Pushing dynamic and ubiquitous interaction between services **L**everaged in the Future Internet by **A**ppl**Y**ing complex event processing

Project number:	258659		
Instrument:	Collaborative Project		
Thematic Priority:	ICT-2009.1.2: Internet of Services, Software and Virtualisation		
Start Date:	01/10/2010	**Duration:**	36 months

D6.3.1 - DEMO environment for the telecom use case

WP6	Trials

Due date:	M20
Submission Date:	19/06/2012
Organization Responsible for the Deliverable:	FT
Version:	V1.5
Status:	Ready for submission
Abstract:	This deliverable aims at describing the demo environment (smartphones and simulator) deployed to get the Telecom Use Case operational and connected to the PLAY platform.

Author(s):	Philippe Gibert, Osvaldo Cocucci,	FT
Reviewer(s):	Života Janković, Dušan Zirojević	CIM

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission)	

* - put **X** in proper field

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	30/04/2012	First ToC	FT
0.2	14/05/2012	Added main chapters	FT
0.3	14/05/2012	Added push + clic2kcall sections	FT
0.4	15/05/2012	First Draft	FT
1.0	18/05/2012	First readable version, section tests to be provided,	FT
1.0	23/05/2012	Section 4 modified	FT
1.1	24/05/2012	Internal review	CIM
1.2	25/05/2012	CIM review + comments included 2.2.4 updated, Section 4 becomes Section 5, Added Section 4 Connections with the PLAY platform (to be inline with Armines)	FT
1.3	26/05.2012	Section 2.2.8 added	ICCS
1.4	28/05/2012	Internal review	CIM
1.5	4/06/2012	CIM review + comments included Added 2.2.5 Adding New Event type Section 6 Installation Added	FT

Table of Contents

Figures

Tables

Executive Summary

This deliverable describes the demo environment for the Management of Missed Calls scenario, the involved components and the proposed testing and validation steps. A quick installation manual is also provided.

The goal of this use-case is to mix Telco Events from smartphones and Web 2.0 information from Social Networks through an Android Application called Missed Calls Manager (MCM). The MCM is implemented through Android Applications running on smartphones, a simulation application hosted in a Siafu simulator [5] and Web services managing the interaction between the PLAY platform DSB and the Orange back-ends.

In order to test and validate the use-case, unit testing and validation testing plans are described. The execution of these tests and validation plans lead to an operational use-case connected to the PLAY platform.

1 Introduction

1.1 Purpose

The purpose of this deliverable is:

- To detail the applications and components involved in the *Missed Call Manager* (MCM) scenario,
- To describe the demo environment for the Telecom use-case, focusing on this scenario,
- To describe the tests validation plan and installation of the software application.
- To describe a quick installation manual.

This document will deliver a concrete and detailed view of the MCM application and the interface of this MCM application with the PLAY platform.

1.2 Document outline

After a description of the MCM scenario in Section 2, Section 3 gives a technical overview of the demo environment. Then Section 4 describes the Connection with PLAY and Section 5 contains the description of the tests. Finally Section 6 gives an overview of the installation process on an Android Smartphone.

1.3 List of Acronyms

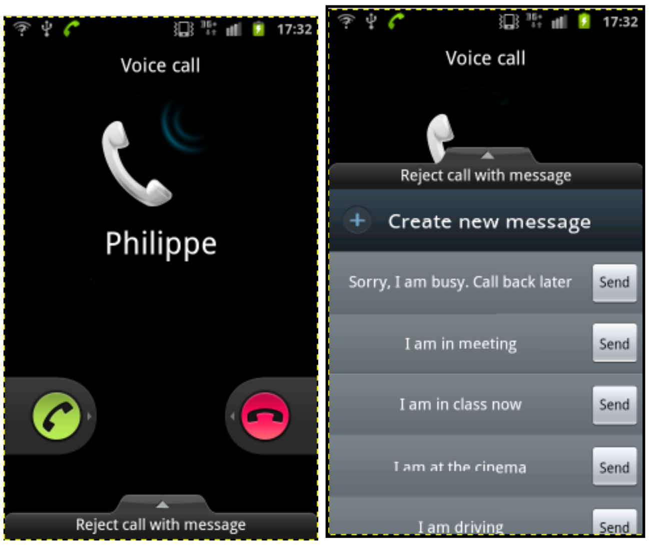
Acronym	Definition
DCEP	Distributed Complex Event Processing
CEPAT	Complex Event Pattern
GSM	Global System for Mobile Communications
SMS	Short Message Service
Outgoing Calls	Telephone calls dial by the Customer
Incoming Calls	Incoming Telephone calls for the Customer
Caller	The party that originates a call
Callee	The called party who answers a telephone call
MCM	Missed Call Manager (Application)
WSN	Web Service Notification

2 Scenario - Management of calls

2.1 Overview

This Management of Calls scenario is described in [4] paragraph 2.3 and is mainly implemented through Android applications that intercept missed *outgoing* and *incoming* calls and push event instances respectively *CallEvent* (Outgoing) and *CallEvent* (Incoming) to the PLAY platform. When the Application is running, *GeolocationEvent* instances, tracking information on user‘s mobility are also pushed regularly to the PLAY Platform.

More generally, the goal of this use-case is to mix Telco Events from smartphones and Web 2.0 information from Social Networks. The challenge for this use-case is to add value, mixing different event sources and combining event sources in a dynamic, not hard-wired way. Dynamic patterns taking into account customer’s mood and/or context and heterogeneous event sources are considered as well, not just telecom events sources. Telco Operators and smartphones providers already deliver some flavour of simple rule mechanism with automatic recall and SMS handling but the approach is static and hard-wired. See below the handling of an incoming call with the possibility to reject the call and send back an SMS (Android Gingerbread 2.3.5)



Sorry, I am busy. Call back later

I am in meeting

I am in class now

I am at the cinema

I am driving

Figure 1: Incoming call with built in reject

So, to deal with this challenge the approach is based on a clever middleware Platform containing a CEP engine. The following picture describes the main idea of the use-case.

- An Android application is deployed on a smartphone (MCM Application). This application intercepts events that occur in the smartphone. Some of them are Telco oriented (Incoming and Outgoing calls), others are Social Network oriented, such as Twitter and Facebook

- The issue is to mix these different event sources through a DCEP engine hosted in the PLAY middleware platform. The added value comes from rules and pattern (CEPAT) mixing these events, detecting complex situations and contexts and finally triggering actions (for example automatically putting users in relation or sending ‘New Event’ to the smartphone).

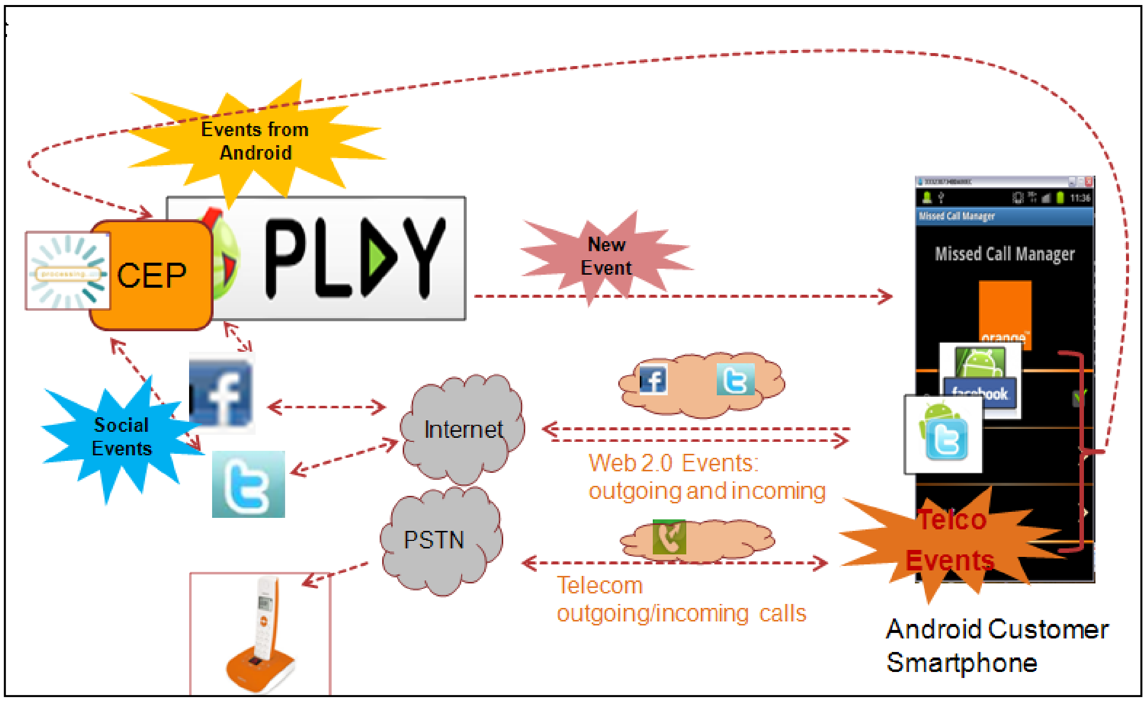

Events from Android
CEP
PLAY
New Event
Missed Call Manager
Social Events
Internet
Web 2.0 Events: outgoing and incoming
PSTN
Telecom outgoing/incoming calls
Telco Events
Android Customer Smartphone

Figure 2: Missed Call Manager Description

2.2 Missed Calls Management scenario (MCM)

2.2.1 Sub scenarios details

The use-case is oriented toward telephone Service available to the customer through the smartphone. The following assumptions are defined:

- The customer‘s smartphone is connected to GSM Service and Data Internet Service and it is receiving incoming calls or making outgoing calls from/to contacts,
- The customer has installed Twitter, Facebook and Google Latitude applications on his smartphone.

The use-case is divided itself in sub-scenarios listed below:

1. Find and start the best service to put Customer and friend in relation:
   a) The customer (Caller) calls a friend. The friend (Callee) does not answer for whatever reason. The PLAY platform is aware of this (after a sequence of calls) and detects that this friend is online on Facebook (or have sent recently tweets on twitter). PLAY informs him about this and automatically opens Facebook or twitter for the Caller,
   b) The customer calls a friend but the friend is located in a crowd of people (in a stadium) and so does not answer. The friend has not started Facebook or Twitter, but the Customer and Friend are using Google Latitude. After N missed calls, PLAY platform reacts and proposes to use Google Latitude to localize the friend, and traces the route between them.

2. Put Customer and friend in relation whatever the service:

The customer calls the friend N times with no answer. The friend has started his smartphone, but neither Facebook, nor twitter, nor latitude application. The PLAY platform tries to send 2 times an SMS with acknowledgment from the friend – no answer - then tries to send twitter messages – no answer - then the same for Facebook. Finally from the Customer contacts list, the email address for the friend is extracted and an email is sent.

3. Put Customer and friend in relation whatever the cellular Network status (overloaded):

The customer’s smartphone is located in an overloaded Cell (cellular Network) for voice and data. So he cannot normally make a call. The customer tries to call his friend but the network refused the call because of the overload network. The network informs the PLAY platform which sends an SMS to the customer with a proposal: For 0.30 €, the customer can make his outgoing call as soon as a time slot is available. These exchanges are done by SMS in order to cope with cell load. If he accepts, a server automatically put them in relation

2.2.2 MCM – Components

The following Figure highlights the different components involved in the MCM scenario:

- *MCM Caller*: this is the part of the application to be considered to have the role of managing outgoing calls,
- *MCM Callee*: this the called part of the application to be considered to have the role of managing incoming calls,
- *Siafu Simulator*: This application is useful for sending *OutNetwork* Events instances. In one sub-scenario, it is mandatory to send this type of Event for simulating overloaded cell areas,
- *PLAY DSB component*: this component, running on the PLAY platform subscribes incoming events from the use-case and from the platform and pushes them to subscribers. The DSB must be configured accordingly to push to/from the use-case applications,
- *CEP Engine component*: The component that receives and detects situations. CEPATs regarding the use-case must be provisioned in the DCEP engine ,
- *WS Orange push2Android Component*: This Web Service component implements the interface between the PLAY DSB ' world' and the Android 'world'. It receives notifications from the DSB and pushes them to the different Android instances (specific section on this, see 3.2.3),
- *WS Click2call component*: This Web Service component implements the interface between the PLAY DSB ' world' and the telephony 'world'. It is used to automatically create the call between parties (Caller and Callee).

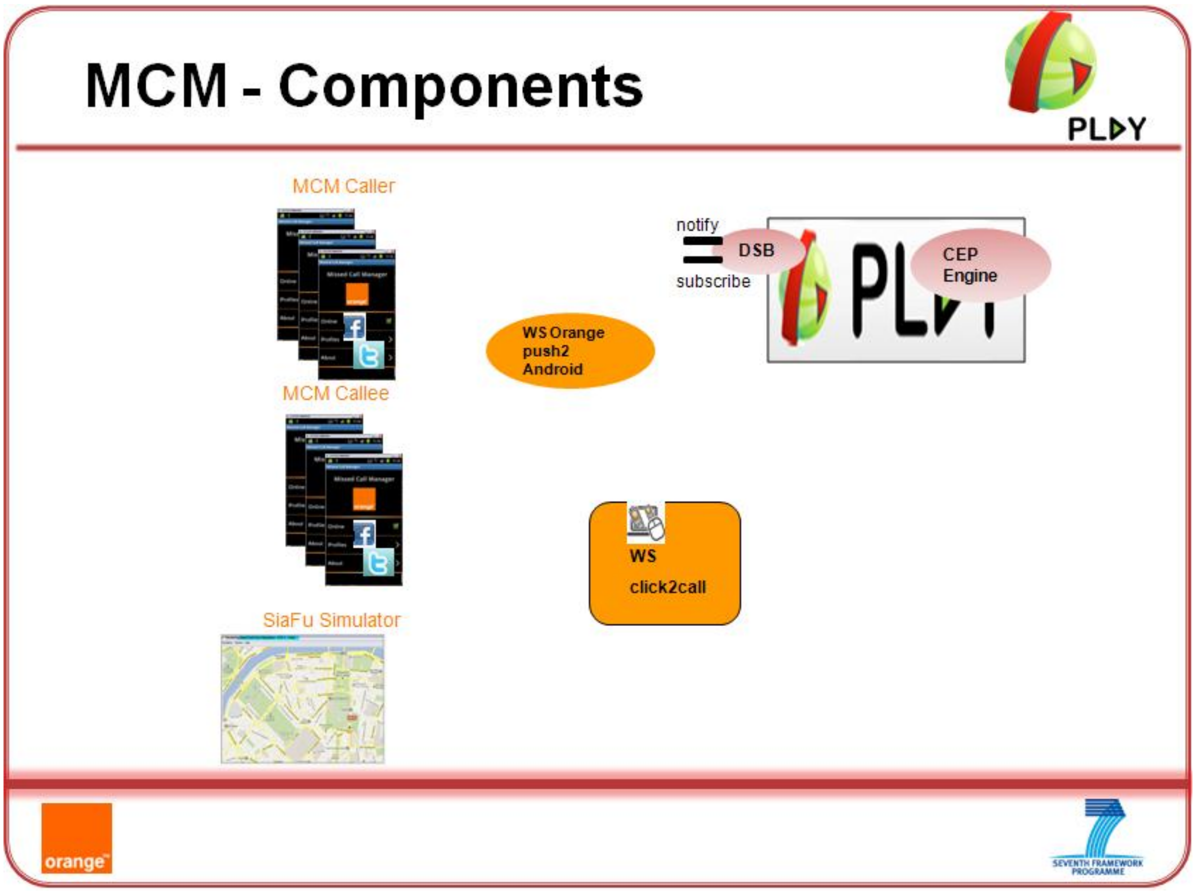


Figure 3: Missed Call Manager – Components

2.2.3 MCM - Events

Events are pushed from the MCM application to the PLAY 'world' thanks to the DSB component. Events come from smartphones and are pushed to the PLAY platform. They are classified as *Events In* from the platform point of view (Event Suppliers).

