Introduction

This project demonstrates a possible generic solution that may be used for developing mobile tango clients. The following technologies are used in this project:

PhoneGap (Cordova) [http://phonegap.com/]

JavaScriptMVC [http://javascriptmvc.com/]

jQuery Mobile [http://jquerymobile.com/]

Java Servlet [http://www.oracle.com/technetwork/java/index-jsp-135475.html]

Tango Client Java API [http://www.tango-controls.org/download/tangorb]

Basic understanding of these technologies is strictly welcome.

Requirements

Java SE 1.6

Tango 7 (access to TangoTest server)

Android SDK

Apache TomCat 6 (or later)

Apache Maven

Apache Ant

Getting started

You have to have Java SE 1.6, Apache TomCat 6 (or any other servlet container) and android SDK installed. You also need an access to TangoTest device (which is the part of the Tango distribution).

In the text below several shortcuts are used:

{PRJ_ROOT} – a location where .zip archive was unzipped or repository was cloned.

{TOMCAT} – TomCat's home directory

{TANGO_SERVER} – a tango server name, aka "sys/tg_test/1"

{JMVC_ROOT} – a root folder of the javascript mvc under {PRJ_ROOT} = mTango_prototype_web/src/main/javascript-mvc-1.5

Setup web server

1. Copy {PRJ_ROOT}/distr/mtango.war to {TOMCAT}/webapps

2. Open mtango.war with any archiving program (WinRAR, 7-Zip etc)

3. Edit mtango.war/WEB-INF/web.xml: change TangoProxyServlet's initial parameters to the values that match your environment.

```
<servlet-name>TangoTestProxyServlet</servlet-name>
<servlet-class>hzg.wpn.mtango.TangoProxyServlet</servlet-class>
<init-param>
    <param-name>tango-host</param-name>
    <param-value>hzgharwi3:10000</param-value>
</init-param>
<init-param>
    <param-name>tango-device</param-name>
    <param-value>sys/tg_test/1</param-value>
</init-param>
```

Servlet connects to hzgharwi3:10000/sys/tg_test/1

4. Adjust url-mapping if necessary.

```
<servlet-mapping>
    <servlet-name>TangoTestProxyServlet</servlet-name>
    <url-pattern>/sys/tg_test/1</url-pattern>
</servlet-mapping>
```

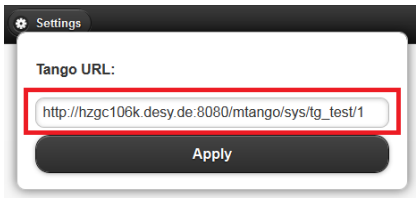
Client connects to http://{TOMCAT}/mtango/sys/tg_test/1

5. Start server ({TOMCAT}/bin/startup)

… with web app

1. Open browser navigate to http://localhost:8080/mtango/web-client/apps/mTango-web you should see web app interface similar to one in Fig. #. Otherwise something is wrong.

2. Set proper tango-url in Settings (tango-url = {TOMCAT}/mtango/{TANGO_SERVER}).

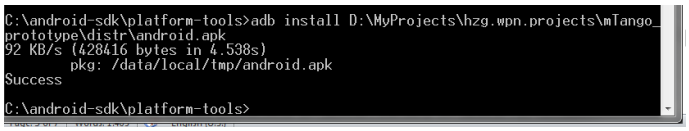


3. The application demonstrates basic features such as reading/writing attributes, executing commands and some sample app where double_scalar attribute is being read and displayed in a plot.

… with android

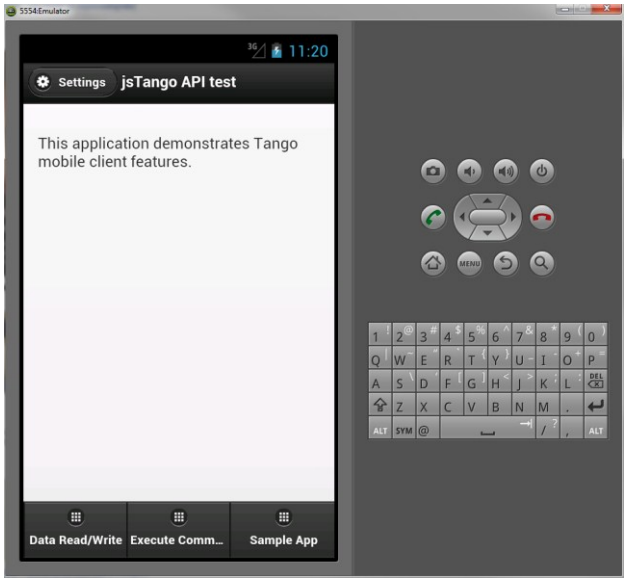
1. Connect your android device to the PC or start emulator

2. In cmd execute: adb {PRJ_ROOT}\distr\android.apk. This installs android application.



3. Make sure that web server is set up and running (See previous section)

4. Run mTango-android application on your device or emulator. You should see application interface similar to what is in the picture below. Functionality of this application is obviously equal to web application's functionality (they share a common code base).



