EXPLOITING INTERACTIVE WORKFLOWS ON ANDROID DEVICES

USAGE RESTRICTED ACCORDING TO LICENSE AGREEMENT.

last update: 29 March 2013. WADE 3.2, JADE 4.3

Authors: Federico Bergenti (Università degli Studi di Parma)
Giovanni Caire (TILAB)

TABLE OF CONTENTS

1 INTRODUCTION

This tutorial describes how to create applications for the Android operating system that fully exploit the possibilities of the WADE *interactivity package*.

The reader is assumed to be familiar with JADE, WADE, Android programming and the Eclipse development environment, which is typically used to create Android applications. If this is not the case, we do recommend reading the following documents before going any further:

- JADE Programming Tutorial (http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf)
- WADE Tutorial (http://jade.tilab.com/wade/doc/tutorial/WADE-Tutorial.pdf)
- Android Tutorial (http://developer.android.com/resources/tutorials/hello-world.html)

This tutorial is organized into the following sections:

- Workflow Driven Interactivity – Quickly describes the fundamental aspects of interactivity as supported by WADE.
- The User Registration Use Case – Presents the running example that this tutorial develops and provides a step-by-step walkthrough to its realization.
- The WADE Interactivity Package – Gives an in-depth presentation of the WADE interactivity package and introduces best practices for its use.
- The Android Interactivity Front-End – Offers a brief introduction to the Android components that can be used to connect users with WADE interactive workflows.

2 WORKFLOW DRIVEN INTERACTIVITY

The workflow metaphor is particularly suited to guide a user in procedures that involve several steps and that may follow different branches depending, e.g., on user inputs or external events. For instance, step *X* of a procedure could consist of asking the user if he or she is sure about something. According to the user response the procedure could then proceed with step *Y* or step *Z*. Representing the procedure as a workflow, the described use case can be easily implemented as depicted in Figure 1.

The WADE interactivity package provides a framework that allows plugging these kinds of workflow-driven procedures into an application taking care, among others, of non-trivial (yet important) issues such as moving backward through previously executed paths and tracing ongoing and completed procedures.

