Quality Assurance in Localization

Aparna Sharma\textsuperscript{1} and Manisha Bhatia\textsuperscript{2}

\textsuperscript{1}Computer Science, Banasthali Vidyapith, C-Scheme, Jaipur, INDIA
\textsuperscript{2}Computer Science, Banasthali Vidyapith, C-Scheme, Jaipur, INDIA

Abstract

This paper presents a Quality Assurance model for localized application which has been implemented as an android based application. The model performs the terminology testing (which is to test the terms used in the android application) and Testing results in a number of suggestions of the words/terms that can also be used in the current context. The purpose of this paper is to present the control functionality of the quality tester module named ‘Localizer’ i.e. the flow of control, how the localized application sends the terms as an attachment to the broadcast message which in turn responded and completed by the Localizer. The Localizer in turn accesses the WebAPI for the corresponding terms and fetches the different suggestions. Also the quality tester module runs as a background service which means testing the quality for terms will not affect the current functionality.

\textbf{Keywords:} WebAPI; Sandbox; Localizer; Intent; localize_intent; Broadcast Receiver; OnReceive; AndroidManifest; SQLite; Threads

1. Introduction

In android as every application runs in its own “sandbox”, a running application (cannot communicate directly) must communicate with other installed applications on device either through intents or through broadcast message. \textit{Intents} are to activate a component in the application and \textit{Broadcast messages} are implemented using the BroadcastReceiver class of Android that sends broadcast messages across applications.

The tester module named “Localizer” uses the concept of Broadcastmessages. Any Localized application which wants to profile the terms used in the localized android app specifies its action as “LOCALIZE_INTENT” which is a custom action
that can only be recognized and received by the Localizer quality tester. The tester module analyses the terms attached with the broadcast message, compare them with Web based API to identify its correctness and as a result it shows a number of suggestions regarding the terms that can be used in the same context.

2. Functionality and Control Flow
Testing the quality of a localized android application corresponding to the terms being used is accomplished by functioning of various components at different stages. Figure 1 shows the complete functionality of Localizer as a combined outcome of functions of different components.

![Figure 1. The overall modular functionality of the Localizer](image)

**2.1 Developer**
The Developer can access Localizer in one of two ways:-

**2.2.1 Through sending broadcast message:** Whenever the developer wants to tests its own local android application he/she will send a broadcast message together attached with its own Application name and the String array of the Application terms that need to be tested.

```java
String[] terms="thanks", "thanku", "thanks giving";
Intent intent = new Intent();
PackageManager packageManager = this.getPackageManager();
ApplicationInfo applicationInfo = null;
try {
    applicationInfo = packageManager.getApplicationInfo
    (getPackageName(), 0);
} catch (final NameNotFoundException e) {}
final String title = (String) ((applicationInfo != null) 
& packageManager.getApplicationLabel(applicationInfo) : "???");
intent.putExtra("AppName", title);
intent.putExtra("AppTerms", terms);
getApplicationContext().sendBroadcast(intent);
```
The tester module Localizer implements the class BroadcastReceiver class of android so that it can receive any broadcast message when its specified action in the AndroidManifest file matches with the intent action of the broadcasted message.

```java
public class LocalizeReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Intent service = new Intent(context, SaveService.class);
        service.putExtra("AppName", intent.getStringExtra("AppName"));
        service.putExtra("AppTerms", intent.getStringArrayExtra("AppTerms"));
        context.startService(service);
    }
}
```

A service that works in background is being started as soon as the application receives the broadcasted message. Localizer updates its local SQLite database with the new terms and application received, by implementing the `onReceive` method of `BroadcastReceiver` class.

In the service activity of the android application the local database is updated with the new data received.

```java
protected void onHandleIntent(Intent intent) {
    String [] terms = intent.getStringArrayExtra("AppTerms");
    SQLiteDatabase db = openOrCreateDatabase("LocalDb", MODE_PRIVATE, null);
    db.execSQL("CREATE TABLE IF NOT EXISTS Application (AppName VARCHAR, Appterms VARCHAR);");
    for (int i=0;i<terms.length;i++){
        db.execSQL("INSERT INTO Application VALUES(" + intent.getStringExtra().getCharSequence("AppName") +"," + terms[i] + ")");
    }
    db.close();
}
```

### 2.2.2 Directly executing Localizer as an application:

![Diagram showing the process of directly executing Localizer as an application](image)

**Figure 2.** Application terms corresponding to their name retrieved from local DB
As shown in Figure 2 whenever the developer accesses the Localizer it retrieves the information of Application local database (i.e. the terms and name) and matches with the google WebAPI to find out the corresponding similar terms with minor variations that can also be used in the same context. Google WebAPI responds to the suggestions in the form of XML as shown in Figure 3.

![Figure 3. Suggestions fetched from WebAPI corresponding to the term “thanks”](image)

### 3. Conclusion
Terminology testing of a local android application is of major importance as it evaluates the effectiveness and acceptance of a local product. The implemented quality tester module checks for the “terms” of the local application sent through broadcast message with WebAPI and results in a number of suggestions that can fit in the current context. Also it works as a generic quality model that can test any android based application because the developer itself need to send the profiling terms with the broadcast message. As it checks for each and every term in the application it can also work as a spell checker or to identify the grammatical violations in the application. As shown in Figure 4 the model also works when two or more applications broadcasts messages parallely, because it retrieves and completes the requests through threads.

![Figure 4. Using threads when two application sends broadcast message in parallel](image)

So, when adopted for a local product (like when a developer need to profile his/her app) it may refer directly to the “Localizer” which in turn retrieves the
suggestions as a result of comparing them with WebAPI and returns back the suggestion to the caller of the application.

In addition to all these factors as per the software localization practices one more important aspect is that whether the end localized product follows the cultural conventions of the locale or not, and if it does then till what level it is being satisfied. So, such factors must be separately considered to achieve the higher quality and maximum satisfaction as an end result.

References:
