Celebrio – Real time communication

Seminar thesis for LaSArIS (PV226)
Pavel Smolka, 324612

The objective of this document is to describe the progress of Celebrio development in last year, especially the domain of real time communication on which I focus the most. Since we have decided to create independent web operating system and we claim to provide all the services elderly need (and the communication is one of the most stressed), there is strong need of reliable communication service. However, one of the most important things for the startup project is interoperability and ability to connect to existing services, such as Skype, Google Mail, Google Talk or Facebook chat in this case.

Although implementation of text-based communication services is more straightforward than multimedia support, we aimed to the talk via audio (and video) from the beginning. First and foremost, it's the simplicity of using such services which makes them not only technological gem of the system, but also great marketing advantage. That's why we assume this part of system to be one of our core fields. I am glad to set out it is me who is responsible for it,[1] even though it's not the easiest part.

Skype failure

Skype is the best known real time service which supports audio and video. That was the reason we had decided to implement it to our system. I am not very familiar with the technical details since not me but Petr was response for it (this attempt took place during the fall and winter, about half year ago). There are two main disadvantages which make integrating Skype to the web OS like Celebrio almost impossible:

1. Skype uses closed protocol to establish and perform the communication between peers. There is no such possibility to implement our own service and connect it to the Skype network, no matter if it would be based on REST, permanent TCP connection or any other technology. Unless...
2. Skype provides client API to create your own client. In fact, not real client, but wrapper around the existing client.[2] That perfectly solves the issue mentioned in number 1. Unfortunately, not in web OS. The “headless” SkypeKit runs only in conventional environment of operating system serving directly the Skype user, as any other desktop application. In such case, we would have to implement the Skype as standalone application on the server computer, which is impossible due to several reasons – extreme performance load, impossibility to serve too many users from one client and implementation difficulties with connecting the users via our web server.

After all, we tried to use IMO-IM service.[3] That's website whose authors successfully hacked Skype protocol and used it for their own purpose (though I doubt about their fulfilling Skype terms of service). We successfully connected to their connection and implemented “proof of concept” application. However, after several days, we realized there is a problem. Either they disclosed our “service hijack” or they secured self in advance, their “API” changes very frequently so it's impossible to predict it and adapt to it. That led to closing the Skype integration to Celebrio project as unsuccessful. However, I personally admire Petr's work with (very unstable and unpredictable) IMO.IM service.

XMPP – Text-based communication

Educated by the mistakes we fell into during the Skype implementation, I decided to try something more

1 http://code.google.com/p/celebrio/issues/detail?id=234
2 http://developer.skype.com/public/skypekit
3 https://imo.im/

reliable. In other words, open protocol. One of the best for text-based communication is XMPP – Extensible Messaging and Presence Protocol.[4] Not only does it provide reliable multi-platform support but also it contains set of useful extensions.[5] There is also huge advantage of using XMPP for inter-process communication if we master it.[6]

It sounds XMPP is perfectly tailored for Celebrio. Unfortunately, the world is every time completely different from how it seems to be. As already mentioned, Celebrio is based on web technologies and every client view, chat window displaying the incoming messages and receiving the input text from the end user, runs in the web browser on the client's computer. And this web browser does not have any permanent TCP connection to our server. And XMPP, not surprisingly for such protocol, uses permanent duplex TCP connection to establish quick real-time communication.[7]

