retrieved, once the server is restarted, in order to offer operational continuity. The ABS ensures the persistence of users, requests and regIDs mapping by employing SQLite. In this way the servlet resorts to a reliable, standardised and light weight solution to persist data in a single disk file [12]. Furthermore, the format of the file is not platform dependent, ensuring the ASETS Bridge Servlet can be deployed along with Apache Tomcat webserver on different machines with a different architecture.

For adding an order to the market, the GUI provides a button marked with "+" signs and, by pressing it, a new window pops up and allows the user to enter the data for the new order, Figure 8.

ASEST *Bridge Servlet* is the key component of our proposed solution, and along with *asets.android.api*, integrates the ASETS Android application within the ASETS trading platform. The servlet receives HTTP requests in JSON format from the Android client and, from that point on, it behaves as a standard ASETS client, built with *asets.jms.api*. In addition to that is has to manage the connections from the Android devices, in order to be able to route back the JMS messages generated by the ASETS server side, converting them in JSON format and delivering the JSON formatted messages to GCM servers. The servlet has to maintain for each Android registered device the corresponding JMS connections to the ASETS platform. When a new JMS message is received from one of the ASETS service, let's say that an order matching occurs and a new execution is generated, this message has to be sent to the destination device via GCM service, regardless of having or not the Android application still running of that particular device.

DOI: 10.12948/issn14531305/17.3.2013.05

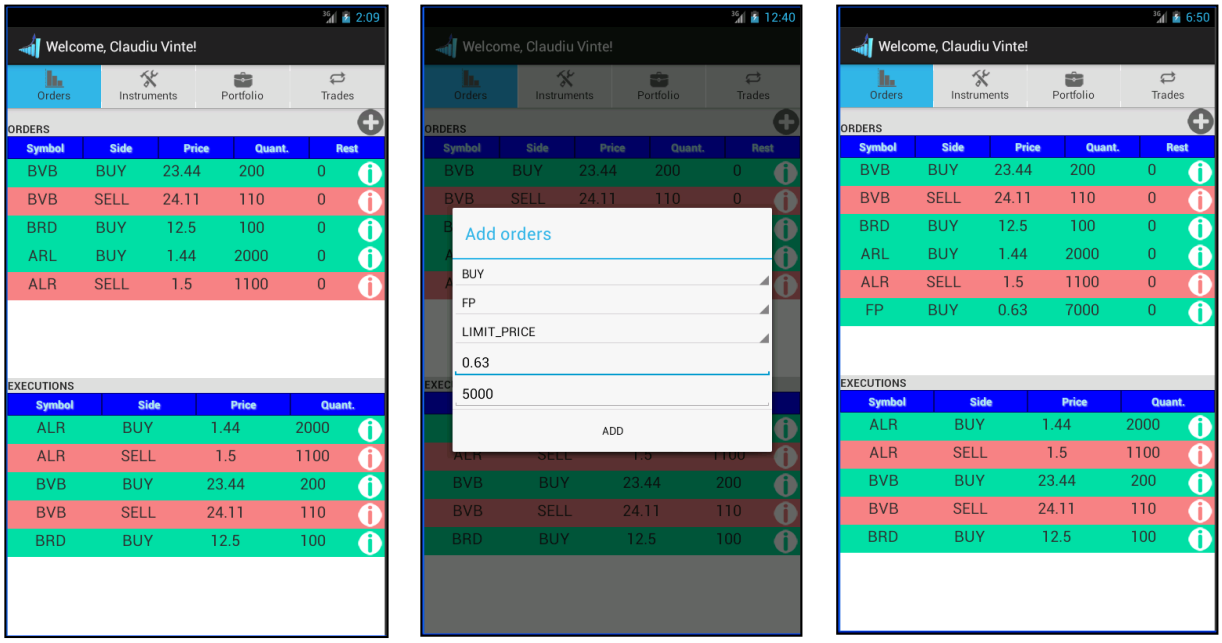

Fig. 8. Orders and Executions panels, and the procedure of adding a new investor's order

The Apache Tomcat web server creates a new thread for the received HTTP request. Since the ASETS *Bridge Servet* interacts with JMS provider through a single thread that deals with all Android based clients, there is necessary to synchronize the method that passes the HTTP request received by *doPost()* method, with the thread that manages the JMS connections.

4 Conclusions

In this paper we briefly presented the current results of our ongoing research focused upon conceiving and implementing a software solution for exposing the trading services offered by our academic trading platform ASETS, to mobile trading applications, based on Android platform. We proposed a communication mechanism which employs the services supplied by Google Cloud Messaging servers, and conceived an integrated solution to support ASETS Android based trading GUI. Our intention is to further explore the ability of employing GCM service for price data dissemination and market alerts.