5. Set proper tango-url.

Application overview

Application has three pages (tabs): Data Read/Write, Execute Command, Sample App. Each of them demonstrates common use cases of the tango client. The first – reading/writing attributes, the second – command execution and the third – some trivial application.

All the pages have friendly and intuitively understandable interface.

Solution general overview

Solution introduces three layers: client layer; web server layer and Tango layer (Fig. 1). Tango and web server layers are located within single network. Web server can be seen "outside" that network. Client can be either a web browser or a mobile device.

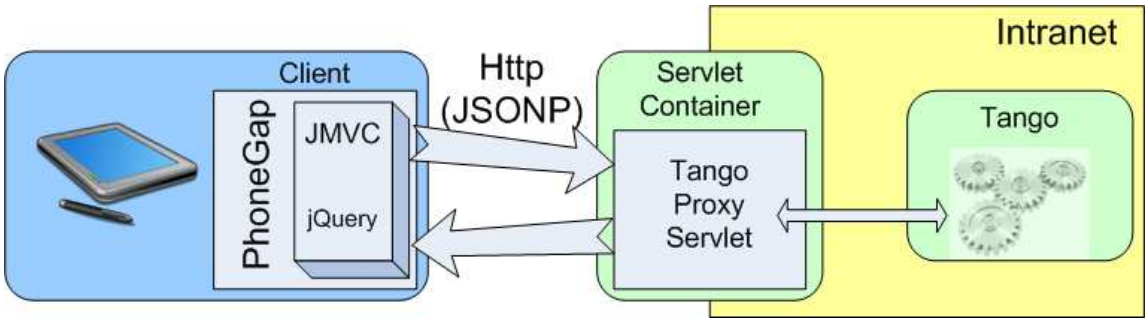


***Figure 1**. JavascriptMVC + jQuery mobile + PhoneGap = mobile app; Java Servlet = http gateway; Tango = Tango*

Tango layer consists of only a Tango process (or Tango server).

Web server layer consists of a servlet container. In case of this solution it is Apache TomCat. Into the servlet container a web app is deployed. This app remains unchanged for many installations; the only thing that needs to be adjusted is a servlet description in web.xml - this can be automatized.

Client layer consists of a javascript rich interface client and a PhoneGap. To implement rich interface JavaScriptMVC together with jQuery Mobile is used. Empty client code can be automatically generated.

Detailed overview

Web client

The main framework that is used in web client (and mobile) is JavaScriptMVC – javascript model-view-controller implementation. This framework provides great possibilities: testing; documenting; compression; automatization etc. For us it is important that using this framework we can develop standalone application which just connects to the server to update its state. Therefore we are completely independent from the server side during development. And this perfectly fits into PhoneGap's ideology – wrap javascript application into a native application.

To develop interface widgets we use jQuery Mobile. jQuery Mobile makes it simple to develop comprehensive tap friendly interface with less efforts.

In general development cycle is as following:

- auto generate jsTango API .js file (in case of this application – {PRJ_ROOT}/{JMVC_ROOT}/models/TangoTest.js)
- develop application
- test
- compress & document

- distribute to a web server or a mobile device (i.e. copy it to PhoneGap and then assembly mobile application)

It is possible to fully automatically generate files like TangoTest.js from the same templates which are used for C++ or Java generation. In case of this application this file was written manually while developing such a generator is more or less time consuming task. This file contains js code that provides an API of the corresponding Tango server. Using this API any web developer can implement application.

Below is a quick guide through application's source code.