BOSH – XMPP over HTTP

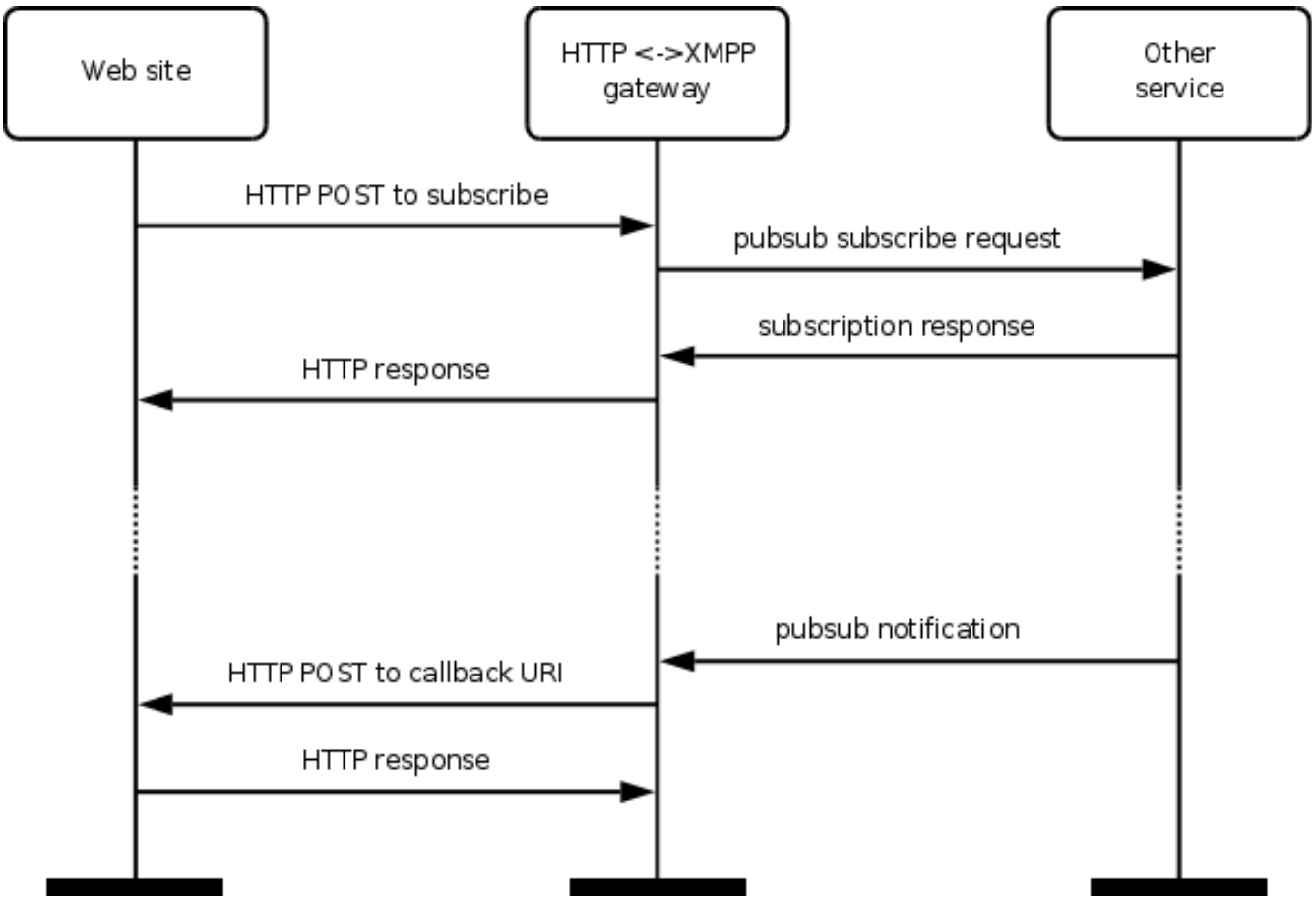
So, having described the problems in the previous paragraph, how do Google or Facebook do it, that they run chat service in the web browser? They use the principle of bidirectional streams over HTTP, implemented in BOSH extension for XMPP.[8]

Note: There are other possibilities which came in HTML5 to establish permanent TCP connection between browser and the HTTP server such as web sockets. We decided not to use such technology because it is not mature yet, the standard for implementing XMPP over WebSockets has not been finished yet[9] and unlike BOSH, there are no libraries for simple usage. There are some attempts for server-side support, but only for ejabberd, the most widespread XMPP server.[10] Unfortunately, we haven't managed to run it on Windows Azure platform yet.

Now back to BOSH. The idea is simple, it's the same thing that special agents in thriller movies use. Although the agent (client browser in our case) is hidden and the headquarters (HTTP server) does not know about her location, they exchange messages in both directions – agent notifies the headquarters and every time he carries out the phone call, he also receives new instructions. That's the first step, we can ask the server on the regular basis and receive news from it. The second and last step to BOSH is the thing that the server keeps the request for a longer time (30 seconds is usual delay) so it's able to answer immediately when any action happens. In fact, there are two communication streams. Formally, both initiated by client (it's necessary since we are in HTTP environment) but one of them stays at the server and it responses if and only if the server needs to notify client – for example when a new message for that client arrives. The server sends the message to the client and the client answers immediately with the pending request. The situation is described on the Communication scheme for BOSH.[11]

