Integrating mobile services and content with the Internet

Ángel Machín
Vodafone R&D
Parque Tecnológico Walqa
22197, Huesca - Spain
+34610517426
angel.machin@vodafone.com

Curro Domínguez
Vodafone R&D
Parque Tecnológico Walqa
22197, Huesca - Spain
+34610513339
francisco-javier.dominguez1@vodafone.com

ABSTRACT

This paper describes a feasible way of integrating mobile networks with the Internet, in order to create an ecosystem that allows mobile devices to receive common web transactions and allows Internet peers to be notified of mobile generated events.

Traditionally, PULL networks have been a concept associated with the internet network style (I want something, I search, I get) because it requires a simpler architecture. On the other hand, pure PUSH networks have been identified as Telco's networks, which are more complex but able to offer users a more customised service. This is caused by the fact that mobile users, when first attached to the mobile network, are automatically subscribed to network events, so they are able to receive notifications from the network like phone calls, SMS, etc.

However, push networks have also been largely developed on the internet, as event-driven notification systems and publish-subscribe mechanisms. SIP or XMPP based networks have been widely adopted and hybrid mechanisms like RSS or SMTP (where notifications are pushed but users have to pull new content) have proven to be extremely successful.

So far, the integration between mobile and internet networks doesn't take advantage of mutual benefits of both kinds of networks. Therefore, the adoption of the solution presented in this paper provides a bridge between mobile networks and the Internet based on data-pushing and event delivering.

Categories and Subject Descriptors

C.2.1 [**Computer communication networks**]: Network architecture and design - distributed networks, network topology, wireless communications, network communications.

D.0 [**Software General**]

General Terms

Performance, Design, Reliability, Security, Standardization.

Keywords

Push networks, event broker, mobile generated event, REST.

1. INTRODUCTION

For the past 3 years, a growing number of solutions aimed at providing e-mail access through push technologies have appeared.

Mail pushing is achieved by having a persistent connection between the device and the mail server, which pushes notifications when a mail box is updated.

This technology has proven to be efficient and reliable and doesn't impose a big impact on mobile device performance and battery life. Current platforms for content pushing are developed by third parties that deploy their infrastructure inside the operator network. None of these current platforms are generic and open, working only with specific applications, typically mail servers, by using proprietary protocols.

In other scenarios different than e-mail, poor polling-based approaches are very common because of the lack of a solution to push messages from external servers. But there are many other areas where the push technology could be particularly interesting. One example is distributed IT applications that typically have very limited interaction between devices and servers on the Internet.

Due to the inherent nature of mobile networks, this project has adopted the PUSH concept as the basis of the solution to link the mobile and internet networks. However, this attempt also put together the specific problems and features of both kind of networks. For instance, mobile devices are not easily addressable because of their limited connection and the lack of a fixed Internet address to access them.

For these reasons, the analysis and understanding of the constraints that mobile devices drag on and the comprehension of mobile and internet architectures are a key part of the design and implementation of the solution.

There exist significant differences in the way internet and mobile architectures have been designed. The main problems the project tries to face are:

1. How to solve the connectivity problem of mobiles devices.
2. How to translate mobile data and services to standard internet language.
3. How to push mobile notifications to internet networks.

1.1 Connectivity problem

Mobile devices present three essential constraints:

- Intermittent connectivity. Mobile devices are not always connected. Even when they are connected to data services, mobile networks manage to tear down the connections when there is no traffic for a certain amount of time or the connection exceeds some defined threshold.

- Network latency. In mobile networks, high bandwidth is not equivalent to high speed interactions with web applications, there are other important factors to take into account such as latency or packet loss rate. Furthermore, latency gets worse with increased bandwidth. As an example, a TCP initialization takes 5 to 7 seconds when using GPRS connections while in 3G networks it takes from 12 to 15 seconds.
- Battery life. Some common network operations such as opening connections or sending/receiving data have a very high impact in terms of battery life. Furthermore, the higher data speed we get the higher transmission frequency is needed which results in a higher power consumption rate.