Events are generated from the PLAY platform and are pushed towards smartphones. They are classified as *Events Out* from the platform point of view (Event Consumers).

2.2.3.1 Events type

Thereafter the basic Hierarchy for Events In:

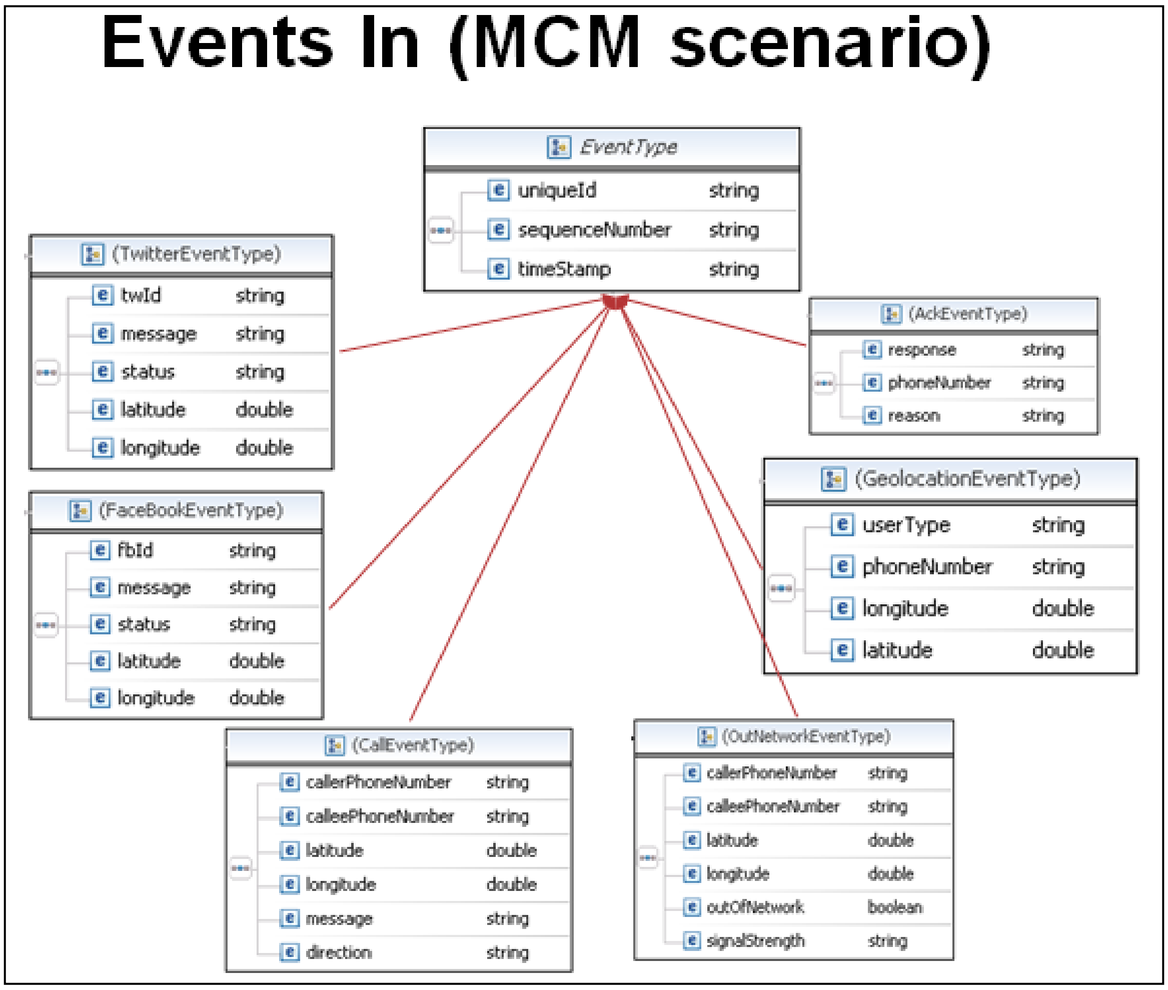


Figure 4: Events In Hierarchy

And the Hierarchy for Events Out:

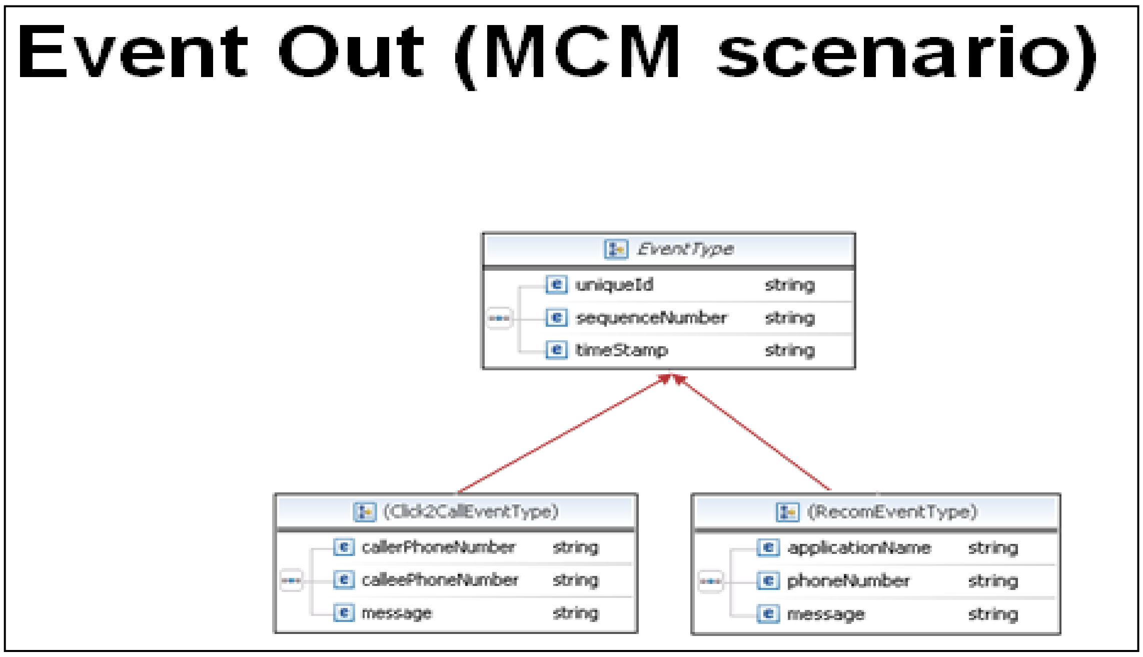


Figure 5: Events Out Hierarchy

2.2.3.2 Events from use-case to PLAY platform

This figure adds to the previous one (MCM components), the event types involved in the stream. The point of view is from the use-case to the platform.

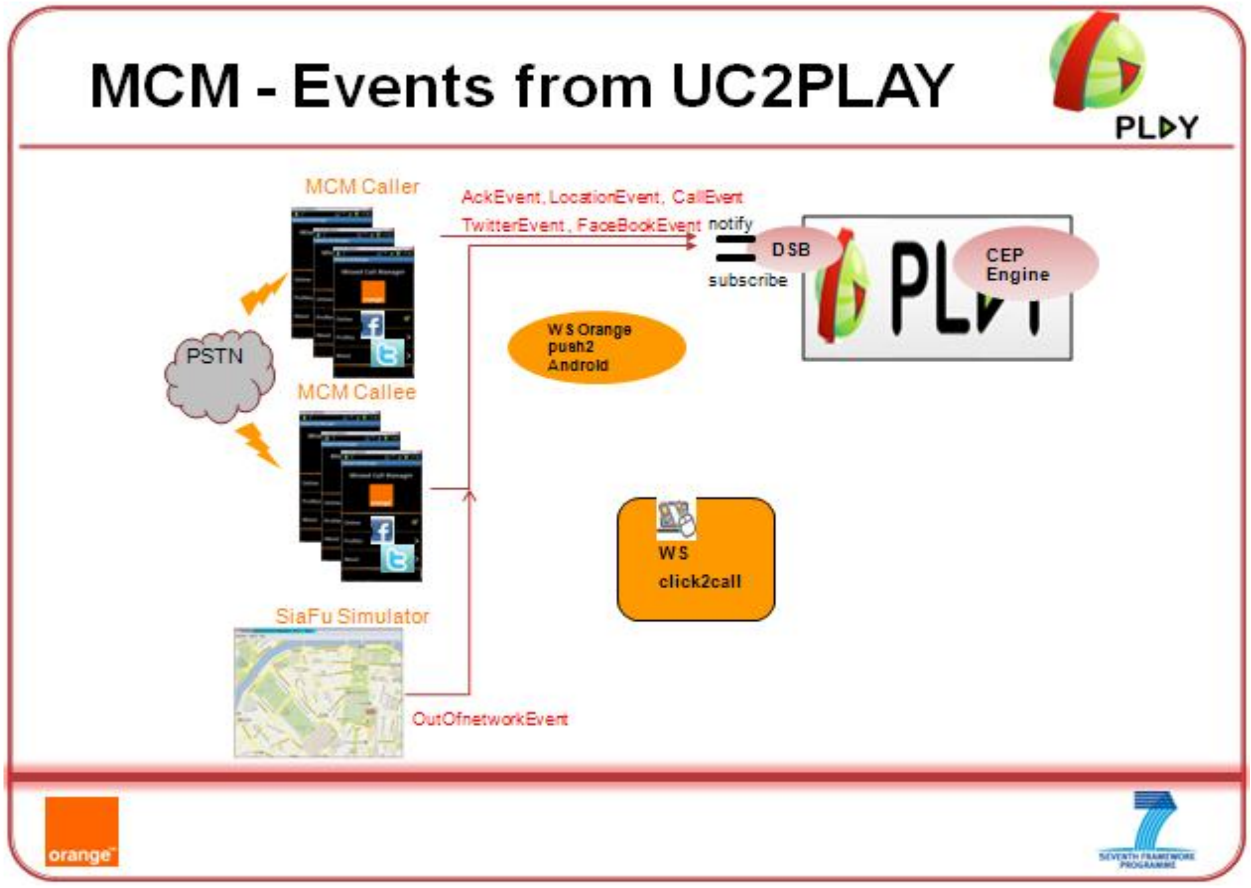


Figure 6: Missed Call Manager – Events from UC2PLAY

The involved Events are described in [6] and implemented in the Eclipse Project *commonDataTaxi* (see the PLAY Forge [7], ***inEvents*** Package).

The different event types pushed to the platform are:

- *GeoLocationEvent* : indicates regularly the position of the Caller and The Callee to the PLAY platform,
- *CallEvent* (Incoming, Outgoing): indicates a missed call to the PLAY platform with mainly Caller#, Callee#, location and direction of the Call,
- *TwitterEvent* and *FacebookEvent*: indicates that a Callee sent tweets or updates to his Facebook wall. Event instances signaling this are sent to PLAY Platform,
- *OutNetworkEvent*: indicates the cellular network status (overloaded, out of reach)
- *AckEvent*: it is sent in response to a *RecomEvent*, when the user acknowledges or not the proposed recommendation.

2.2.3.3 MCM - Events from PLAY platform to use-case

This figure describes the events involved in the stream from the PLAY platform to the use-case. The DSB component is involved making the interface with 2 Orange Web Services: *WS Orangepush2Android*, *WS Click2Call*.

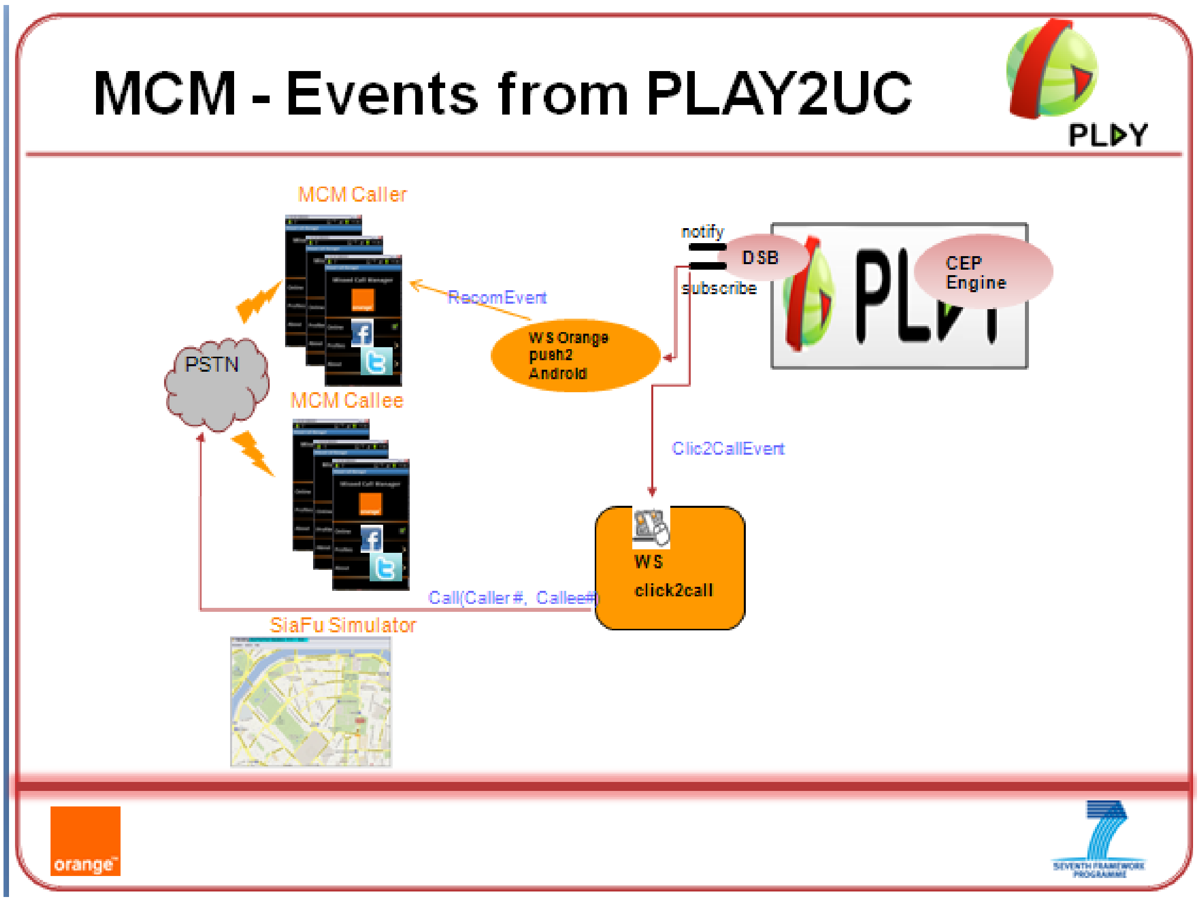


Figure 7: Missed Call Manager – Events from PLAY2UC

The involved events are described in [6] and implemented in the Eclipse Project *commonDataTaxi* (see the PLAY Forge [7], ***outEvents*** package).

