PROJECT REPORT

NAME OF THE PROJECT:

An Android application for monitoring the problems in the ALICE Grid

Prepared by:

Vladimir Ilievski, CERN summer student

Supervisor:

Predrag Buncic

CERN, august 2014

Abstract

This project is developing an Android application, which will help in monitoring of the ALICE Grid of computers. It uses a lot of Android related technologies, Cloud solutions and server side programming. It is supposed all users to get notifications for the events that are from their interest. This application will contribute to notify the users in real time, in which way the existing problems will be solved faster.

Contents page

Introduction

ALICE experiment possesses a computing infrastructure consisting from more than 100 sites. In this kind of environment errors can appear in every possible moment. The aim of this application is to help on the site administrators to prevent the errors as much as fast they can. Official ALICE monitoring system is on the web page http://alimonitor.cern.ch/. On the web page exists an alert XML feed which is automatically generated structure, describing the current problems in the computer sites around the world. Using this feed, the application instantaneously notifies the users when new problem arise. The application is consisted of many modules specific for the Android and a lot of modern technologies.

Main mechanism for notifying the users relies on Google Cloud Messaging service which provides reliable way for delivering notifications. The model in this application is Notification, an entity that contains all of the necessary data. With the help of Broadcast Receiver, the application listens for an event that there is something new in the alert XML feed. The occurrence of this event triggers the Synchronization Adapter to begin with downloading, processing and filtering the content from the web. Because the content is in XML format, special module parses the content and creates entities from the model (Notification). Then, with the using of Content Provider this entities are inserted into the phone data base. In the end, all of these entities are nicely presented on the screen with by means of Activities.

Content

1. Creating the model and designing the user interface

The model in this application is named Notification. The data encapsulated in the model is consistent with the data provided by the XML feed on http://alimonitor.cern.ch/. Namely, the model is consisted of: id, tittle, link, summary, content, category, starting time, ending time, status, importance and novelty flag.

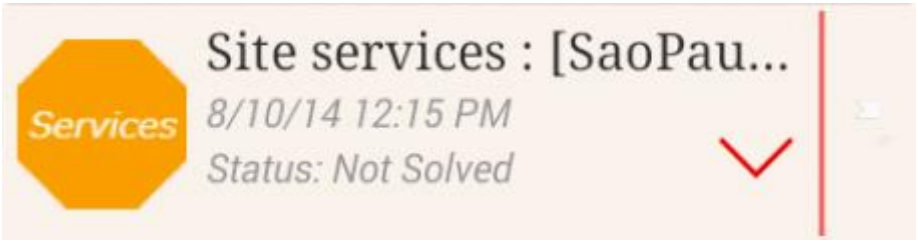
The id uniquely identifies the notification, the title gives the initial information about the notification together with the link. Summary is brief description of the problem, while content is extensive description. The category of the notification is related to the nature of the problem, so we have notifications for problems with the storage, proxies, site services and network. Also there are informational and category for everything else. The starting time denotes when the problem occurred, and the ending time when the problem was solved. The status is the flag that marks does the problem is solved or not, the importance flag indicates the importance of the problem for a given user and the novelty flag serves to remind the user that the notification is new for him.

Having the model as a baseline, we can build the user interface, the interactions and all of the manipulations that user can do. The user interface is consisting of three screens. One screen is for showing all notifications, the second for showing the details about the particular notification and the last one for setting the user's options. Due to the software reusability, the first two screens are composed of two fragments.

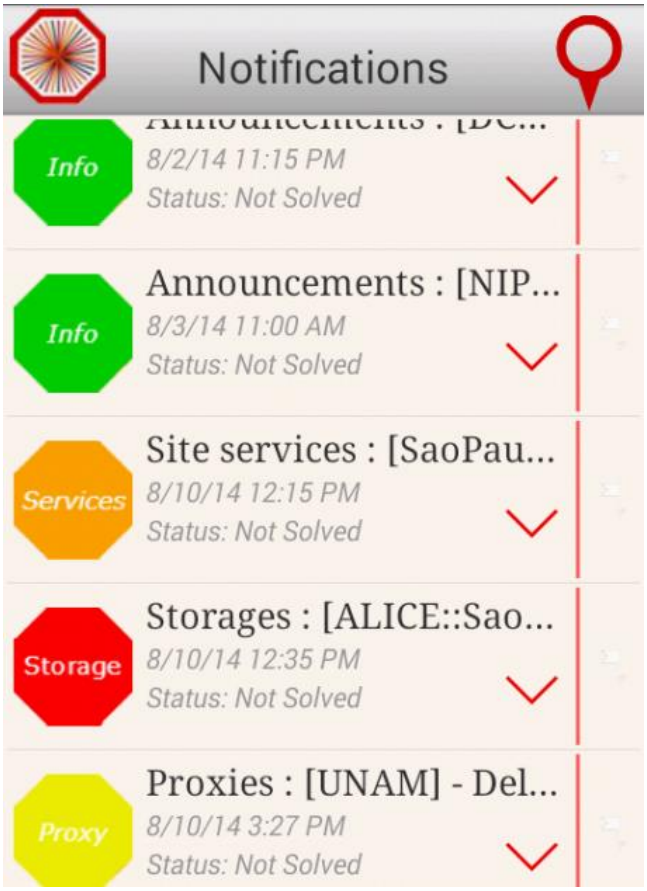
As a specific platform, Android separates the building of the user interface and the code behind the user interface. So, the structure of the user interface is written in XML, and then reference to this code is passed to the java classes later.