The solution presented in this paper improves remarkably the constraints mentioned before.

1.2 Access to mobile resources

Mobile devices store Personal Information, provide Phone Network features and are able to generate events. Most of this is tightly integrated with the mobile network (making and receiving phone calls or SMS, divert, mobile services) and with the device itself (battery status, contacts, calendar, phone status). But, as far as the authors are concerned, they are not aware of any standard way to expose this data and functionality and make them accessible.

The solution provided in this project treats every mobile service and every mobile piece of information as a Web resource, it means, actually allocating a URL to each resource. Once this is established, REST principles provide the basic verbs to interact with these new resources (GET, POST, PUT, DELETE)

1.3 Push mobile generated events

Mobile network architectures are complex and tightly coupled. Mobile events usually trigger different kinds of services and actions in the mobile network. However, there is no easy way to distribute this kind of event information to the internet.

This project presents a solution to connect mobile and internet architectures and provide a way to collect mobile generated events and make them accessible to the internet networks.

2. PLATFORM DESCRIPTION

2.1 Overview

The interaction between devices and Web application is based on a Push approach. The heart of the platform is the Push Gateway which makes the bridge between mobile networks and the Internet by pushing common HTTP transactions to mobile devices.

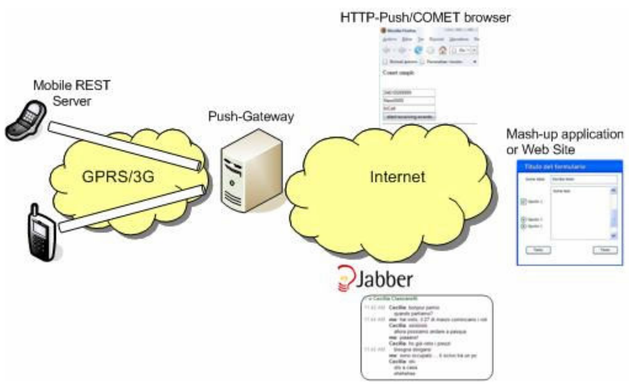


Figure 1. Functional architecture.

The infrastructure developed provides the next key features:

- Always-on connectivity for mobile devices
- A mechanism to easily deploy REST services in mobile devices.
- An addressing system to send messages to mobile devices by specifying a phone number and the REST resource being accessed. As an example, a GET request over this URL, http://push-gateway.com/34610599999/Battery, returns the battery status corresponding to the phone specified.
- A system to propagate device-originated events to internet peers. There are two ways of propagating these events: XMPP and AJAX/COMET clients.

2.2 Architecture description

Figure 2 summarizes the general architecture of the solution.

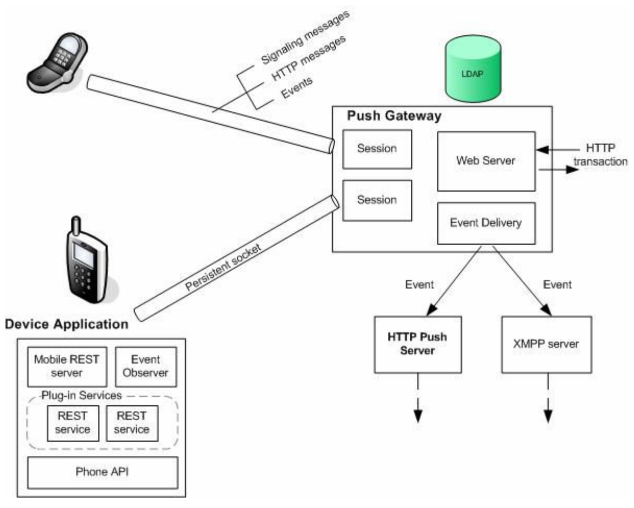


Figure 2. Logical architecture.

The solution consists of two main parts:

- A server infrastructure that
    - maintains the persistent connections with each device,
    - pings the connected devices periodically to keep the connection alive,
    - authenticates and authorizes new connection requests against a LDAP server,
    - redirects incoming HTTP requests from the open Internet to the suitable device,
    - propagates notifications from mobile devices to external peers connected through a COMET browser client,
    - propagates notifications to XMPP contacts,
    - receives requests embedded in messages from XMPP contacts and delivers them to the corresponding device.