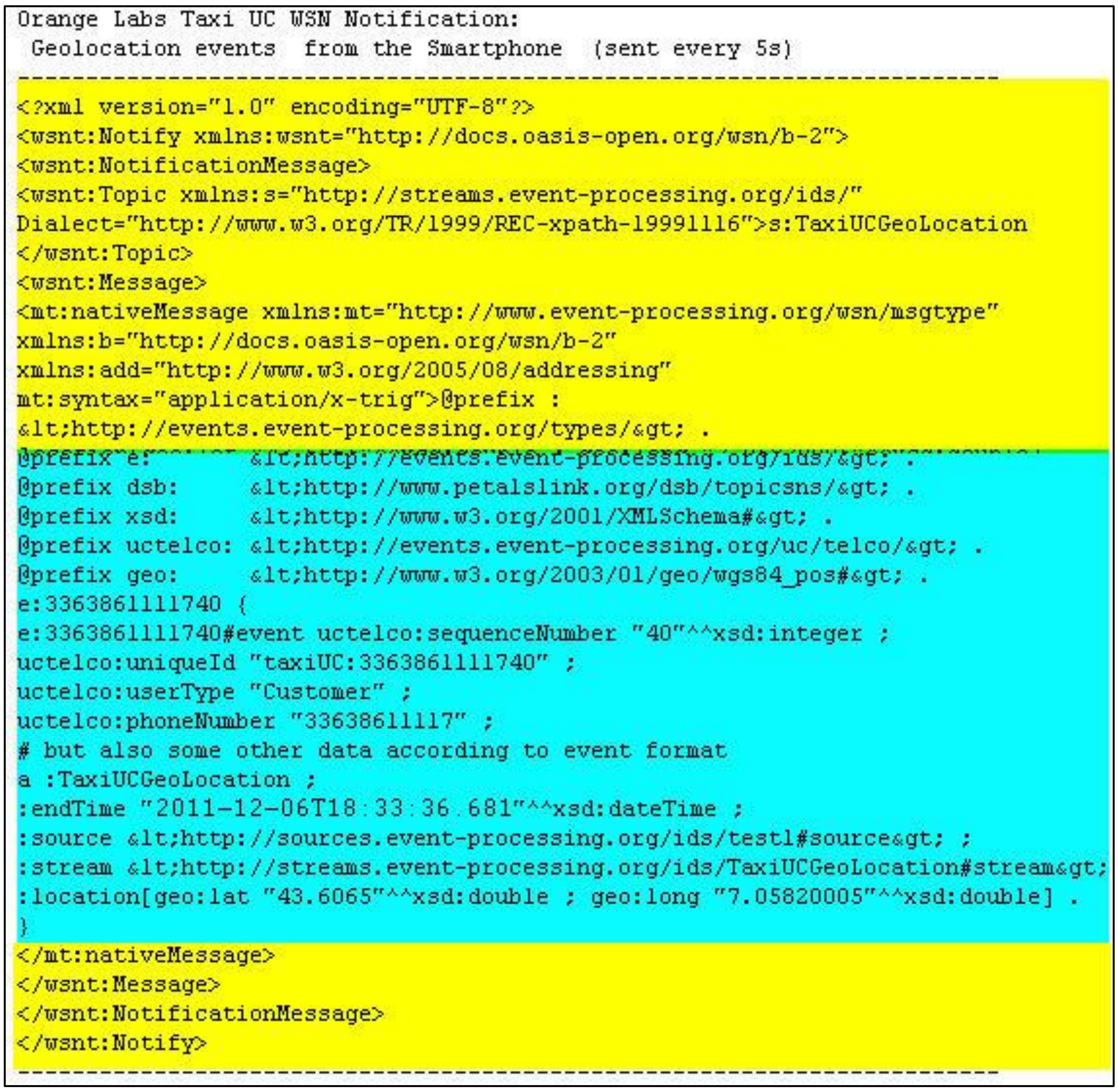
The different event instances pushed from PLAY platform to the MCM application are:

- *RecomEvent* : They indicate recommendation to use an application to the Caller (Twitter or Facebook, Google latitude)
- *Clic2callEvent*: They indicate that an automatic call between phones is going to be operated.

2.2.4 Design-time connection

Taking into account the requirements of the MCM Application, these event instances are pushed to the PLAY Platform. They are firstly, generated in a bean JAVA format and secondly transformed to *RDF* format thanks to textual templates provided by PLAY (see [3]) .

Thereafter is an example of business message for a *GeolocationEvent* instance containing the RDF (in cyan). The RDF message is enclosed with WSN notification tags + xml glue (in yellow).

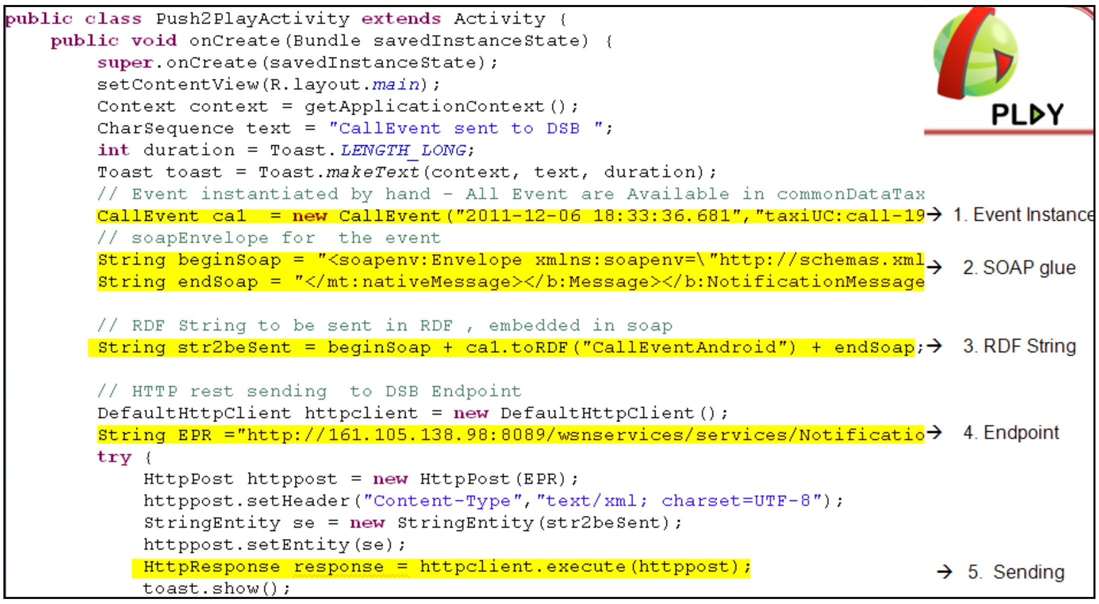


```
Orange Labs Taxi UC WSN Notification:
 Geolocation events  from the Smartphone  (sent every 5s)
-----------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
<wsnt:NotificationMessage>
<wsnt:Topic xmlns:s="http://streams.event-processing.org/ids/"
Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">s:TaxiUCGeoLocation
</wsnt:Topic>
<wsnt:Message>
<mt:nativeMessage xmlns:mt="http://www.event-processing.org/wsn/msgtype"
xmlns:b="http://docs.oasis-open.org/wsn/b-2"
xmlns:add="http://www.w3.org/2005/08/addressing"
mt:syntax="application/x-trig">@prefix :
&lt;http://events.event-processing.org/types/&gt; .
@prefix e:        &lt;http://events.event-processing.org/ids/&gt; .
@prefix dsb:      &lt;http://www.petalslink.org/dsb/topicsns/&gt; .
@prefix xsd:      &lt;http://www.w3.org/2001/XMLSchema#&gt; .
@prefix uctelco: &lt;http://events.event-processing.org/uc/telco/&gt; .
@prefix geo:      &lt;http://www.w3.org/2003/01/geo/wgs84_pos#&gt; .
e:3363861111740 {
e:3363861111740#event uctelco:sequenceNumber "40"^^xsd:integer ;
uctelco:uniqueId "taxiUC:3363861111740" ;
uctelco:userType "Customer" ;
uctelco:phoneNumber "33638611117" ;
# but also some other data according to event format
a :TaxiUCGeoLocation ;
:endTime "2011-12-06T18:33:36.681"^^xsd:dateTime ;
:source &lt;http://sources.event-processing.org/ids/test1#source&gt; ;
:stream &lt;http://streams.event-processing.org/ids/TaxiUCGeoLocation#stream&gt;
:location[geo:lat "43.6065"^^xsd:double ; geo:long "7.05820005"^^xsd:double] .
}
</mt:nativeMessage>
</wsnt:Message>
</wsnt:NotificationMessage>
</wsnt:Notify>
-----------------------------------------------------------------------
```

Figure 8: GeolocationEvent payload

Then the pushing of events to the PLAY platform is straightforward; the following snippet of Android Activity code gives the basic sequence for pushing a *CallEvent* instance.

1) Instantiate (with the help of a textual template) a *CallEvent* instance,
2) Set up the SOAP glue: this involves calls to WSN libraries (SOAP message compliant to WSN) ,
3) Transform the event in RDF and add SOAP glue before and after,
4) Set up the Endpoint,
5) And finally send the event thanks to HTTP in REST mode.



```
public class Push2PlayActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Context context = getApplicationContext();
        CharSequence text = "CallEvent sent to DSB ";
        int duration = Toast.LENGTH_LONG;
        Toast toast = Toast.makeText(context, text, duration);
        // Event instantiated by hand - All Event are Available in commonDataTax
        CallEvent ca1  = new CallEvent("2011-12-06 18:33:36.681","taxiUC:call-19→ 1. Event Instance
        // soapEnvelope for  the event
        String beginSoap = "<soapenv:Envelope xmlns:soapenv=\"http://schemas.xml→  2. SOAP glue
        String endSoap = "</mt:nativeMessage></b:Message></b:NotificationMessage

        // RDF String to be sent in RDF , embedded in soap
        String str2beSent = beginSoap + ca1.toRDF("CallEventAndroid") + endSoap;→  3. RDF String

        // HTTP rest sending  to DSB Endpoint
        DefaultHttpClient httpclient = new DefaultHttpClient();
        String EPR ="http://161.105.138.98:8089/wsnservices/services/Notificatio→  4. Endpoint
        try {
            HttpPost httppost = new HttpPost(EPR);
            httppost.setHeader("Content-Type","text/xml; charset=UTF-8");
            StringEntity se = new StringEntity(str2beSent);
            httppost.setEntity(se);
            HttpResponse response = httpclient.execute(httppost);                → 5.  Sending
            toast.show();
```

Figure 9: Missed Call Manager – Push2Play Android code example

This transformation occurs for every event instance that is pushed from Android phones to the PLAY DSB component in a very lightweight approach.

2.2.5 Adding New Event type

An important issue in the context of PLAY is to have the capability to add New *Event Type*. For example, the goal could be to add a new *EventType* signaling something happening on the sensors of the smartphone. We can describe a very simple *BatteryEvent type* that will deliver two interesting properties: the level of charge and the temperature of the battery. The goal is to push these event instances to the PLAY platform when for example the level is greater than 80% or less than 20%. The procedure will be

1. Describe the new *Event type*.

```
public class BatteryEvent extends inEvent {
protected String level;
protected String temperature;
}
```

2. Add it the *Event type* to the *EventIn* Hierarchy :

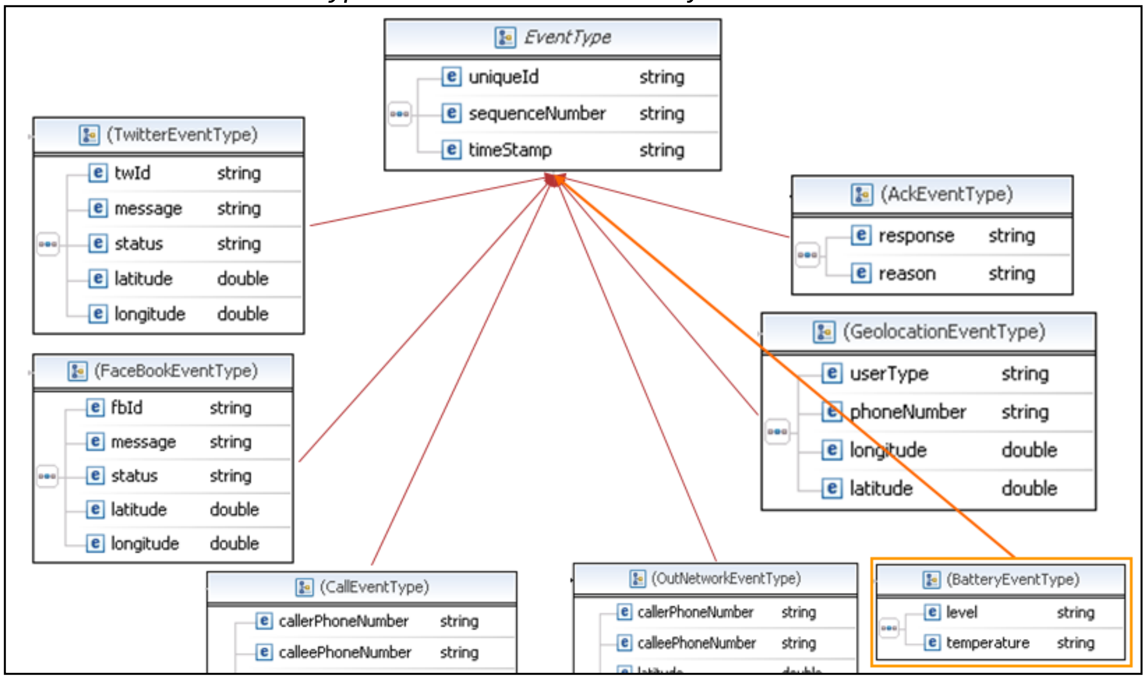
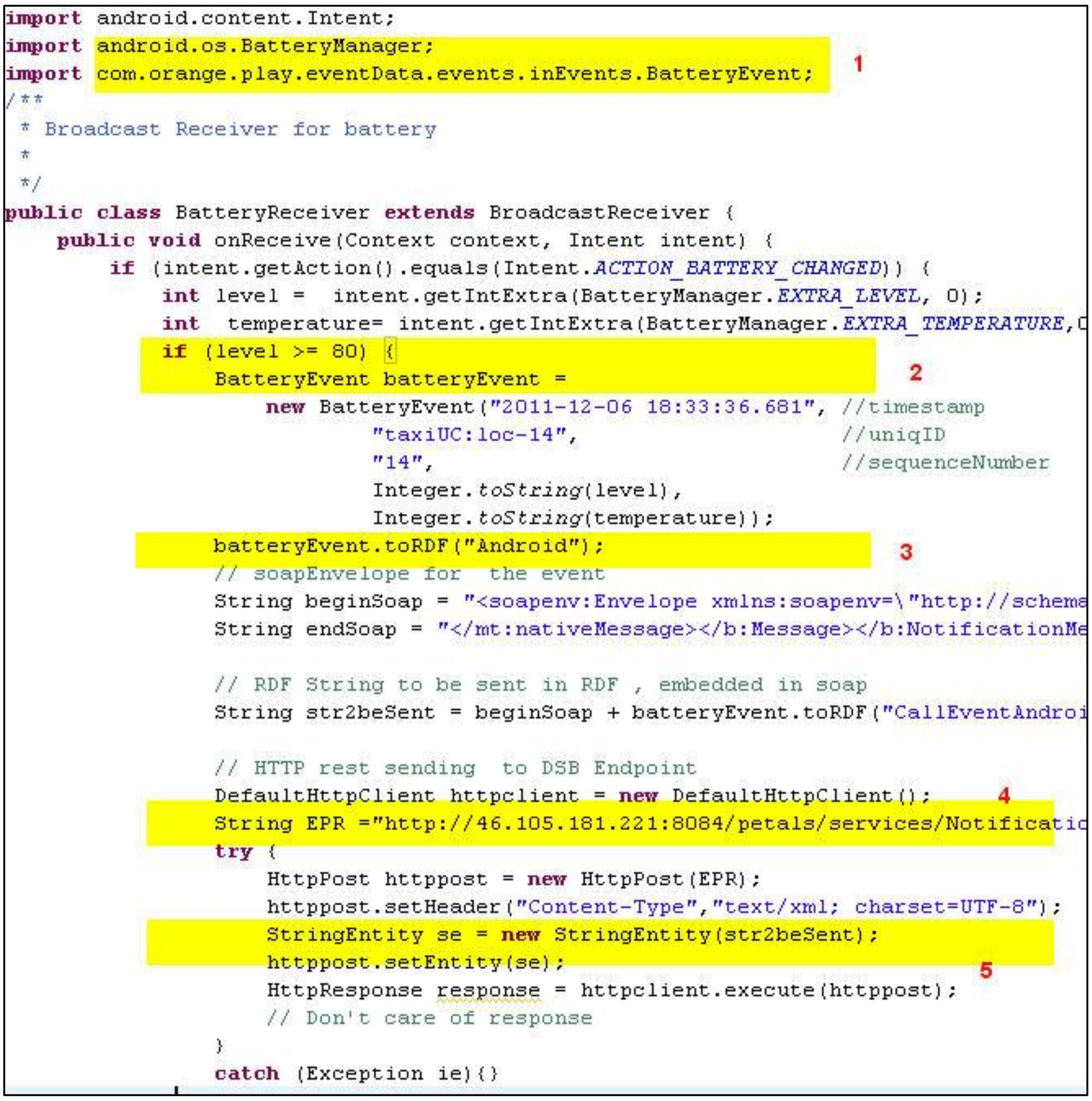