First I've created the fragment for the top of the screens. The class for this fragment is named *TopFragment.java* and the XML is named *top_fragment.xml.* Then in the *onCreateView ()* method of the *TopFragment.java* class we just bind the user interface. In the end the top fragment looks like this:

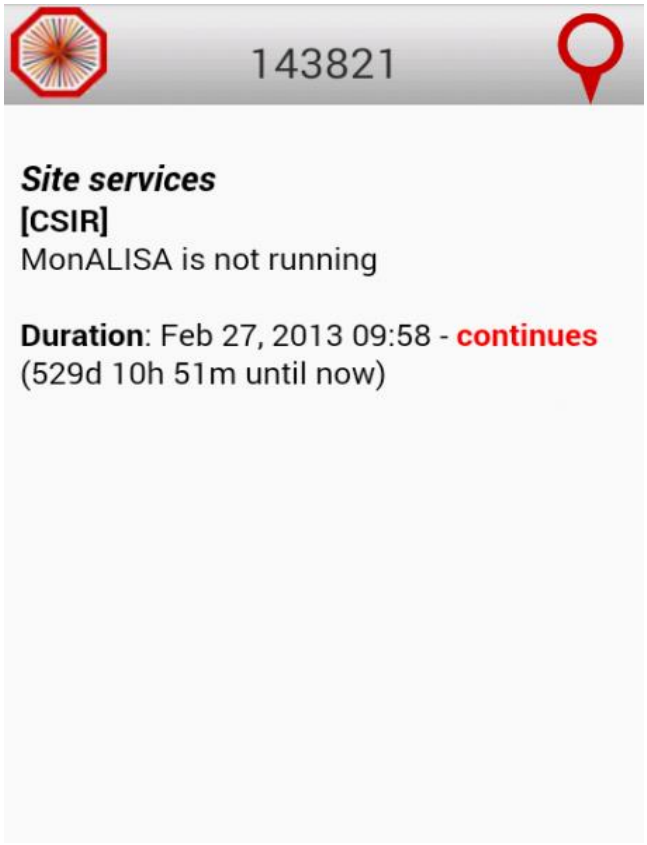
Other fragment is for showing the notifications in one list of scrollable items, so it is very simple and is consisting of only one ListView element. Because every Notification is constituted from different information it is needed to make one general pattern for showing every single Notification in one of the list's item. Special classes named Adapters act as a bridge between the lists and the underlying data for that view. So, I've made a special adapter for this named *NotificationAdapter.java*, which defines the view of every item in the list depending on the information in the particular Notification. This adapter as a pattern uses the XML file for one item which is named *notification_item.xml*. In this way, one item of the list will look like this:



After creating the two fragments, I've made the screen for showing all of the Notifications. Finally the screen looks like this:



After clicking on one of the items in the list, the screen for showing the details about the notification is appearing. This screen also uses the TopFragment and other graphical interfaces to show the content. This is the look of the screen:



The third screen is still in developing phase.

2. Creating Content Provider and the XML parser

In order to insert the notifications in the mobile data base there must be Content Provider, a helper class with which you can insert, delete, update and query in the database. For this purpose I've made a class named *NotificationContentProvider.java* with a lot of helper methods, only to be able to manipulate data. This class will serve me to insert the notifications once the application will download them from the alert XML feed. But, the content that will be downloaded is in XML format, with special XML tags, so there is a need for creating module that will parse the content and will create entities from that. One notification as a XML code is represented as:

```
<entry>
      <id>161538</id>
      <title type="html">Storages:[ALICE::ZA_CHPC::SE]-ADD test
fails</title>
      <link rel="alternate"
href="http://alimonitor.cern.ch/stats?page=SE/table"/>
      <published>2014-07-25T12:45:20.000Z</published>
      <updated>2014-07-25T12:45:20.000Z</updated>
      <summary type="html">Something</summary>
      <content type="html">Something</content>
      <category term="Storages">Storage</category>
</entry>
```

So I've build special purpose class named *AlimonitorXmlParser.java.* This class contains special functions for parsing the information from every tag. With this piece of data it creates objects from the Notification class and passes it to the Content Provider class.

3. Configuring Google Cloud Messaging Service

Google Cloud Messaging is the technology that enables reliable and very easy way to send data from servers to Android applications. This could be a lightweight message telling the Android application that there is new data to be fetched from server. Prerequisite for this is the device must be already connected to the network. For this reason, the application contains a separate Broadcast Receiver named *NetworkReceiver.java*, to listen for network connectivity changes.