In addition to the herein presented results, we are exploring the possibility of leveraging the ASETS server-side trading system functionality through a JavaScript web-based client, employing AJAX, which is an Asynchronous JavaScript And Xml mechanism for real time web applications [13]. The ActiveMQ message provider offers support for AJAX by the means of a servlet, AMQ AjaxServlet, which is designed to handle the JMS requests and responses straight in JavaScript client-side web application. The AMQ features are provided on the client side by the *amq.js* script, opening the possibility of creating a complex and highly real time web trading applications, taking full advantage of the publish/subscribe nature of ActiveMQ. This research path may potentially lead to a more generic solution for exposing the core functionality of ASETS trading platform to a broader range of platforms and mobile devices.

Acknowledgment

We would like to extend our thanks and appreciations to the following students of The Master Program in Economic Informatics, series 2012-2013, who had a significant contribution to the integration effort required by this research project – in alphabetical order: Cătălina Ioana BOGOȘ, Carmen Daniela BRIȘAN, Ștefania CIRIPIALĂ, Claudia-Anitta IVAȘCU, Dan-Marian MIRESCU, Stefăniță Alexandru MITRAN, Marius Constantin MOGA, Ioana MOLDOVAN, Radu

DOI: 10.12948/issn14531305/17.3.2013.05

NICULAE, Silviu NICOLESCU, Paul POTERAŞI.

References

[1] M. Richards, R. Monson-Haefel, D. A. Chappell, *Java Message Service (Second Edition)*, O'Reilly Media Inc., Sebastopol, California, 2009

[2] C. Vinţe, "Upon a Message-Oriented Trading API", *Informatica Economica Journal*, Vol. 14, nr. 1/2010, pp 208-216, ISSN 1453-1305, Available: http://revistaie.ase.ro/content/53/22%20Vinte.pdf

[3] C. Vinţe, "ASETS – An Academic Trading Simulation Platform", *Informatica Economica Journal*, Vol. 14, No 2/2010, pp 97-107, ISSN 1453-1305, Available: http://revistaie.ase.ro/content/54/10%20Vinte.pdf

[4] S. Mffeis, *Professional JMS Programming*, Wrox Press 2001, pp. 515-548, Available: http://www.maffeis.com/ articles/softwired/profjms_ch11.pdf

[5] E. Vollset, D. Ingham, P. Ezhilchelvan, "MS on mobile ad-hoc networks", Personal Wireless Communications, PWC Conference 2003, pp. 40-52.

[6] Google Cloud Messaging for Android, Internet: http://developer.android.com/google/gcm/index.html, as of August 1st, 2013

[7] C. Vinţe, "Integrated JMS-GVM Solution to Support Android Based Trading GUI", *Proceedings of the 12th International Conference on Informatics in Economy, IE 2013*, April 25-28, 2013, Bucharest, Romania, ASE Printing House, ISSN 2284-7472, ISSN-L 2247-1480, pp. 20-25.

[8] A. S. Tanenbaum, M. van Steen, *Distributed Systems - Principles and Paradigm*, Vrije Universiteit Amsterdam, The Netherlands, Prentice Hall, New Jersey, 2002, pp. 99-119, 414-488, 648-677

[9] Google Cloud Messaging APIs: http://developer.android.com/reference/com/google/android/gms/gcm/GoogleCloudMessaging.html

[10] C. Toma, M. Popa, C. Boja, "Mobile Application Security Frameworks" , *Annals of the Tiberiu Popoviciu Seminar – Supplement Romanian Workshop on Mobile Business*, vol. 6, 2008, Mediamira Science Publisher, Cluj-Napoca, pp. 79–93, ISSN 1584-4536

[11] A. S. Tanenbaum, *Computer Networks* - Fourth Edition, Vrije Universiteit Amsterdam, The Netherlands, Pearson Education Inc., Prentice Hall PTR, New Jersey, 2003, pp. 651-673, 772-784

[12] SQLite web resources: http://www.sqlite.org/

[13] ActiveMQ and Ajax web resources: http://activemq.apache.org/ajax.html

Claudiu VINŢE has over sixteen years of experience in designing and implementing software solution for trading systems and automatic trade processing. In 2007 Claudiu co-founded Opteamsys Solutions, a software provider in the field of securities trading technology and equity markets analysis tools. Previously, he was for over six years with Goldman Sachs in Tokyo, Japan, as Senior Analyst within the Trading Technology Department. Claudiu's expertise in trading technologies also includes working in Tokyo with Fusion System Japan, and Simplex Risk Management as Software Engineer, and Senior Software Engineer, respectively. Since 2009, Claudiu has been given lectures and coordinated the course and seminars upon *The Informatics of the Equity Markets*, within the Master's program organized by the Department of Economic Informatics. He holds a PhD in Economic Cybernetics and Statistics from The Bucharest Academy of Economic Studies. His domains of interest and research include combinatorial algorithms, agent-based simulation, middleware components, algorithmic trading and web technologies for equity markets analysis.

DOI: 10.12948/issn14531305/17.3.2013.05