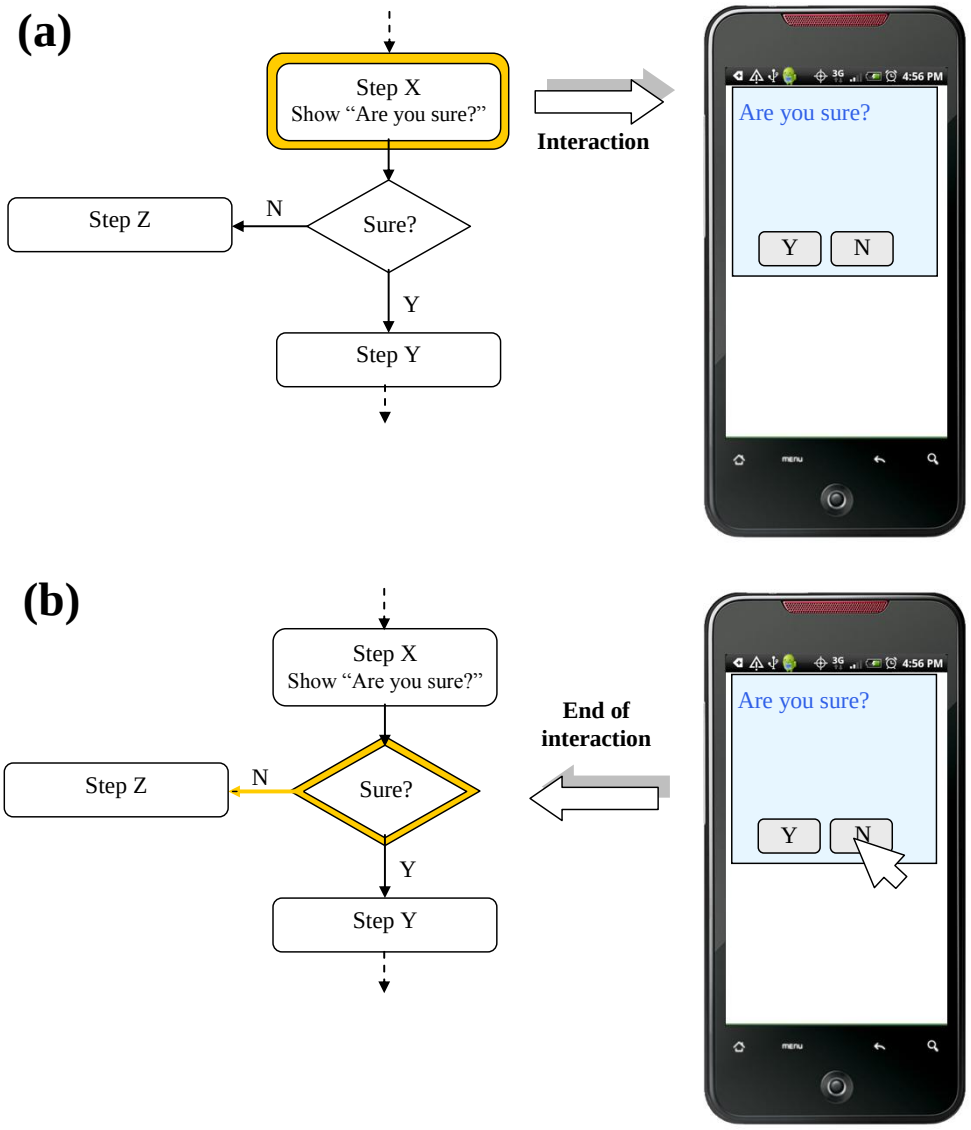

Figure 1. Workflow driven interactivity

3 THE USER REGISTRATION USE CASE

In order to explain how to use the WADE interactivity package to implement workflow driven procedures, this tutorial addresses a minimal use case that is briefly described in this section. The complete sources of the User Registration application that implements the addressed use case, as well as readymade packages that can be used to try it in practice, can be downloaded from the WADE web site at http://jade.tilab.com/wade/resources/UserRegistration.zip.

The User Registration use case that this tutorial develops comprises simple interactions between a user and a back-end service, as follows:

Bob needs to register for an on-line service and the registration procedure requires him to provide basic information such as his first name, his family name and a chosen nickname. Moreover, the registration

procedure also asks Bob for some additional information about his interests. First, the registration procedure asks Bob if he likes sports and, in case, which is his favorite sport (taken from a precompiled list). Then, a similar question is asked about movies: if Bob likes movies, he can select his favorite movie genre from a list. Finally, the registration procedure provides a summary of the answers that Bob provided so that he can confirm them. Notably, the registration procedure allows Bob easily moving backward in the path of answers.

3.1 The User Registration Application architecture

Figure 2 depicts the architecture of the User Registration sample application. As it always happens for a typical WADE-based application accessed by users through Android terminals, two main components are identified:

- The Server part – This is a typical application that runs on top of the WADE platform (see http://jade.tilab.com/wade/doc/tutorial/WADE-Tutorial.pdf).
- The Android Client – This is an Android application that includes the WADE interactivity front-end library for Android (described in next sections) to communicate with the Server part and to manage interaction steps.