Figure 10: new Event type - BatteryEvent

3. Implement a *Receiver* on the Android smartphone extending the *BroadcastReceiver* class (see the Android documentation),

4. Listen to the Intent ACTION_BATTERY_CHANGED (see Android Intent documentation) to get battery information,

5. Register the Receiver for this Intent. The `onReceive` operation of the receiver class will be triggered automatically when there is a battery change and will send appropriately the event instance in RDF when the battery level is below or above the required value. The snippet of code for this `onReceive` operation is presented below:

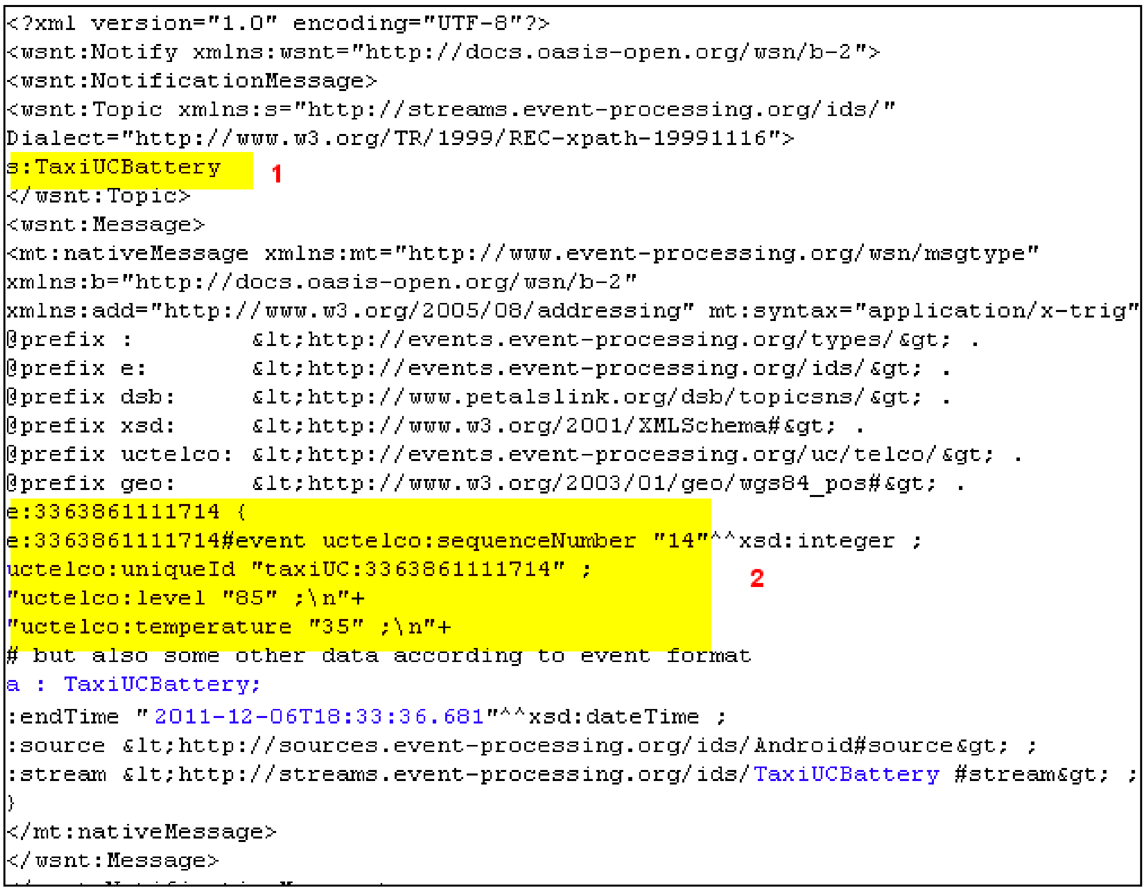


```
import android.content.Intent;
import android.os.BatteryManager;
import com.orange.play.eventData.events.inEvents.BatteryEvent;
/**
 * Broadcast Receiver for battery
 *
 */
public class BatteryReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(Intent.ACTION_BATTERY_CHANGED)) {
            int level =  intent.getIntExtra(BatteryManager.EXTRA_LEVEL, 0);
            int  temperature= intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE,
            if (level >= 80) {
                BatteryEvent batteryEvent =
                    new BatteryEvent("2011-12-06 18:33:36.681", //timestamp
                            "taxiUC:loc-14",                     //uniqID
                            "14",                                //sequenceNumber
                            Integer.toString(level),
                            Integer.toString(temperature));
                batteryEvent.toRDF("Android");
                // soapEnvelope for  the event
                String beginSoap = "<soapenv:Envelope xmlns:soapenv=\"http://scheme
                String endSoap = "</mt:nativeMessage></b:Message></b:NotificationMe

                // RDF String to be sent in RDF , embedded in soap
                String str2beSent = beginSoap + batteryEvent.toRDF("CallEventAndroi

                // HTTP rest sending  to DSB Endpoint
                DefaultHttpClient httpclient = new DefaultHttpClient();
                String EPR ="http://46.105.181.221:8084/petals/services/Notificatic
                try {
                    HttpPost httppost = new HttpPost(EPR);
                    httppost.setHeader("Content-Type","text/xml; charset=UTF-8");
                    StringEntity se = new StringEntity(str2beSent);
                    httppost.setEntity(se);
                    HttpResponse response = httpclient.execute(httppost);
                    // Don't care of response
                }
                catch (Exception ie){}
```

Figure 11: Android code example – pushing batteryEvent

The steps numbered in red are 1) import the mandatory Android package and the new Event type from inEvents, 2) get the battery level and if the condition is true, instantiate a *BatteryEvent* with the appropriate properties, 3) convert it to RDF and post it to the DSB 4) et 5).

6. The RDF output for this new *EventType* must be written with the help of a template. For this *BatteryEvent*, following the approach described in 2.2.4 we will get something like:



```
<?xml version="1.0" encoding="UTF-8"?>
<wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
<wsnt:NotificationMessage>
<wsnt:Topic xmlns:s="http://streams.event-processing.org/ids/"
Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
s:TaxiUCBattery    1
</wsnt:Topic>
<wsnt:Message>
<mt:nativeMessage xmlns:mt="http://www.event-processing.org/wsn/msgtype"
xmlns:b="http://docs.oasis-open.org/wsn/b-2"
xmlns:add="http://www.w3.org/2005/08/addressing" mt:syntax="application/x-trig"
@prefix :         &lt;http://events.event-processing.org/types/&gt; .
@prefix e:        &lt;http://events.event-processing.org/ids/&gt; .
@prefix dsb:      &lt;http://www.petalslink.org/dsb/topicsns/&gt; .
@prefix xsd:      &lt;http://www.w3.org/2001/XMLSchema#&gt; .
@prefix uctelco: &lt;http://events.event-processing.org/uc/telco/&gt; .
@prefix geo:      &lt;http://www.w3.org/2003/01/geo/wgs84_pos#&gt; .
e:33638611111714 {
e:33638611111714#event uctelco:sequenceNumber "14"^^xsd:integer ;
uctelco:uniqueId "taxiUC:33638611111714" ;                  2
"uctelco:level "85" ;\n"+
"uctelco:temperature "35" ;\n"+
# but also some other data according to event format
a : TaxiUCBattery;
:endTime "2011-12-06T18:33:36.681"^^xsd:dateTime ;
:source &lt;http://sources.event-processing.org/ids/Android#source&gt; ;
:stream &lt;http://streams.event-processing.org/ids/TaxiUCBattery #stream&gt; ;
}
</mt:nativeMessage>
</wsnt:Message>
```

Figure 12: Android code example – RDF for batteryEvent

The steps numbered in red are: 1) the new Event Type (TopicName). The PLAY platform must be changed to add this new TopicName to the configuration. 2) The different properties of the *batteryEvent* wrapped in the RDF.

Finally, these *batteryEvents* pushed to the PLAY platform can be exploited with a new CEPAT taking into account battery events.

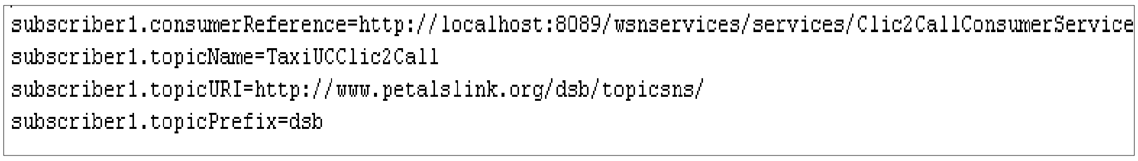
2.2.6 Run-time behavior

The Events come from the PLAY platform to the use-case when CEPATs have been triggered in the DCEP engine. These CEPATs have to be provisioned in the DCEP engine.

When CEPATs are matched, actions are triggered and events are sent back to the Orange back-ends. Two paths are described below (between the DSB and the WS Clic2call and between the DSB and the WS Orange Push2Android):

1. The description of the path from the DSB to the WS Clic2Call is:

An instance of the DSB, running in Orange Labs needs to be configured for receiving events. The following is an excerpt from the configuration file of the Orange Labs DSB Instance. When this DSB receives a *Clic2Call* Event, it notifies the `Clic2CallConsumerService` which parses the event and then automatically calls the PSTN network to operate the call between both entities (Caller and Callee)

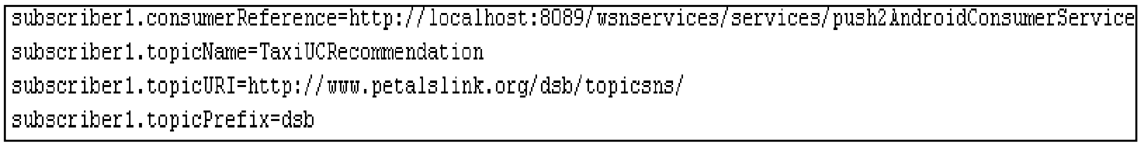


```
subscriber1.consumerReference=http://localhost:8089/wsnservices/services/Clic2CallConsumerService
subscriber1.topicName=TaxiUCClic2Call
subscriber1.topicURI=http://www.petalslink.org/dsb/topicsns/
subscriber1.topicPrefix=dsb
```

The endpoint `http://localhost:8089/wsnservices/services/Clic2CallConsumerService` receives the clic2Call instance, parses the RDF thanks to the adaptors provided by PLAY and finally makes the call.

2. The description of the path from the DSB to WSOrangePush2Android is:

An instance of the DSB, running locally in Orange needs to be configured for receiving events. The following is an excerpt from the configuration file of the Orange Labs DSB Instance. When this DSB receives a *recomEvent*, it notifies the `push2AndroidConsumerService` on the local DSB to parse the event and automatically pushes it to the WS `push2AndroidConsumerService.`



```
subscriber1.consumerReference=http://localhost:8089/wsnservices/services/push2AndroidConsumerService
subscriber1.topicName=TaxiUCRecommendation
subscriber1.topicURI=http://www.petalslink.org/dsb/topicsns/
subscriber1.topicPrefix=dsb
```

The endpoint

`http://localhost:8089/wsnservices/services/push2AndroidConsumerService` receives a *recomEvent* instance, parses the RDF thanks to the adaptors provided by the PLAY platform and finally pushes the notification to the right smartphone (as described in the section 3.2.3 , Android Push mechanism).

2.2.7 MCM – Overall Environment

Finally, the following gives an overall description: the main components (applications, Web Services and Events) and their path from and to applications and services.