- A device application that
    - initiates a persistent connection with the gateway using a mobile data connection (GPRS/EDGE or UMTS),
    - receives HTTP requests invoking a certain REST resource and dispatches them to the module that implements them,
    - sends a message to the gateway through the connection when certain predefined events occur,
    - reconnects with the gateway when the connection drops

2.3 Push Gateway

It provides routing capabilities and makes mobile devices addressable on the open Internet, despite the nature of the mobile data connection. So Push Gateway works with private IP addressing, behind NATs and firewalls. It redirects incoming HTTP requests through the correspondent persistent connection to mobile devices. It also receives events originated in mobile devices and sends them to the event delivery subsystems: the XMPP server and the HTTP-Push server.

The persistent connection is always initiated by the mobile device. When established, this connection is managed in the server by a session object which is in charge of managing HTTP traffic and the internal signaling needed to coordinate both sides of the conversation. This session is also in charge of keeping the connection alive, otherwise intermediate proxies could terminate it after a few minutes. This is mainly important when dealing with mobile data connections, which are allowed to have very short live time. So the PushGateway pings periodically all the open connections to check their status by sending a very short message (2 bytes).

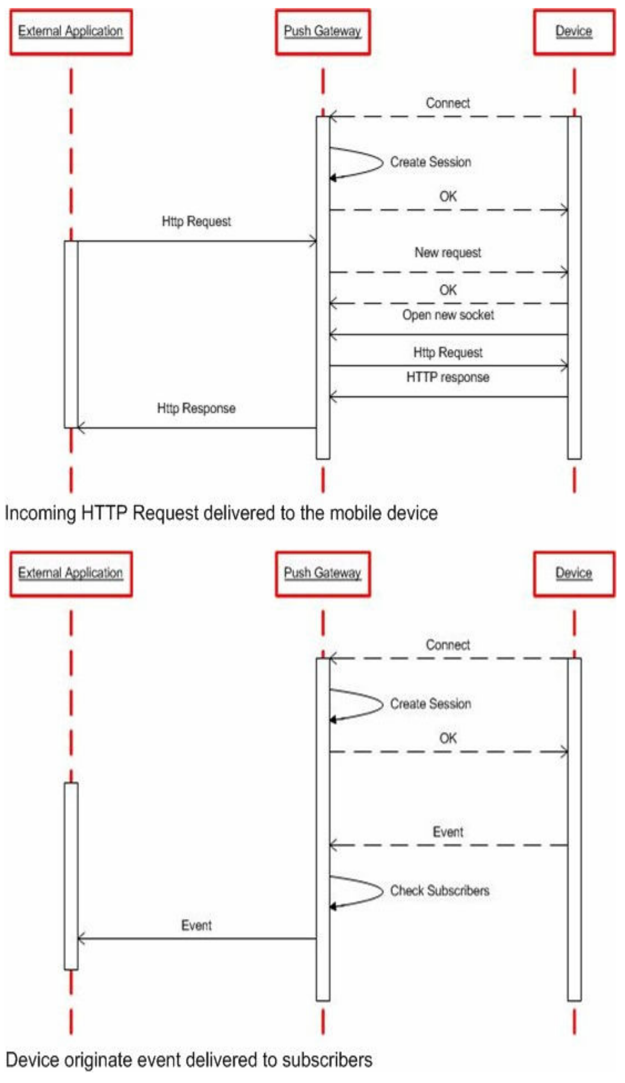


Figure 3. Messaging workflow.

When a new HTTP request arrives, the server identifies the session through the phone number item within the URL. Then, the server sends a signaling message to the correspondent device informing it that there is a new HTTP request ready in the PushGateway. The device then opens a new socket to the same server endpoint to receive the request which is passed to the suitable REST service. The response is then re-sent to the gateway which delivers it to the initial caller.

The device also has an observer that listens for events from the device O.S. and delivers them to the Push Gateway through the persistent socket. The Push Gateway sends these notifications to the defined delivery channels which currently are the XMPP server and the HTTP-Push server.

The whole process is depicted in figure 3.

2.4 Mobile REST server

HTTP requests coming are redirected by the PushGateway to the suitable device according to the following URL construction rules:

- The URL contains the destination phone number (MSISDN).
- The URL contains the resource being invoked.
- The HTTP verb performed over the URL determines the action executed in the device.
- Supported HTTP verbs are GET, PUT, POST and DELETE.

The project also comprises the development of two phone resources: Battery and SMS (Short Message Service).

As an example, performing a HTTP GET over the URL http://push-gateway.com/34610599999/Battery should return the battery status, while a HTTP POST over the URL http://push-gateway.com/34610599999/SMS should force the device to send a SMS message which content and destination should be specified in the HTTP POST request body.

Furthermore, the adopted model is extensible and supports adding new resources in a very easy way. New modules are added as plug-ins, just deploying software libraries with the same name of the resource being implemented. The mobile web server dynamically loads these plug-ins when a new resource is accessed. Then it searches for a class implementing the IPushGatewayService interface which defines the four methods corresponding to the four HTTP verbs supported:

```
interface IRESTService
{
 HTTPResponse Get (HTTPRequest message);
 HTTPResponse Put (HTTPRequest message);
 HTTPResponse Post (HTTPRequest message);
 HTTPResponse Delete (HTTPRequest message);
}
```

As an example, these would be the steps executed when a new request arrives to the mobile server. Supposing that someone sends a HTTP message to access the SMS repository to send a new message, the requester URL sent to the gateway should be something like this:

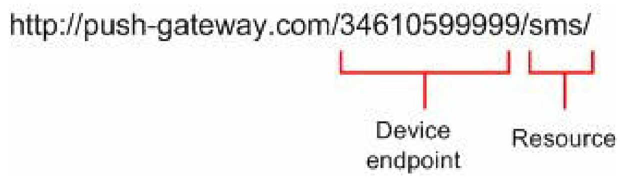


Figure 4. Example of access to sms resource.

So, in order to send a new text message, the message would post the next XML data:

```
<sms>
 <to>34610588888</to>
 <msg>testing the mobile server</msg>
</sms>
```

The gateway uses the persistent connection that it maintains with the specified phone number (3461059999) and resends the request to the mobile REST Server. It will extract the resource from the URL (SMS) and will dynamically search and load a library called sms.dll. Then, the server will search for a class implementing the IRESTService interface and will create an instance of it. Finally, it will invoke the "post" method passing the posted XML data. Once the message has been successfully sent, the service will respond with a "200 OK" HTTP response.

Adding new services to this platform is a very simple task. There are no complex configuration files or registry settings. Provisioning new services is as easy as downloading signed code libraries and copying them to the suitable mobile device folder.

2.5 Handling events

Handling events is one of the key parts of the system. Users are allowed to subscribe to mobile generated events and receive notifications using the notification channel they choose (SMS, e-mail, HTPP-Push, XMPP, MMS). The next sections will delve into details of two of the notification channels used: HTTP-Push and XMPP.

2.5.1 HTTP-Push server

The platform contains a asynchronous HTTP module that sends events to connected browsers supporting HTTP-Push.

HTTP-Push or Comet proposes a smarter interaction between web applications and clients than the typical polling approach from AJAX code. So, an asynchronous server is able to send multiple notifications to web browsers when an event occurs using a single HTTP transaction.

The implementation is based on a feature of the HTTP protocol. The server uses the multipart/x-mixed-replace content type when sending a response. This content type is expected to send a series of documents one after the other, where each one will replace the previous one.