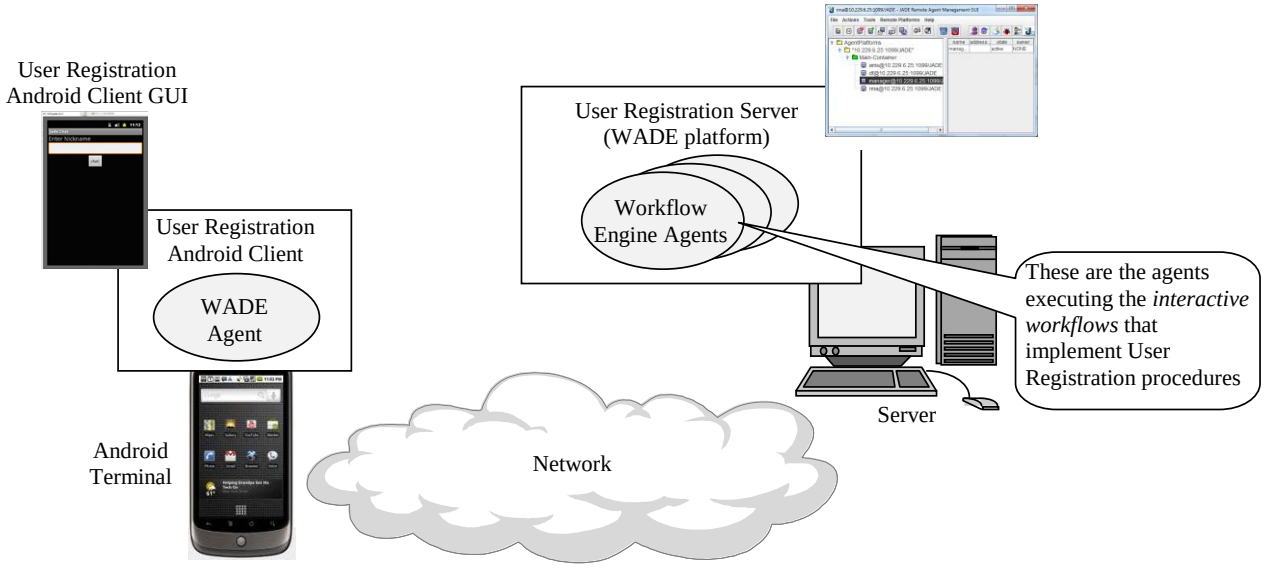

Figure 2. The User Registration Application Architecture

3.2 Downloading the User Registration Application software

The User Registration application can be downloaded from the WADE web site at http://jade.tilab.com/wade/resources/UserRegistration.zip . The package contains three zip files:

- *UserRegistrationServer.zip* – This is the Server part and in this simple scenario there are no application specific classes except for the `UserRegistrationWf` workflow, which concretely implements the user registration interactive procedure.
- *UserRegistrationAndroidClient.zip* – This is the client part and includes both the Android project sources and resources to build the User Registration client application in Eclipse (as will be described in section 5) and the `UserRegistrationAndroidClient.apk` file ready to be installed in the Android emulator or in an Android device.

- *UserRegistrationGWTClient.zip* - This is analogous to the UserRegistrationAndroidClient package, but addresses the case where the user interacts with the UserRegistration application by means of a Web application based on the GWT technology. **It is not addressed in this tutorial**.

3.3 Running the User Registration Application

Once the package is downloaded, follow the steps below to try the User Registration application in practice.

Server Part

0) Unless you already have it, install WADE 3.1 on your PC. This simply means unzipping the WADE distribution package somewhere on your disk. More details can be found in the WADE User Guide available at http://jade.tilab.com/wade/doc/WADE-User-Guide.pdf.

1) Unpack the server part of the application (*UserRegistrationServer.zip*) in the WADE home directory `wadeSuite` thus obtaining the following structure (only relevant entries are shown). The files and directories added in this step are highlighted in bold.

```
wadeSuite/
    |--wade/
           |--bin/
           |--...
           |--examples/
                     |--...
                     |--userRegistration/
                              |--server
                                    |--cfg/
                                    |--src/
                                    |--log/
                                    |--deploy/
                                    |--build.xml
                                    |--...
           |--projects/
                     |--...
                     |--userregistration.properties
```