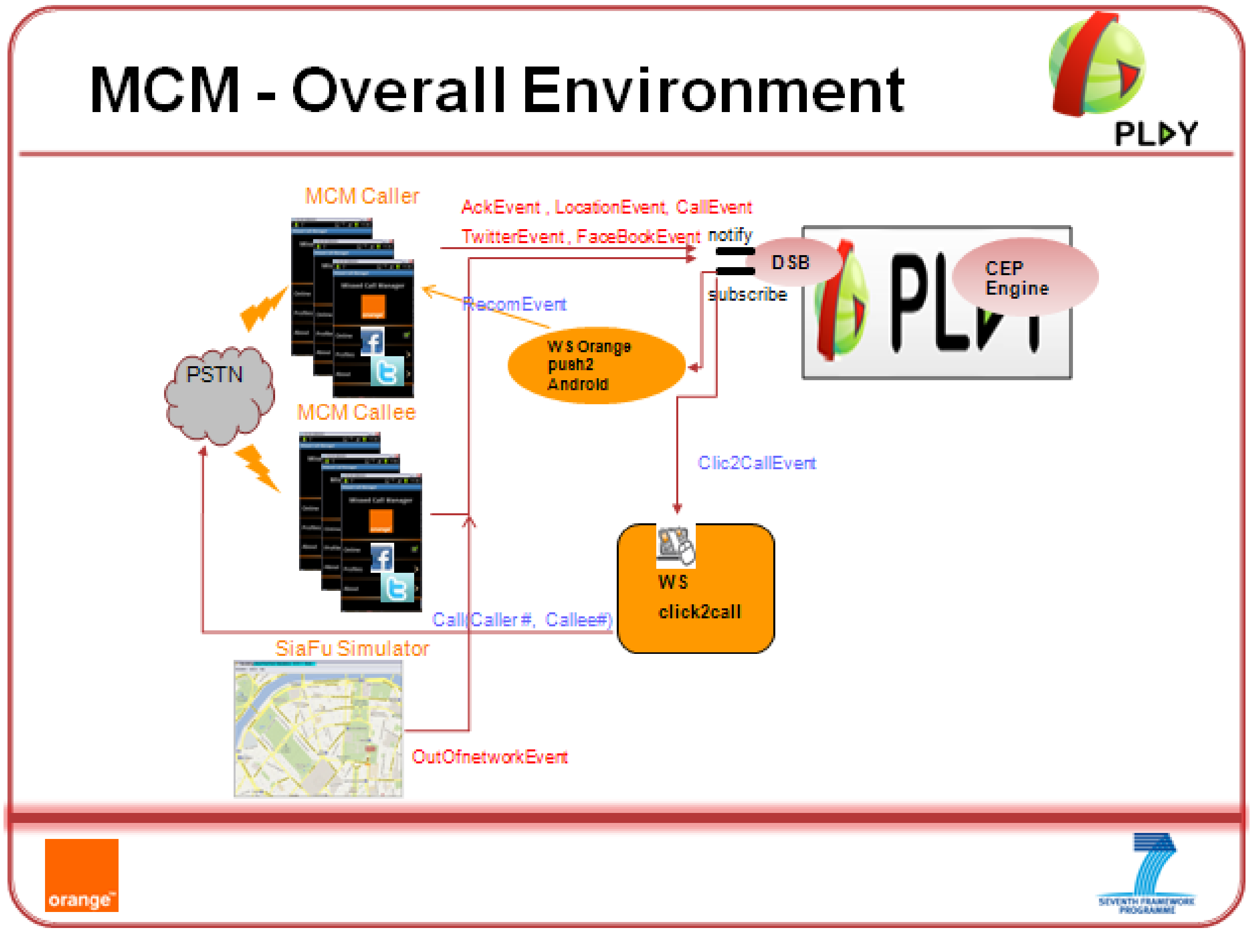


Figure 13: Missed Call Manager – Overall Environment

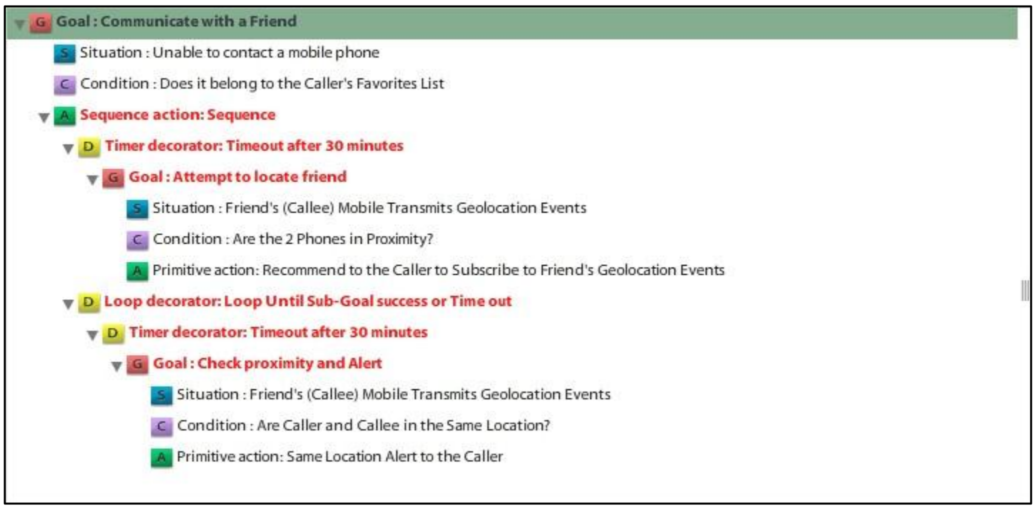
2.2.8 MCM – Outgoing calls detailed

Thereafter a detailed description for the processing for (missed) outgoing calls is presented:

- Every time the user makes an outgoing or incoming call, the MCM Application tracks it through opportune Android mechanism (Intents [2]) ,
- At the end of the call the Tracker checks if it was a missed call. In such a case, it sends a *CallEvent* to the PLAY DSB, containing all the data related to the user call (timeStamp, uniqueId, sequenceNumber, callDirection, message, callerPhoneNumber, calleePhoneNumber, latitude, longitude)
- According to the PLAY DCEP engine and the user-customized (current) profile configuration, when a specific situation of outgoing missed calls occurs in a specified time window, PLAY DCEP Engine triggers some CEPAT and generates a complex event, which activates some actions,
- Actions performed might be for example 'perform automatically a call when the Callee is available' or 'recommend using twitter to contact the Callee'.

2.2.9 ESR's Role in the Missed Calls Management Scenario

The Missed Calls Management scenario is considered a good case for providing dynamic event subscription capabilities through the ESR value added software component of PLAY. Event subscription recommender can be used for perceiving the situation in which a user is unable to contact a friend and finds it interesting to subscribe through a mobile device for the geolocation event stream of his/her friend. This information could be used by a dedicated mobile application that uses this event stream in order to constantly present the friend's position on a map once the 2 mobile phones appear to be in proximity. This scenario involves outgoing missed calls events and mobile phone geolocation events that are currently available in PLAY portal.



- G Goal : Communicate with a Friend
  - S Situation : Unable to contact a mobile phone
  - C Condition : Does it belong to the Caller's Favorites List
  - A Sequence action: Sequence
    - D Timer decorator: Timeout after 30 minutes
      - G Goal : Attempt to locate friend
        - S Situation : Friend's (Callee) Mobile Transmits Geolocation Events
        - C Condition : Are the 2 Phones in Proximity?
        - A Primitive action: Recommend to the Caller to Subscribe to Friend's Geolocation Events
    - D Loop decorator: Loop Until Sub-Goal success or Time out
      - D Timer decorator: Timeout after 30 minutes
        - G Goal : Check proximity and Alert
          - S Situation : Friend's (Callee) Mobile Transmits Geolocation Events
          - C Condition : Are Caller and Callee in the Same Location?
          - A Primitive action: Same Location Alert to the Caller

Figure 14: "Communicate with a Friend" SAN

We have deployed the SAN (Figure 14, Figure 15) "Communicate with a Friend" that involves the achievement of the two following subgoals:

- "Attempt to Locate Friend" – is considered achieved/successful when ESR issues a recommendation to the user to subscribe to geolocation events of his/her friend
- "Check Proximity and Alert" - is considered achieved/successful when ESR issues a proximity alert once the user and his/her friend are really close (i.e. less than 100m)

The traversal of the specific SAN results in the following actions from ESR perspective:

1. ESR waits for events that denote that the user (i.e. Caller) is unable to contact a specific mobile phone (Callee) 3 times in a 60 seconds time window (i.e. performs topic-based subscription).
2. If the above situation is detected then ESR checks whether or not this mobile phone belongs to a friend (i.e. compare against Caller's favorites list)
3. ESR waits for geolocation events transmitted by friend's mobile phone (i.e. content-based subscription). ESR will wait for such events for 30 minutes. After that period the Callee is considered unreachable.
4. ESR checks whether the two phones are in proximity (i.e. less than 500m)
5. ESR recommends to the Caller to subscribe to friend's geolocation events stream
6. After this recommendation ESR still waits for geolocation events transmitted by friend's mobile phone

7. If the above situation is detected then ESR checks whether the two phones are in the same location (i.e. less than 100m). The two last steps are repeated until the Caller and Callee appear in the same location.
8. ESR issues proximity Alert in order to inform the Caller that they both are in the same location.

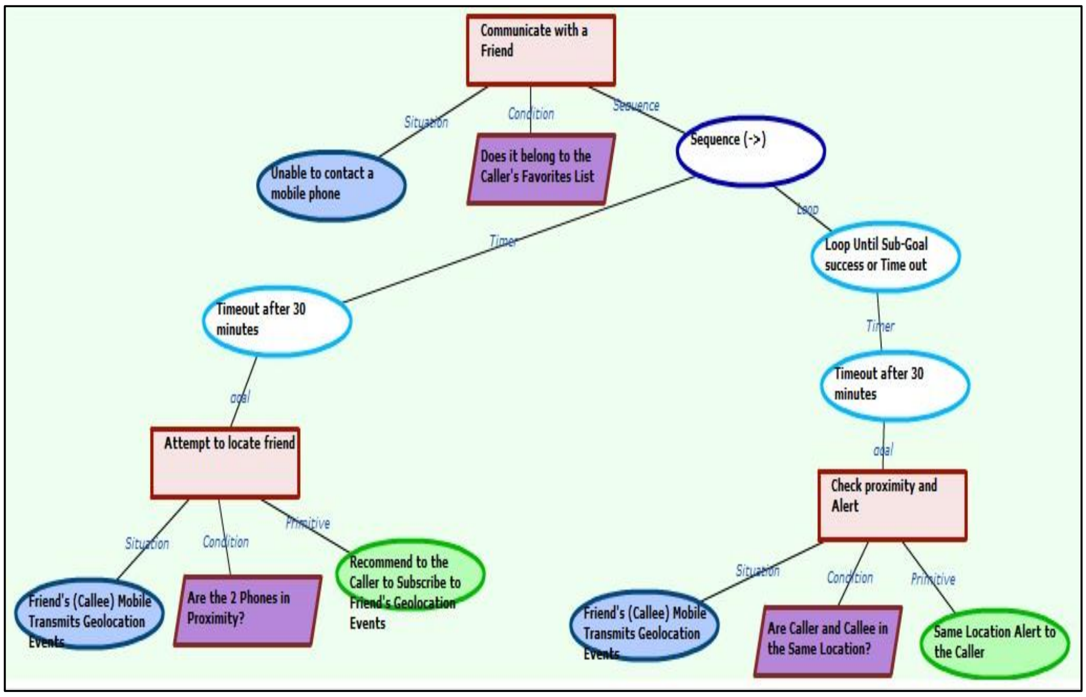
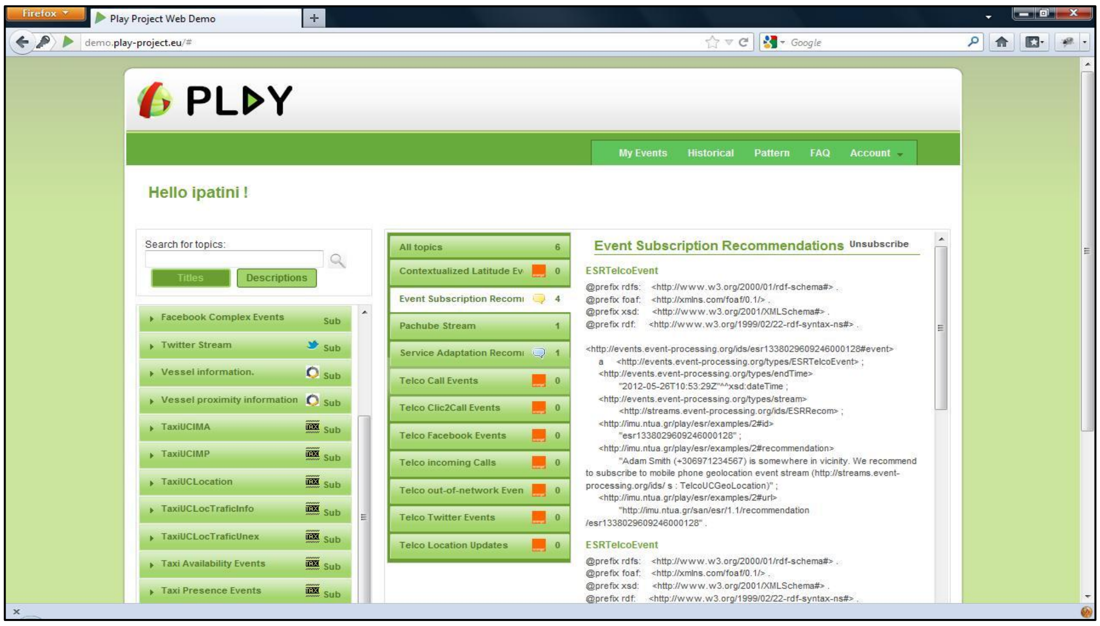

Communicate with a Friend
Situation
Condition
Sequence
Unable to contact a mobile phone
Does it belong to the Caller's Favorites List
Sequence (->)
Loop
Timer
Loop Until Sub-Goal success or Time out
Timeout after 30 minutes
Timer
Timeout after 30 minutes
goal
Attempt to locate friend
goal
Check proximity and Alert
Situation
Condition
Primitive
Friend's (Callee) Mobile Transmits Geolocation Events
Are the 2 Phones in Proximity?
Recommend to the Caller to Subscribe to Friend's Geolocation Events
Situation
Condition
Primitive
Friend's (Callee) Mobile Transmits Geolocation Events
Are Caller and Callee in the Same Location?
Same Location Alert to the Caller

Figure 15: "Communicate with a Friend" SAN – Graphical representation

The recommendations issued by ESR based on the discussed SAN currently appear as event notifications in PLAY portal under the event topic "ESRRecom". In the following Figure 16 and Figure 17 screenshots of ESR recommendations, relevant to the missed calls scenario, are depicted. Specifically, Figure 16 presents a recommendation for subscribing to a specific geolocation event stream while in Figure 17 the "same location" alert is depicted.



Hello ipatini !

Event Subscription Recommendations

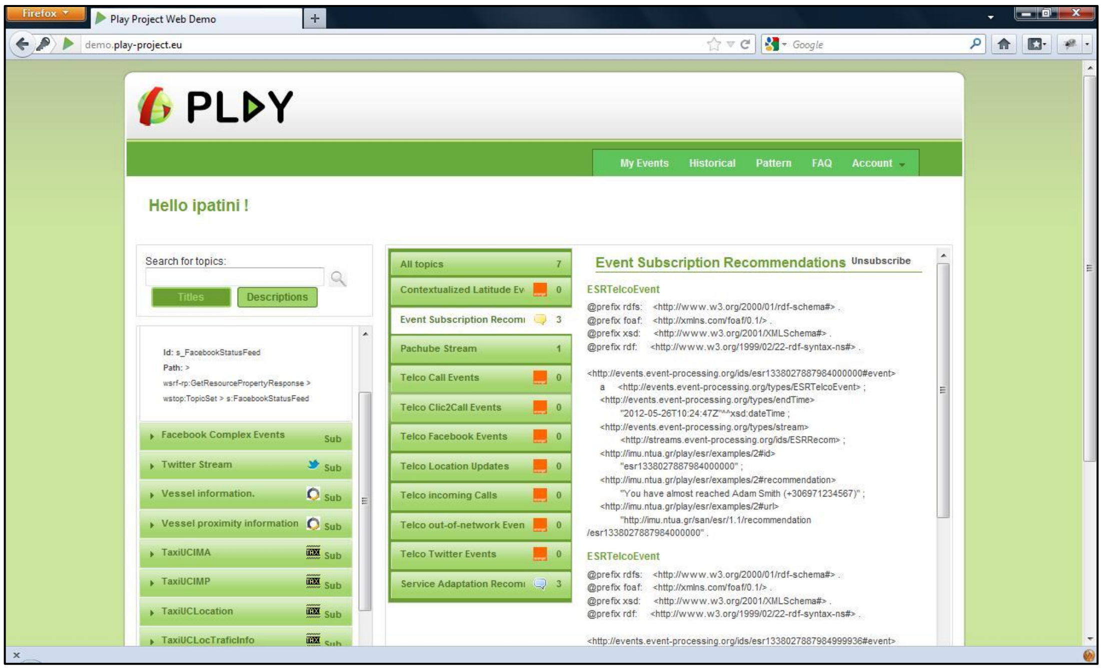
ESRTelcoEvent

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://events.event-processing.org/ids/esr1338029609246000128#event>
    a    <http://events.event-processing.org/types/ESRTelcoEvent> ;
    <http://events.event-processing.org/types/endTime>
          "2012-05-26T10:53:29Z"^^xsd:dateTime ;
    <http://events.event-processing.org/types/stream>
          <http://streams.event-processing.org/ids/ESRRecom> ;
    <http://imu.ntua.gr/play/esr/examples/2#id>
          "esr1338029609246000128" ;
    <http://imu.ntua.gr/play/esr/examples/2#recommendation>
          "Adam Smith (+306971234567) is somewhere in vicinity. We recommend
to subscribe to mobile phone geolocation event stream (http://streams.event-
processing.org/ids/ s : TelcoUCGeoLocation)" ;
    <http://imu.ntua.gr/play/esr/examples/2#url>
          "http://imu.ntua.gr/san/esr/1.1/recommendation
/esr1338029609246000128" .
```

ESRTelcoEvent

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```

Figure 16: Screenshot of the Event Subscription Recommendation



ESRTelcoEvent

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://events.event-processing.org/ids/esr1338027887984000000#event>
    a    <http://events.event-processing.org/types/ESRTelcoEvent> ;
    <http://events.event-processing.org/types/endTime>
          "2012-05-26T10:24:47Z"^^xsd:dateTime ;
    <http://events.event-processing.org/types/stream>
          <http://streams.event-processing.org/ids/ESRRecom> ;
    <http://imu.ntua.gr/play/esr/examples/2#id>
          "esr1338027887984000000" ;
    <http://imu.ntua.gr/play/esr/examples/2#recommendation>
          "You have almost reached Adam Smith (+306971234567)" ;
    <http://imu.ntua.gr/play/esr/examples/2#url>
          "http://imu.ntua.gr/san/esr/1.1/recommendation
/esr1338027887984000000" .
```