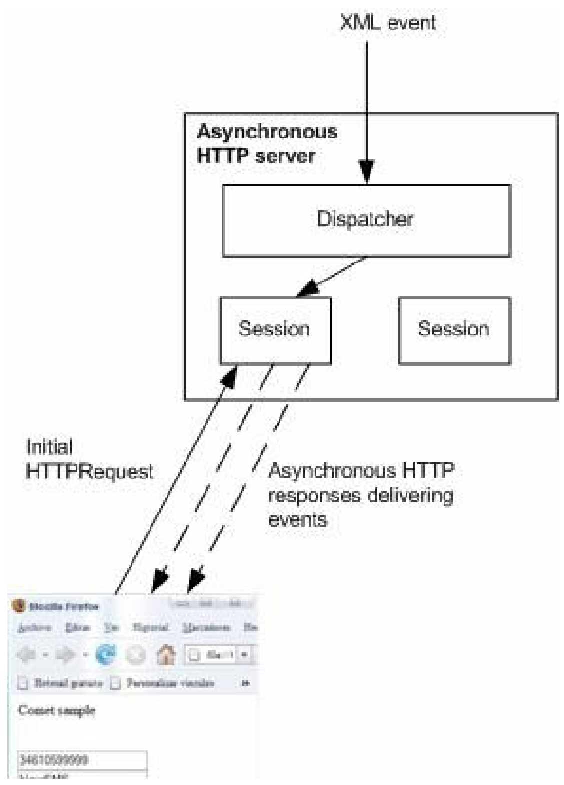
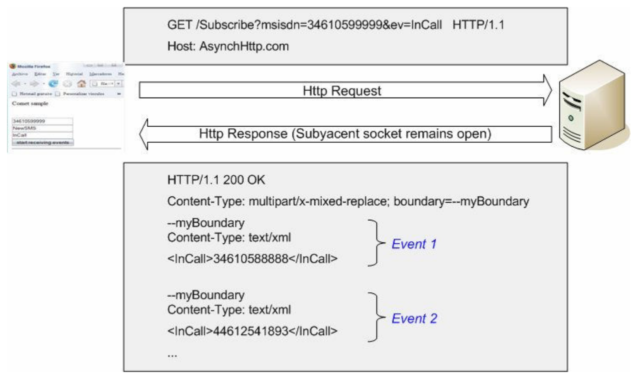


Figure 5. HTTP-Push server mechanism.

The only problem with this technology is that only Mozilla based browsers (Firefox) supports HTTP-Push nowadays.

The server is accessed from a web page containing DHTML and embedded JavaScript/AJAX code.



```
GET /Subscribe?msisdn=34610599999&ev=InCall   HTTP/1.1
Host: AsynchHttp.com
```

```
HTTP/1.1 200 OK
Content-Type: multipart/x-mixed-replace; boundary=--myBoundary
--myBoundary
Content-Type: text/xml
<InCall>34610588888</InCall>

--myBoundary
Content-Type: text/xml
<InCall>44612541893</InCall>
...
```

Figure 6. Event notification with the HTTP-push server.

The web page makes an initial GET request specifying optionally in the query string parameters to filter the source and the event types. This initial connection is then used to deliver all the subsequent events matching the specified filter. So, all the events are received in the same HTTP transaction.

2.5.2 XMPP

XMPP (eXtensible Messaging and Presence Protocol) is an open and extensible XML-based protocol which powers a wide range of applications including instant messaging, presence, media negotiation, whiteboarding, collaboration, lightweight middleware, content syndication, and generalized XML routing.

So, for the interest of this project, XMPP networks are PUSH networks by design, which means they are able to act as a powerful distribution network for mobile based events.

The solution selected in this project has been to create a controlled XMPP user for every mobile device connected to the system. This XMPP user is managed by the Push Gateway, which has an extension to provide two basic features:

- Receive events generated from the mobile device and distribute them to the XMPP network.
- Receive commands from the XMPP network to interact with the mobile device.

Each device connected to the system has an XMPP address like myphone@mydomain.com allocated. The only action that any user interested in receiving events from the mobile phone has to do is to add this XMPP address to their usual XMPP client. º The Event Delivery module included in the Push Gateway includes an XMPP client. When a new event is generated on the mobile phone, the aforementioned XMPP Client transforms the event into an Instant Message, which is distributed to the buddy list defined by the user. This simple mechanism lets the system distribute mobile events to the XMPP networks.