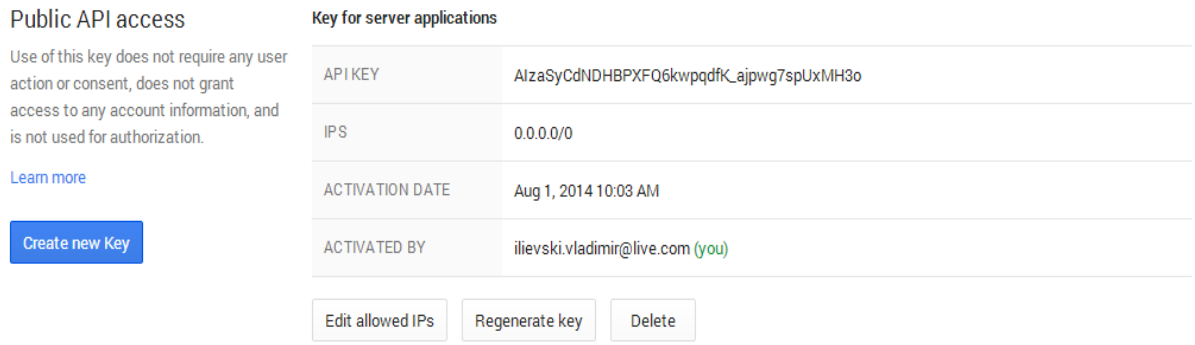
When the device is connected to the network, first it sends a sender id and application id to the GCM server for registration. Upon successful registration, the GCM server issues a registration id to the Android device. After receiving registration id, the device will send the

registration id to our server. Then our server will store registration id in the database for later usage. So, whenever push notification is needed, our server sends a message to the GCM server along with the device registration id. Then the GCM server will deliver that message to the respective mobile device. This is the main mechanism behind GCM.

For this to work we need to configure our application. First I've created new project on https://console.developers.google.com named Alimonalisa. Then from API & auth section

I've activated Google Cloud Messaging for Android API:

After that, from Credentials section I've generated API key. This API key is needed for authentication of the 3-rd party server:



Upon successful API key generation, I've created *GCMClientServices.java* class. With a help of this class, the app check the device for a compatible Google Play services APK, register with GCM servers and store the registration id for future use. The next thing was creating GCM Broadcast Receiver. So, I've created *GcmBroadcastReceiver.java* class and with this module, the application listens for event that there is something new on the server. For all of this to work in the *AndroidManifest.xml* I've put all of the necessary permissions. With all of these steps the client side of the Google Cloud Messaging system is done.

Only for testing purposes, I've downloaded almost prepared code for the server side. I've built and booted the server using the Google App Engine and Maven technologies. In the server side code I've just put the previously generated API key for the authentication of the server with GCM. In the end I tested all of this, and it was successful.

4. Implementing Synchronization Adapter Framework

This framework helps manage and automate data transfers, and coordinates synchronization operations across different apps.

The sync adapter framework assumes that the sync adapter transfers data between device storage associated with an account and server storage that requires login access. For this reason we need to provide a component called an authenticator, even if the app doesn't use accounts like in this case, the authenticator component just contains stub method implementations. To add a stub authenticator component, I've created a class *Authenticator.java* that extends *AbstractAccountAuthenticator.java* class.

In order for the sync adapter framework to access our authenticator, there is a bound Service for that, named *GenericAccountService.java*. The service provides an Android binder object that allows the framework to call our authenticator. To complete this process of authentication, there must be an authenticator metadata file that describes the component, named *authenticator.xml.*

The synchronization adapter encapsulates the code for the tasks that transfers data between the device and a server. Based on triggers in our application the synchronization adapter runs the code in the sync adapter component. All this code is in the method *onPerformSync ( )* method in *NotificationSyncAdapter.java* class. In this method, the app downloads the content from Alimonitor XML feed and passes the content to the XML parser. In order to access code in *onPerformSync ( )* method the application contains a bound Service named *NotificationSyncService.java*. The sync adapter framework requires each sync adapter to have an account type. This type of account is created with a help of *AccountManager's* method named *addAccountExplicitly ().*

The last things to do were to create sync adapter metadata file named *syncadapter.xml* and to declare the adapter in the Manifest with all necessary tags.

After completing all of the previous steps, I've just put all the code for running the sync adapter in the method *onReceive ()* in the *GcmBroadcastReceiver.java* class that I wrote while configuring GCM.

In the end a message is displaying to the user outside of the application normal UI, in the notification drawer of the Android device. This is done with a lot of helper classes like *NotificationManager.java*, *NotificationCompat.Builder.java* and *TaskStackBuilder.java*.

Conclusions and Future work

The use of Android and Google Cloud Technologies provides a very elegant solution for this kind of problem. Most of the project is already done. In future, the 3-rd party server will be created and the user interface will be enhanced. After that, this system will be test extensively with a purpose to be published on the Google Play Store without bugs.