ESRTelcoEvent

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://events.event-processing.org/ids/esr1338027887984999936#event>
```

Figure 17: Screenshot of the “Same Location” Alert

3 Demo Environment

3.1 Global Description

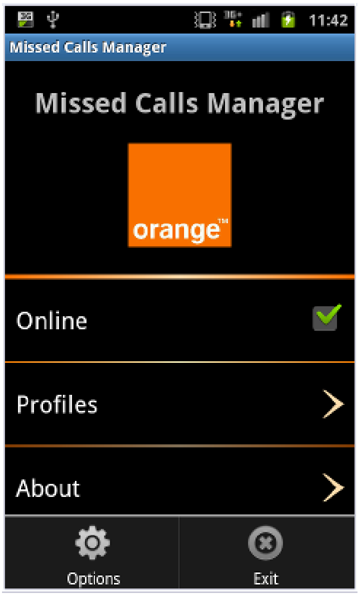
The demo environment consists of different applications and services deployed on smartphones devices and Application's servers. These applications and services are used during the demo.

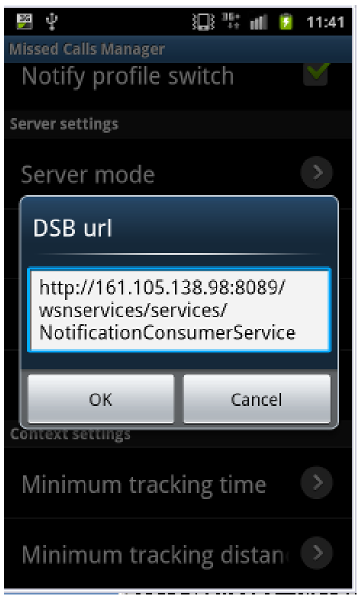
3.2 Applications and web services

3.2.1 Android smartphones Application

The MCM Application consists of:

- A background tracker Service that listens to incoming/outgoing calls,
- A GUI form that helps for customization and setup,





http://161.105.138.98:8089/wsnservices/services/NotificationConsumerService

Figure 18: MCM Service and MCM Set up DSB

By using this MCM Application the Caller and Callee are able to set up their personal context relatively to incoming and outgoing calls.

3.2.2 Siafu Simulator Application

In the MCM scenario #3, the Siafu Simulator is used to generate *OutNetwork* Events instances. The simulation must be closely in relation with the Smartphone Caller context to help testing correctly the scenario #3.

The goal is to gather events coming from ‘real’ sources (Smartphone’s) with events coming from virtual ones (Siafu simulator).

The Siafu simulation delivers *OutNetwork* Event instances when the Caller (User #10 below) is moving in the street. For example if the user enters the clear grey area, the simulation generates and pushes *OutNetworkEvent* instances with outofRange information meaning that the user’s Smartphone is out of network. The PLAY platform receives this event and reacts accordingly, thanks to CEPAT.

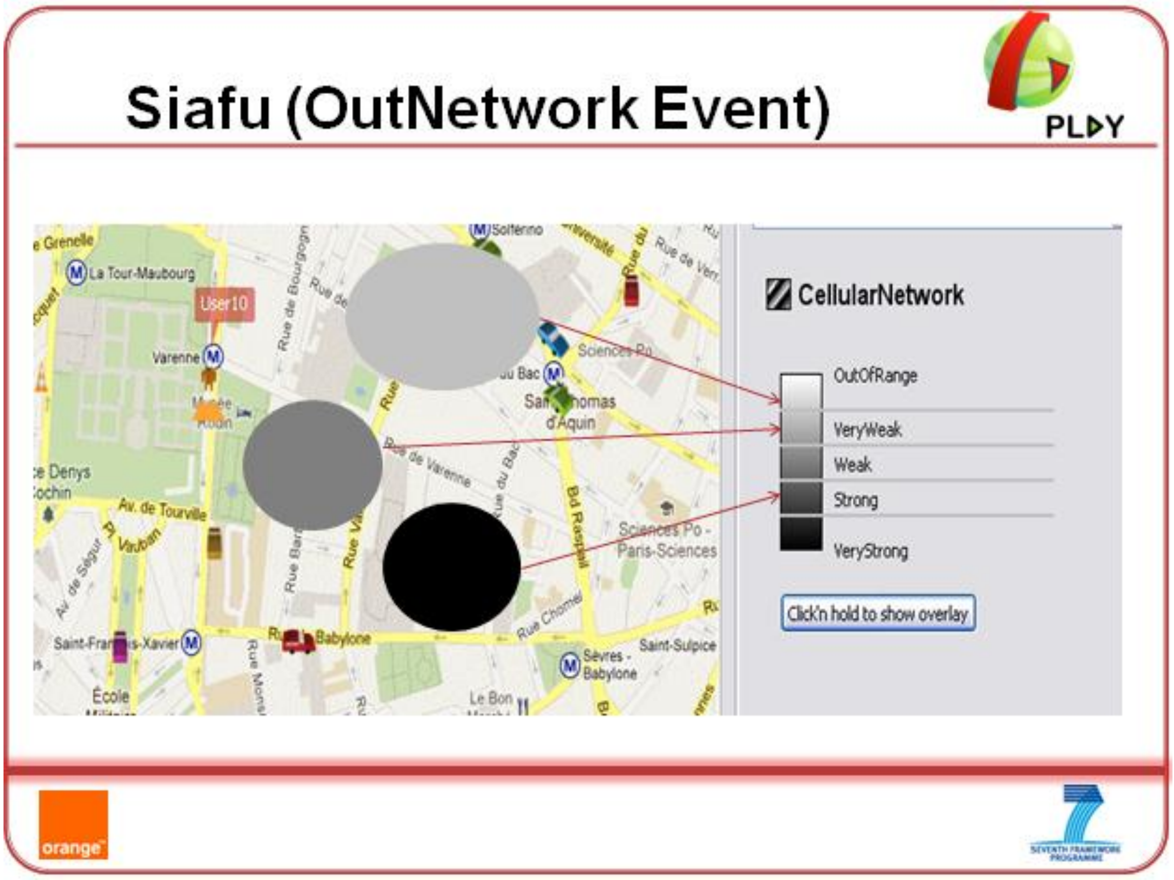


Figure 19: Siafu Simulator

3.2.3 Android Notification push mechanism

3.2.3.1 Description/Design

In order to push notifications from the PLAY DSB, an operational solution is needed. This push mechanism is based on Google Android C2DM see [8]. We started implementing the whole use case using SMS to push information to the customers. The problem with SMS could be the latency and the reliability of the service. This is the reason why we looked for something more reliable and with a better user experience.

The C2DM service stands for *Cloud To Device Messaging*. It answers our need offering a possibility to push information to the customer without having to keep a connection alive on the customer side.

The protocol is quite easy to use, it needs two steps before the applications can exchange with the mobile application. First the mobile application has to send a registration request to a Google server then it gets a deviceId. The mobile application can then provide this deviceId to the application supposed to send the notification (remote application, see below).

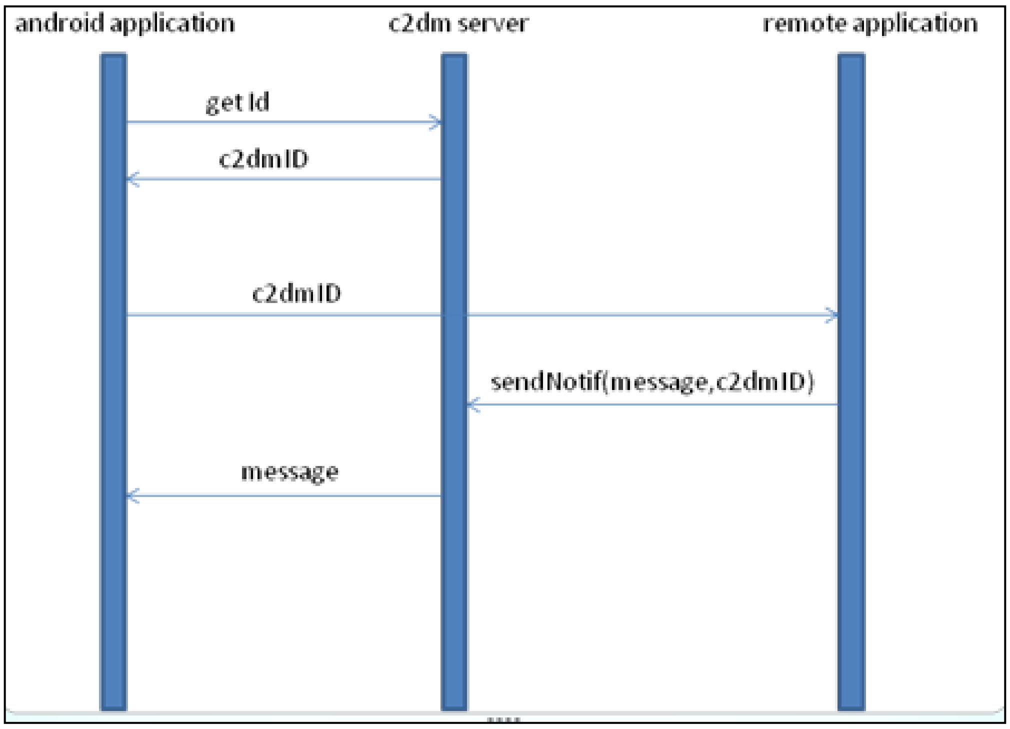


Figure 20 C2DM protocol - sequence diagram

3.2.3.2 Interface to DSB

When the PLAY DSB component wants to communicate with a user it uses the remote application that exposes a Web Service to push notification through the C2DM service but providing the phone number instead of the C2DM ID. In fact, the remote application stores the mapping between phone Number and C2DM ID during the registration. That way the DSB can push a notification to a mobile device using the phoneNumber.

3.2.3.3 Implementation

A general view describing how the push mechanism has been implemented is presented below.

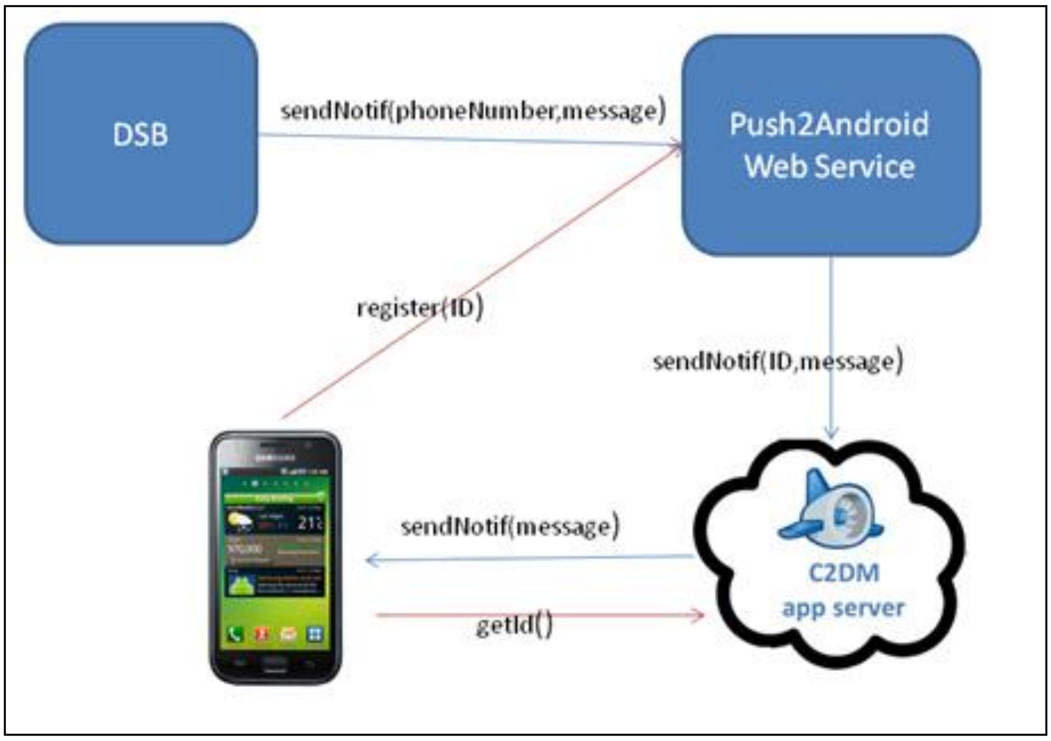


Figure 21 C2DM Interface to DSB

3.2.4 Click2Call Web Service

The click to call Web Service is an Orange Enabler available to Orange's customers through a SOAP or REST interface. In the case of PLAY usage we decided to make it simpler and linked it to one of our internal account in order to minimise the number of parameters. Basically we wrapped a Web Service on top of the official service. That way our partners can use the 'new service' providing the minimum number of parameters (in this case only two phone numbers).

This Web Service makes a call between two given phone Numbers. The official description is:

The application can make telephone calls to fix and mobile telephones worldwide

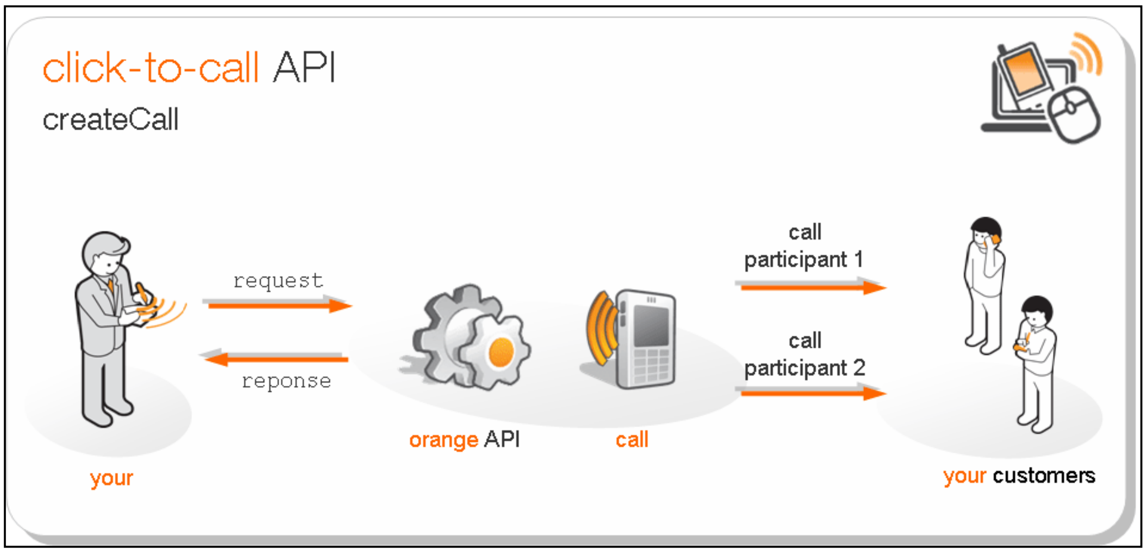


Figure 22 click-to-call API

In the case of PLAY we expose this Web Service through a REST API with only two parameters:

http://urlOfOurWebService?from=phone1&to=phone2

In this case phone1 and phone2 are the ones used to create the call. The two phones will ring and the communication will be opened when they both answer.

4 Connections with the PLAY platform

4.1 Constraints from the PLAY platform

4.1.1 Scalability

The telecom use-case through the MCM Application, deployed on a large scale, has the ability to push a lot of *GeoLocation* events, with frequent updates for the GPS position (every 10 sec for example). These flows of events pushed from smartphones must be absorbed quickly by the PLAY platform.

4.1.2 Security

High level of security regarding *GeoLocation Event* is required. The *Geolocation* data is very sensitive information as it directly concerns the privacy of customers. More generally, the properties such as PhoneNumber (for Caller and Callee) are also very sensitive.

4.2 Requirements for the PLAY platform

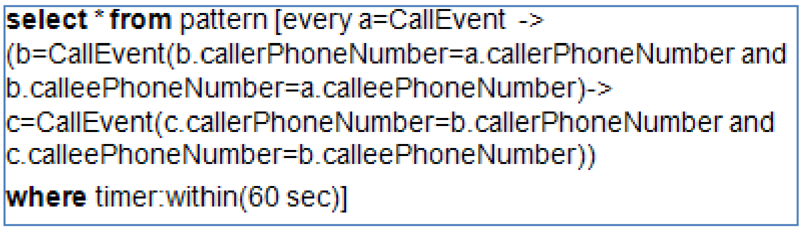
4.2.1 DSB Easy Interfacing

MCM application instances must be easily connected to the DSB and must be able to send event to the DSB in the required RDF format (as explained in 2.2.4).

The DSB must offer the publish/subscribe mechanism to MCM applications and orange Web Service back-ends (Push2Android and Clic2Call WS as explained in 2.2.63.2.3).

4.2.2 Definition of business rules: DCEP

The different scenarios described require the implementation of rules in the DCEP based on CEPATs. The first described situation concerns the detection of missed calls. The goal is to detect N missed calls from/to the MCM application in the time window (60s) and to react accordingly:



```
select * from pattern [every a=CallEvent  ->
(b=CallEvent(b.callerPhoneNumber=a.callerPhoneNumber and
b.calleePhoneNumber=a.calleePhoneNumber)->
c=CallEvent(c.callerPhoneNumber=b.callerPhoneNumber and
c.calleePhoneNumber=b.calleePhoneNumber))
where timer:within(60 sec)]
```

The reaction should vary from:

- The PLAY platform detects that this friend is online on Facebook (or have sent recently tweets on twitter from the Callee MCM application). PLAY informs him about this and automatically recommend Facebook or Twitter for the Caller,

- The PLAY platform reacts and recommends using Google Latitude to localize the friend, and traces the route between them,

- The PLAY platform reacts and recommends using Email to the Caller to contact the Callee,

- The PLAY platform detects that the Caller is located in an area where the cellular Network is out of reach (thanks to *outNetwork* Events sent by Siafu Simulator).

5 Tests

This section provides some basic requirements to answer the issues of testing and validation. The goal is to describe how to test and validate the use-case connected to the PLAY platform. The strategy is:

- To test outgoing events from the use-case to the PLAY platform,
- To test incoming events pushed from the PLAY platform to the use-case,
- Finally to validate the whole MCM application connected to the PLAY platform.

These testing steps need some platform-oriented and event-oriented requirements to be fulfilled (see [1] Section 11).

5.1 Testing environment

5.1.1 Smartphones pushing events

As the testing with many smartphones pushing events is really complex to setup (for GPS updates especially), **virtual** smartphones instances will be simulated with a *Siafu* simulation. Siafu features have been described in [4] 4.2.1.

Siafu will be customized to the MCM use-case testing requirements:

- *Agents* will be customers (Caller and Callee) and Location update events are sent automatically based on the original map of the simulation,
- *Places* are identified as Points Of Interest on the selected map,
- *Overlays* will describe where special conditions may occur (*OutNetworkEvent* or *CallEvent* in the context of MCM).

Key input parameters for the simulation will be:

- Number of simulated smartphones,
- GPS location update frequency,
- Duration of the simulation.

By varying these different parameters, the simulation will test various situations regarding the PLAY platform. For example, considering 50 smartphones with a frequency of 5 sec and duration of 12 hours, the simulation will be started from the command line like this: *startSiafu.sh –n 50 -f 5 – d 12*.

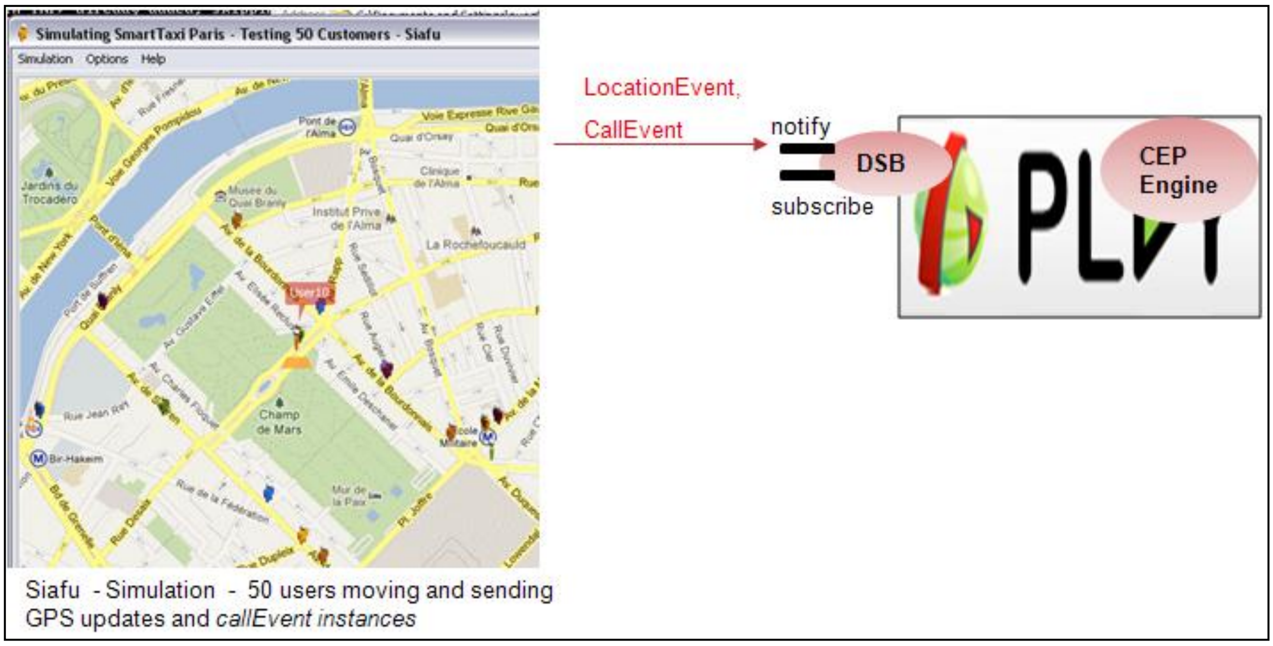

Simulating SmartTaxi Paris - Testing 50 Customers - Siafu
Simulation
Options
Help
LocationEvent,
CallEvent
notify
subscribe
DSB
CEP Engine
PLAY
Siafu - Simulation - 50 users moving and sending GPS updates and *callEvent instances*

Figure 23 Siafu simulation – pushing notifications

5.1.2 Smartphones receiving notifications

For the testing of notifications from the PLAY platform towards smartphones, we will emulate smartphones with the Android Emulator available with the Android SDK [9].

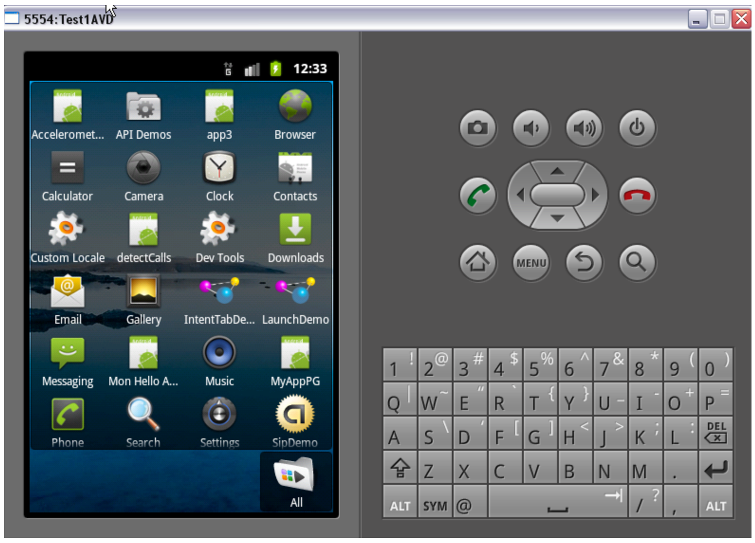


Figure 24 Android Emulator

Instances of the emulator will be started on demand from the command line: *emulator.exe -avd testAVD10* → (The targeted device is an Android 2.3.3 (API level 10)).

These different emulator instances will receive notifications from the *AndroidPush* mechanism described before in 3.2.3. Key parameters will be:

- Number of emulated smartphones and duration of the emulation.

For example, considering 4 emulated smartphones with emulation duration of 1 hour, the test will be started from the command- line like this:

startEmulators.sh –n 4 – d 1

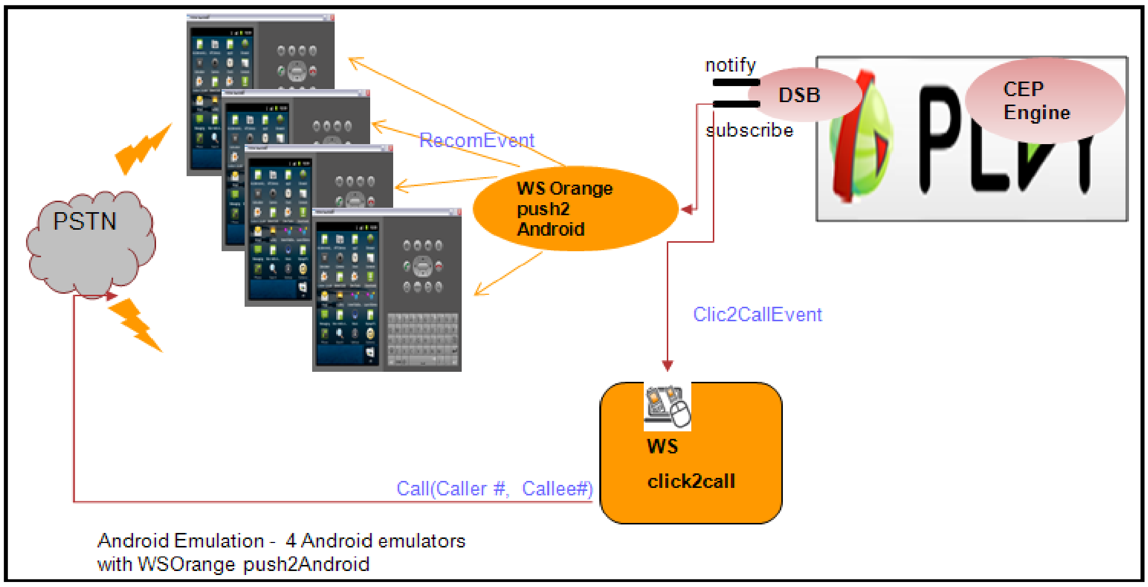


Figure 25 Android Emulation – Getting notifications

5.2 Basic Unit Testing

5.2.1 Testing outgoing events (MCM → PLAY)

The goal of this test is to stress the platform with events coming from many smartphones. The events considered are *GeolocationEvent* instances and *CallEvent* instances. This testing scenario refers to Figure 6: Missed Call Manager – Events from UC2PLAY and will be implemented using the approach described in 5.1.1.

Test-MCM-outgoing-1	
Description	Test the pushing of *GeolocationEvent* instances to the PLAY platform. The varying parameters for the simulation will be: • The number of virtual smartphones (Siafu) (5, 10, 50), • The frequency of GPS updates (ie *GeolocationEvent* instances pushed) per virtual smartphones (every 10s, 1 min, 1 hour), • The duration of the test (1 hour, 12 hours, 1 day), • The map perimeter will be Paris intra-muros.
Test Type	• Performance
Goal	• To stress the platform with *GeolocationEvent* instances, • To count the number of sent events and compare it with the number of events received by the platform, • To get the latency between event generation at the source and event detection at the platform level
Prerequisites	• Siafu simulation ready, • DSB operational *,* • PLAY monitoring logs available.
Output	• The test trace is stored locally. Occurrence time of events is to be compared with detection time provided by the logs of the PLAY platform

Table 1: Test-MCM-outgoing-1

The goal of this test is to mix the sending of *GeolocationEvent* and *CallEvent instances* and stress the platform*.*

Test-MCM-outgoing-2	
Description	Test the pushing of *GeolocationEvent and CallEvent* instances together to the PLAY platform. The varying parameters for the simulation will be: • The number of virtual smartphones (5, 10, 50), • The frequency of GPS updates (ie *GeolocationEvent* instances pushed) per virtual smartphones (every 10s, 1min, 1 hour), • The frequency of Call*Event* instances pushed per virtual smartphones (every 10s, 1mn, 1 hour) , • The duration of the test (1 hour, 12 hours, 1 day), • The map perimeter will be Paris intra-muros.
Test Type	• Performance
Goal	• To stress the platform with *GeolocationEvent* instances and *CallEvent* instances, • To count the number of sent events and compare it with the number of events received by the platform, • To evaluate the latency between event generation at the source and event detection at the platform level
Prerequisites	• Siafu simulation ready, • DSB operational, • PLAY monitoring logs available.
Output	• The test trace is stored locally. Occurrence time of events is to be compared with detection time provided by the logs of the PLAY platform

Table 2: Test-MCM-outgoing-2

5.2.2 Testing incoming events (PLAY → MCM)

The goal here is to test the events originating from the PLAY platform targeting the right smartphone. The events considered are *RecomEvent* instances. As the pushing of notifications with many smartphones involved is really complex to setup, Android emulator of smartphones instances will be used. With the emulator we can start up to 10 android virtual smartphones with automatic shell scripts. This testing scenario refers to Figure 7: Missed Call Manager – Events from PLAY2UC and will be implemented using the approach described in 5.1.2.

So the test for incoming *RecomEvent* instances will be:

Test-MCM-incoming-1	
Description	Test the pushing of *RecomEvent* instances to the *Orange WS Push2Android*. The Orange WS realizes the interface between the PLAY DSB and android emulator instances The varying parameters for the simulation will be: • The number of android emulators (2, 5, 10), • The duration.
Test Type	• Scalability
Goal	• Check that *RecomEvent* instances are correctly delivered to the right Android virtual smartphone. • Check the latency between event generation by the platform and event reception by the Android virtual smartphone.
Prerequisites	• MCM application installed on Android emulator, • Orange WS operational, • *WS Orange Push2Android* operational,
Output	• Emulator instances logs the received events

Table 3: Test-MCM-incoming-1

The goal here in this test is to test the events from the PLAY platform targeting the clic2call Scenario. The events considered are *Clic2callEvent instances*.

Test-MCM-incoming-2	
Description	Test the pushing of *Click2CallEvent* instances to the *Orange WS Clic2call* (Caller# and Callee#)
Test Type	• Functional
Goal	• Check that Click2CallEvent instances arrived correctly and are processed (with the PTSN network)
Prerequisites	• MCM application installed on smartphone • *WS Orange Click2call* operational
Output	• Communication is automatically established between the Caller and the Callee

Table 4: Test-MCM-incoming-2

5.3 Validation

The goal is to test each sub-scenario from a functional point of view and then to validate the whole application.

5.3.1 Functional Test – Sub-Scenario 1a)

Test-MCM-Sub-Scenario 1a	
Description	1. Find and start the best service to put Customer and friend in relation: The customer (Caller) calls a friend. The friend (Callee) does not answer for whatever reason. The PLAY platform is aware of this (after a sequence of calls) and detects that this friend is online on Facebook (or have sent recently tweets on twitter). PLAY informs him about this and automatically opens Facebook or twitter for the Caller
Test Type	• Functional
Goal	• Testing the situation for Twitter or Facebook recommendation
Prerequisites	• MCM application installed on the Caller Smartphone • MCM application installed on the Callee Smartphone • *WS Orange Push2Android* operational, • CEPATs (see 4.2.2) operational.
Output	• Caller receives *recomEvent* to use either Twitter either Facebook on the smartphone

Table 5: Test-MCM Sub-Scenario 1a

5.3.2 Functional Test – Sub-Scenario 1b)

Test-MCM- Sub-Scenario 1b	
Description	1. Find and start the best service to put Customer and friend in relation: The customer calls a friend but the friend is located in a crowd of people (in a stadium) and so does not answer. The friend has not started Facebook or Twitter, but the Customer and Friend are using Google Latitude. After N missed calls, PLAY platform reacts and proposes to use Google Latitude to localize the friend, and traces the route between them
Test Type	• Functional
Goal	• Testing the situation for Google Latitude recommendation
Prerequisites	• MCM application installed on the Caller Smartphone • MCM application installed on the Callee Smartphone • *WS Orange Push2Android* operational, • CEPATs (see 4.2.2) operational.
Output	• Caller receives *recomEvent* to use Google Latitude on the smartphone

Table 6: Test-MCM Sub-Scenario 1b

5.3.3 Functional Test – Sub-Scenario 2

Test-MCM- Sub-Scenario 2	
Description	2. Put Customer and friend in relation whatever the service: The customer calls the friend N times with no answer. The friend has started his smartphone, but neither Facebook, nor twitter, nor latitude application. The PLAY platform tries to send 2 times an SMS – no answer - then tries to send twitter messages – no answer - then the same for Facebook. Finally from the Customer contacts list, the email address for the friend is extracted and an email is sent.
Test Type	• Functional
Goal	• Testing the situation "Email sent to the Caller"
Prerequisites	• MCM application installed on the Caller Smartphone • MCM application installed on the Callee Smartphone • *WS Orange Push2Android* operational, • *WS Orange Click2call* operational (for sending SMS) • CEPATs (see 0) operational.
Output	• Caller receives an Email

Table 7: Test-MCM Sub-Scenario 2

5.3.4 Functional Test – Sub-Scenario 3

Test-MCM- Sub-Scenario 3	
Description	3. Put Customer and friend in relation whatever the cellular Network status: The customer tries to call his friend but the network refused the call because of the overload network. The network informs the PLAY platform which sends an SMS to the customer with a proposal: For 0.30 €, the customer can make his outgoing call as soon as a time slot is available. These exchanges are done by SMS in order to cope with cell load. If he accepts, a server automatically put them in relation
Test Type	• Functional
Goal	• Testing the situation "Email sent to the Caller"
Prerequisites	• MCM application installed on the Caller Smartphone • MCM application installed on the Callee Smartphone • *WS Orange Push2Android* operational, • SiafuSimulator ready (for sending *OutNetworkEvent*) • *WS Orange Click2call* operational (for sending SMS) • CEPATs (see 0) operational.
Output	• Communication is automatically established between the Caller and the Callee after some SMS exchanges between Caller and PLAY platform.

Table 8: Test-MCM Sub-Scenario 3

5.3.5 Validation for the whole MCM application

The goal is to validate the whole MCM application. The MCM application is installed on both Caller and Callee smarpthones.

Validation-MCM	
Description	The MCM application is installed for the Caller and for the Callee
Test Type	• Validation
Goal	• To Validate the whole Application
Prerequisites	• MCM application installed on the Caller Smartphone • MCM application installed on the Callee Smartphone • *WS Orange Push2Android* operational, • *WS Orange Click2call* operational • Siafu simulation ready, • DSB operational, • PLAY monitoring logs available. • CEPATs (see 0) operational.
Output	• MCM Caller and Callee applications are operational on smartphones. Different situations described are experimented by the Caller and Callee.

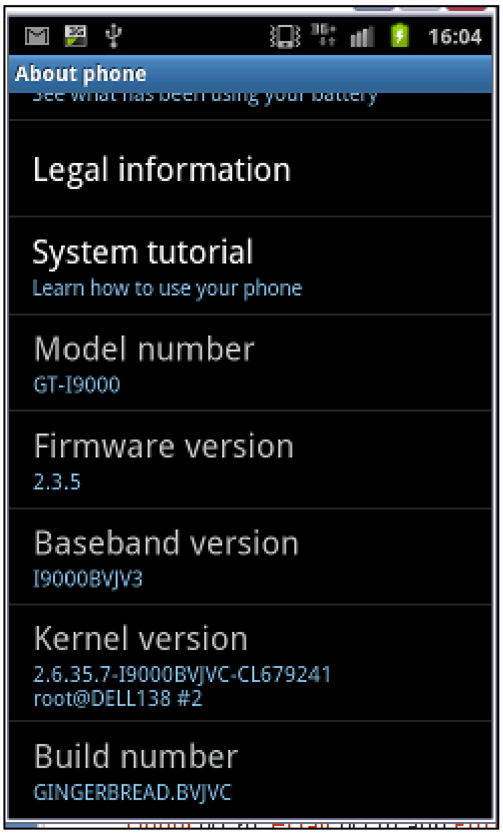
Table 9: Validation-MCM

6 Installation

6.1.1 Installation procedure

This procedure describes how to install manually the MCM.apk application on a Samsung S1 Android Smartphone.

Prerequisite: Firmware version should be > 2.3.5 (Gingerbread codename). Go to general Menu *Settings -> AboutPhone* and please check it.



16:04

About phone

See what has been using your battery

Legal information

System tutorial
Learn how to use your phone

Model number
GT-I9000

Firmware version
2.3.5

Baseband version
I9000BVJV3

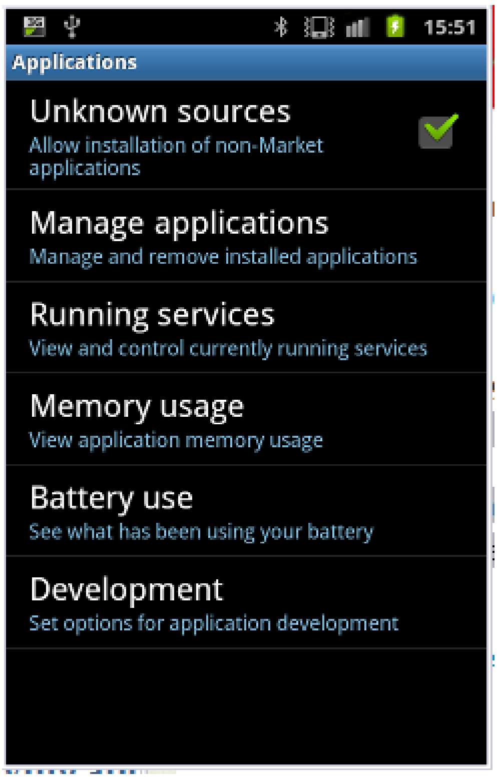
Kernel version
2.6.35.7-I9000BVJVC-CL679241
root@DELL138 #2

Build number
GINGERBREAD.BVJVC

Figure 26 Android Gingerbread codename (2.3.5)

Then

1. Allow your phone to install android Applications from “*Unknown Sources*” (i.e. non-Market apps). To do this, navigate to *Menu -> Settings -> Applications* and check the box marked “*Unknown Sources*“.



15:51

Applications

Unknown sources
Allow installation of non-Market applications ✓

Manage applications
Manage and remove installed applications

Running services
View and control currently running services

Memory usage
View application memory usage

Battery use
See what has been using your battery

Development
Set options for application development

Figure 27 Android Menu Settings

2. Then either
   a. Copy the *mcmVx.y.apk* file you want to install to your phone's memory card and insert the card into your Android phone,
   b. Go to Android Market and search for the *Fast Installer* application,
   c. Open it and click on the Install button,
   d. After it is installed, just open it. It will show you all the *apk* files stored directly in the root directory of your memory card,
   e. Just click on the *mcmVx.y.apk* application to install it.

   or

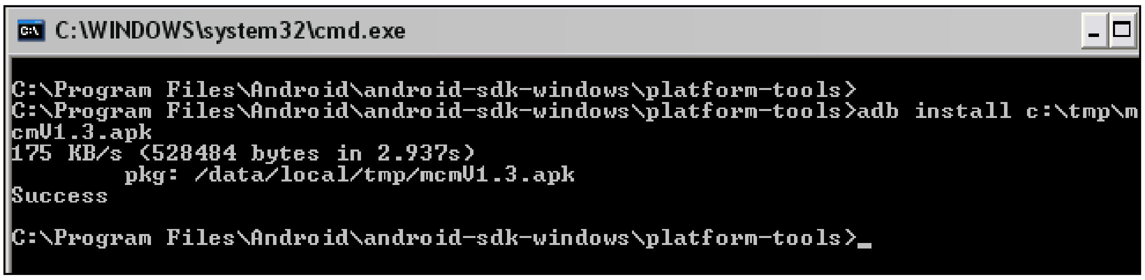
   a) Install the Android SDK on your PC (You should get something like *C:\Program Files\Android\android-sdk-windows\* for the default installation folder).
   b) Connect your phone to the computer with your USB cable
   c) Next, just open Command Prompt on the PC and type:

   adb install path/ mcmVx.y.apk

   → *adb.exe* is located in C:\Program Files\Android\android-sdk-windows\platform-tools (by default).

→ path it the full path on your PC to reach your *mcmVx.y.apk*.

d) Your *mcmVx.y.apk* application is now installed. (See below the trace for adb.exe installation on a PC, taking the *mcmV1.3.apk* file from the *c:\tmp* folder



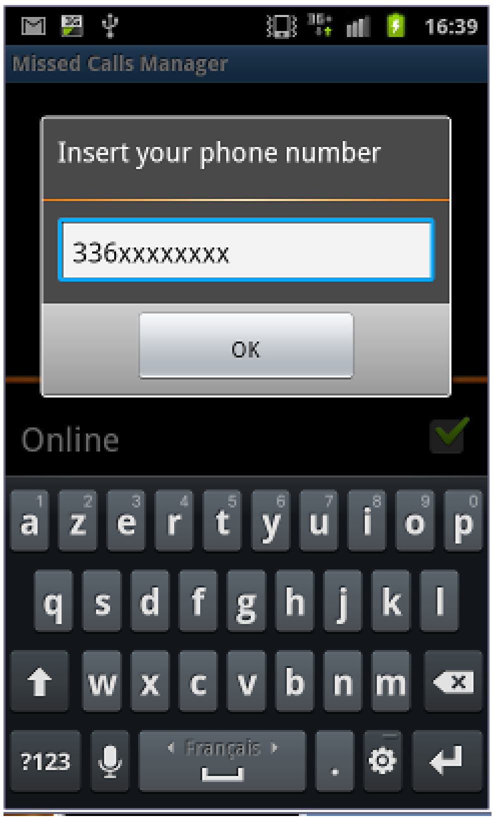
```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\Android\android-sdk-windows\platform-tools>
C:\Program Files\Android\android-sdk-windows\platform-tools>adb install c:\tmp\m
cmV1.3.apk
175 KB/s (528484 bytes in 2.937s)
        pkg: /data/local/tmp/mcmV1.3.apk
Success