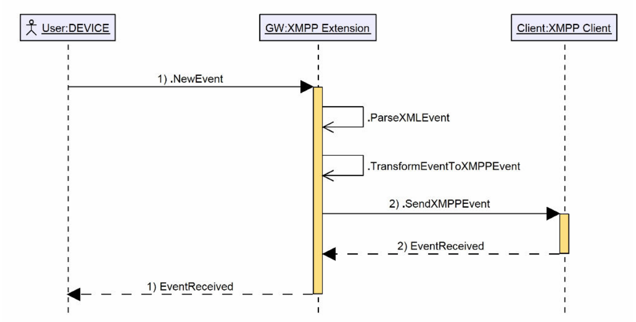


Figure 7. XMPP server mechanism.

On the other hand, a common XMPP client is able to control the mobile device via Instant Message commands, provided the user is authorized to do it. In this case, the external XMPP client sends an Instant Message to the mobile phone XMPP address, following an ad-hoc message used in this project. The XMPP client included in the Push Gateway receives the Instant Message, transforms it into a XML message and sends it to the mobile device to perform an action.

The following is an example of the code used in the Instant Messages to send a SMS:

- *Description: Sends messages or emails to another device*
- *Syntax: SEND 'SubCommand':'text'*

- *Subcommands: SMS [u/'phone number', c/'contact name', g/'group']*

For instance, *SEND SMS 'John Doe': 'Hi John, How's life?* would be a valid command.

3. FURTHER STEPS

The platform presented in this paper provides a significant amount of new features that can be used by other services. There are plans to add new capabilities to the platform and, thanks to its modularity, to integrate it with other platforms in order to multiply the advantages they offer. These further steps are:

- Data compression: Using data compression will allow reduce traffic over the air interface.
- Integration with mobile operator network authentication.
- Extend the platform to work with any kind of event distribution channel.

3.1 Services

There are hundred of different services that may be improved using the features provided by this solution. Because of the REST API, all kind of services can easily be deployed on the mobile phone.

For instance, remote control of mobile devices could become a reality. Ad-hoc sensor networks can use the mobile device as a mechanism to propagate real-time events and measurements. Ambient intelligence systems or domotic systems may take advantage of the benefits of the system and create rich, fast applications which are allowed to be accessible and to propagate events.

4. SUMMARY

The solution presented in this paper provides a new way of interaction between Web applications and mobile devices, overcoming the connectivity limitations present in mobile data connections. The adoption of an always-on approach supposes an improvement from typical polling based solutions. This results in battery life savings and bandwidth optimization.

It proposes a way of integrating mobile devices in existing Web solutions and mashups. The project provides a Web address to each device, a way of routing web traffic towards them and a way of delivering events to Instant Messaging solutions and HTTP-Push browsers.

So with this solution a mobile device is able to participate in distributed Web applications acting as a REST service provider by using an extensible mechanism which allows the deployment of new services in a very easy way. It is a platform or enabler that allows the integration with existing IT solutions.

There are many applications of this concept. The clearest one is to provide access to personal information and resources stored on the mobile phone (contacts, messages, calls...). It will also be particularly interesting for real-time monitoring applications (such as e-health solutions) because the developed platform would provide access to sensors connected to the phone and a way of implementing alerts through events.

5. REFERENCES

[1] XMPP protocol overview. http://www.xmpp.org/about/

[2] A hybrid push-pull mechanism. http://dev.llup.org/wiki.

[3] Hypertext Transfer Protocol -- HTTP/1.1 http://www.w3.org/Protocols/rfc2616/rfc2616.html

[4] Architectural Styles andthe Design of Network-based Software Architectures. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[5] COMET. http://alex.dojotoolkit.org/?p=545

[6] Mobile Messaging and Business Application Solutions with Windows Mobile: Addressing Key Questions. Microsoft, 2006

[7] Bozdag, E., Mesbah, A. and van Deursen, A.: A comparison of Push and Pull techniques for Ajax. Delf University of Technology, 2007.