---

4 http://xmpp.org/
5 http://xmpp.org/xmpp-protocols/xmpp-extensions/
6 There are plenty of completed solutions based on XMPP just off the shelf, for example chess game: http://xmpp.org/extensions/inbox/chess.html
7 http://tools.ietf.org/html/rfc6120#page-16
8 BOSH: http://xmpp.org/extensions/xep-0206.html
9 http://stackoverflow.com/questions/1850162/is-there-an-open-source-websockets-javascript-xmpp-library/1875848#1875848
10 https://github.com/superfeedr/ejabberd-websockets
11 The scheme picture source: http://idavoll.ik.nu/raw-attachment/wiki/HTTP_Interface/gatewayed_subscription.png



Illustraion 1: Communication scheme for BOSH

Now, little personal confession... At first, I totally misunderstood the point of BOSH. Since we don't run our own XMPP server, we have to connect to the existing public XMPP service. I tried to establish the connection between our HTTP server and the remote XMPP server using BOSH. That would work, but it would be totally useless in our case. However, I spent almost two weeks by trying such a silly thing. At least, I explored a bit two existing libraries which provide BOSH for PHP – XMPPHP[12] and Jaxl.[13] The former seemed to be quite bugged and I got it work only after rewriting several parts of the library. The latter looked quite better and I managed it to communicate flawlessly with both jabber.org and jappix.com services. However, the most desirable Google Talk, declined connections from other jabber accounts than Google itself.

Strophe JS – Javascript library for BOSH

After realizing that BOSH needs to be implemented directly in the web browser, I started looking for the suitable library written in Javascript. Fortunately, there is (from my experience up to now) very good and reliable library initially created by Jack Moffitt and later improved by community – StropheJS.[14] [15] It provides very highly abstracted access to the XMPP protocol. I haven't managed to explore all its possibilites but I successfully implemented text-based communication for Celebrio which is accessible in the public version of the system.[16]

12 http://code.google.com/p/xmpphp/
13 https://github.com/abhinavsingh/JAXL
14 http://strophe.im/
15 Strophe.JS sources: https://github.com/metajack/strophejs
16 http://system.celebriosoftware.com/

Since we don't own our XMPP server, we use metajack.im server (provided by the authors of Strophe). Little exapmle of the usage comes here, assuming that `jid` and `pass` are variables containing valid XMPP credentials, for example "pavel.smolka@jappix.com" and "mypassword". Parameter `onConnect` is the callback function which is called when the library finishes connecting, either successfully or not.

```
var BOSH_SERVICE = 'http://bosh.metajack.im:5280/xmpp-httpbind';
var connection = new Strophe.Connection(BOSH_SERVICE);
connection.connect(jid, pass, onConnect);
```

Not only does the XMPP protocol allow to control incoming and outcoming messages between peers but also presence of the peers. However, "who is online" functionality hasn't been implemented yet. On the other hand, I managed implementation of subscription – i.e. the functionality widely known as "someone wants to chat with you". It is performed automatically when Celebrio user adds someone to her contact list in People application. The library also provides very neat functionality of "persisting" connection when moving on the website (changing URL and therefore losing JS context). After all, Strophe.JS seems to be good choice.

Jingle – Long way to go

I have two big tasks for this summer: To improve existing text-based Talker to provide all the functionality we are used to receive from conventional XMPP clients – presence distinction, new message notification and a lot of others. The second task is even more difficult and it's the original objective we set up for communication in Celebrio – audio and video chat.

There is XMPP extension which provides multimedia transfer called Jingle.[17] It is able to transfer both video and audio but I haven't discovered whether it works over BOSH. However, Google talk uses Jingle (or at least the basics of it) for its communication[18] and they actually ARE able to get it work. If they can, we can do it too!

There's a long way to go. I hope it won't be necessary to implement our own (or use the existing) browser plugin as everyone who wants to use Google video talk needs to install such. It won't be problem when using Celebrio client but it would become annoying for the users working with the system directly in the web browser. There is possibility, in the new versions of web browsers – I mean really the newest, to access the webcam directly from javascript, similarly as geolocation.[19] It might be one of the ways we will take. Also, we won't develop audio/video from scratch. Apart from the Jingle extension, there is Google Hangout API published this spring[20] and still being rapidly developed and changed. It seems Google considers video calling in browswer to be one of the core features of its social network. And so do we.

Summary

There has been implemented basics of text-based real time communication in Celebrio using Strophe.JS library and BOSH. Now we explore the possibilities about the audio/video calling, using Jingle, Google Hangout API and new javascript features providing access to camera and microphone.

17 Jingle: http://xmpp.org/extensions/xep-0166.html
18 https://developers.google.com/talk/open_communications?hl=cs#protocols
19 http://paulrouget.com/e/getUserMedia/
20 http://googleplusplatform.blogspot.com/2012/03/moving-google-hangouts-api-out-of.html