C:\Program Files\Android\android-sdk-windows\platform-tools>_
```

Figure 28 Installing apk

3. Then

a) Check that you have Data Internet Access and GPS activated on your smartphone,

b) Now open the application on your phone and use it. You should get something like the following MCM application asking for your local *phone number* with international prefix (33 for France, 6 for mobile in the example). This step is mandatory. Then the MCM Application will be ready (fig Figure 30 Application ready)



Missed Calls Manager

Insert your phone number

336xxxxxxxx

Figure 29 Inserting Phone Number

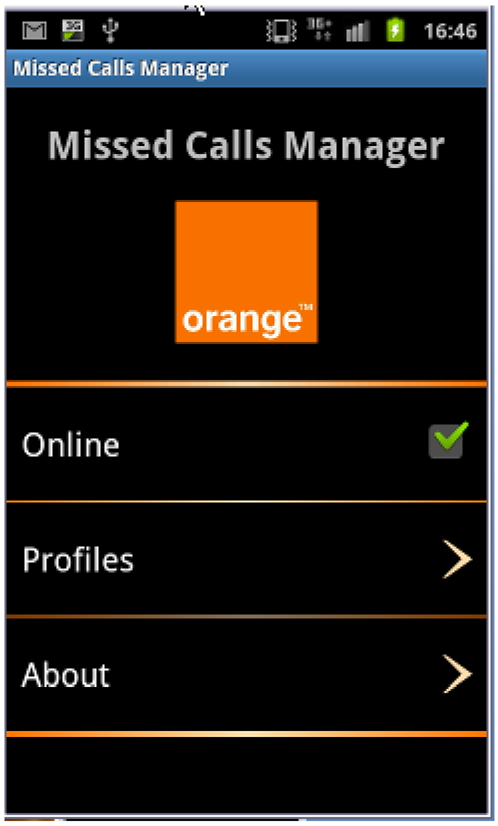


Figure 30 Application ready

c) The Online status means that the background Service is running, ready to intercept calls (incoming and outgoing). It can be unchecked by the user.

7 Conclusion

The MCM scenario has been described and a technical view of the demo environment presented.Then the tests and validation to be passed have been listed. A quick installation manual has been described.

The execution of these tests and validation plans will lead to an operational MCM application connected to the PLAY platform.

References

[1] D1.3 PLAY Requirements Analysis: http://www.play-project.eu/index.php/documents/doc_download/106-play-d13-requirements-analysis-final.html

[2] Android Intents http://developer.android.com/reference/android/content/Intent.html

[3] PLAY Event Adaptors http://research.petalslink.org/display/play/Event+Adapter

[4] D6.1.1 Scenario of the Telecom Use-Case http://www.play-project.eu/documents/viewdownload/3-deliverables-final/24-play-d611-scenario-of-the-telecom-use-case

[5] Siafu http://siafusimulator.sourceforge.net/

[6] Telco Event Types http://research.petalslink.org/display/play/Event+Types

[7] Eclipse commonDataTaxi https://svn.petalslink.org/svnroot/trunk/research/projects/play/play-usecases/play-usecase-telecom/commonDataTaxi/

[8] Android push C2DM https://developers.google.com/android/c2dm/

[9] Android Emulator http://developer.android.com/guide/developing/tools/emulator.html