The main entry point of the application is located in {PRJ_ROOT}/{JMVC_ROOT}/mTango-web_controller.js#load method. This method is like a main method in C++ or Java. In case of our application in this method Tango device proxy is initialized:

```
        //create tango device

        var deviceUrl = $('#settings-tango-url').val();//read current tango-url

        GlobalContext.proxy = new TangoTest(deviceUrl);//create TangoTest and place into global
context
```

Then three pages are created and loaded:

```
        //load pages

        var readWrite = new Page("data_read_write");

        readWrite.load();


        var execCmd = new Page("exec_cmd");

        execCmd.load();


        var smplApp = new Page("smpl_app");

        smplApp.load();
```

And in the end corresponding controllers are initialized:

```
        //create controllers

        var readWriteController = new ReadWriteController("data_read_write",GlobalContext);

        GlobalContext.readWriteController = readWriteController;


        var execCmdController = new ExecuteCommandController("exec_cmd",GlobalContext);

        GlobalContext.executeController = execCmdController;


        var smplAppController = new SampleApplicationController("smpl_app",GlobalContext);
```

```
        GlobalContext.sampleAppController = smplAppController;
```

The last page demonstrates some sample application where a particular attribute is polled (double_scalar) and displayed in a plot:

```
//this method is from jsTangoTest API – it reads double_scalar attribute and executes onComplete function when receives response

proxy.get_double_scalar({

                onComplete:function(response){

                    $('#double_scalar').html(response.argout);//display current value on the page

                    //below jquery plot plugin's specific code which updates plot on the page

                    ...

                }

            });
```

The code snippet above is from {PRJ_ROOT}/{JMVC_ROOT}/controllers/SampleApplication_controller.js# startPolling method. Notice how Tango proxy is used here. It is simple!

JavaScriptMVC allows automated testing (both unit and functional tests). After testing JavaScriptMVC compresses the whole application into a single .js file that then can be easily distributed.

All the steps of the development cycle can be automatized with Apache Ant. For instance:

```
<target name="compress-javascript">

        <echo>Compressing javascript...</echo>

        <echo>Executing: js.bat apps/${appName}/compress.js</echo>

        <exec dir="${src}" executable="cmd" failonerror="true" failifexecutionfails="true">

            <arg line="/c"/>

            <arg line="js.bat"/>

            <arg line="apps/${appName}/compress.js"/>

        </exec>

    </target>
```

In the code snippet above the whole application is compressed into a single production.js file on Windows platform.

During compression documentation is being also generated. Generated documentation is located under folder {PRJ_ROOT}/{JMVC_ROOT}/docs.

Mobile client

Mobile client uses the same code base as the web client. After compression resulting production.js file is copied to the PhoneGap ({PRJ_ROOT}/android/asset/www/apps/mTango_web). That's it, now

PhoneGap application is ready to be deployed to the corresponding platform (android in the case of this application). This can be automatized with Apache Ant!

Web server

Web server has only one servlet class which can act as a http proxy to any Tango server. Which Tango server will be targeted is specified in web.xml:

```
<servlet>

        <description>Servlet stub for TangoTest</description>

        <display-name>TangoTest</display-name>

        <servlet-name>TangoTestProxyServlet</servlet-name>

        <servlet-class>hzg.wpn.mtango.TangoProxyServlet</servlet-class>

        <init-param>

            <param-name>tango-host</param-name>

            <param-value>hzgharwi3:10000</param-value>

        </init-param>

        <init-param>

            <param-name>tango-device</param-name>

            <param-value>sys/tg_test/1</param-value>

        </init-param>

    </servlet>


    <servlet-mapping>

        <servlet-name>TangoTestProxyServlet</servlet-name>

        <url-pattern>/sys/tg_test/1</url-pattern>

    </servlet-mapping>
```

This configuration can be automatically added during jsTango API file generation. Succinctly this means that requests to http://{TOMCAT}/mtango/sys/tg_test/1 will be translated to tango://hzgharwi3:10000/sys/tg_test/1 tango device.

Application development does not affect web server's code. This means that web developer does need to make any changes to the web server's code.

Tango server

Tango server is an independent Tango process that runs somewhere in the same network together with web server process.

Conclusions

To implement a mobile client web developer needs to download a single .zip archive. This archive already contains everything to start development except Tango specific files. It is possible to provide command line utilities which can generate all necessary files. After that web developer is able to implement any application which will act as a mobile client.

Most of the routine work can be further automatized – test, compress, deploy.

Using this solution any web developer is able to develop a tango mobile client using HTML, JavaScript and CSS incl. cutting edge HTML5, CSS3 and jQuery.

These were the pros. And here are the cons:

- complex project structure (many layers)

- slow (each request client first connects to a web server and then to a tango and back)

There is still a lot of room for further optimizations and improvements. For instance some comprehensive logic can be added to the servlet such as combine several request to a single tango server into one.