2) Move to the *bin* directory of the WADE installation (`wadeSuite/wade/bin`) and launch the *WADE boot daemon* and the main container for the User Registration application by typing respectively (if you are on a Linux environment use the *.sh* scripts).

```
startBootdaemon
```

and

```
startMain userregistration
```

Client part

0) To start the client application on the Android emulator it is necessary to setup the Android SDK **(with Android SDK Tools at least version 17)**, create an AVD with **at least an API level 10 (Android 2.3.3)** and run the emulator. Alternatively, if you have a real android device, make sure the appropriate drivers are installed and connect the device to your computer via a data cable (usually USB).
NOTE – If you use a real device, remember that the Android client needs to connect to the platform, therefore make sure that your device is configured so that it is possible to open a TCP connection with the PC where the platform is running over either a LAN or the Internet. Refer to the documentation of your device or to general Android documentation for more details.

1) Unpack the client part of the application (`UserRegistrationAndroidClient.zip`) somewhere on your disk thus obtaining the following directory structure (only relevant entries are shown).

```
UserRegistrationAndroidClient/
    |--...
    |--bin/
           |--UserRegistrationAndroidClient.apk
           |--...
    |--libs/
           |--interactivityAndroidFE.jar
           |--JadeAndroid.jar
           |--wadeInterface.jar
```

2) Install the Android application by moving to the `UserRegistrationAndroidClient/bin` directory and typing the Android SDK commands

```
emulator -avd AVD
```
(to start an emulator named *AVD*)

```
adb install UserRegistrationAndroidClient.apk
```

A new application should appear on your device called *User Registration Client* as shown in Figure 3.

3) Click on the new icon to start the application. The screen depicted in Figure 4 should appear.

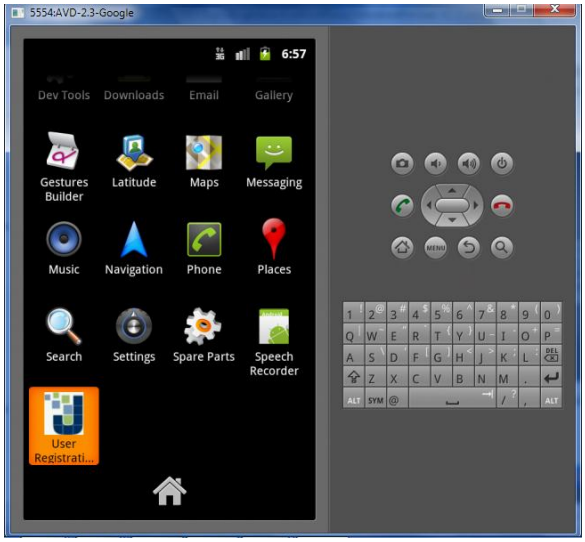

Figure 3. The applications list on the Android emulator

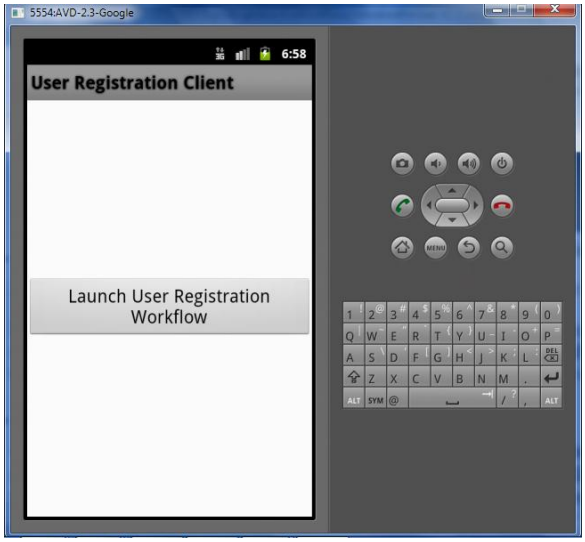

Figure 4. The Android client application initial form.

If you are running the client application on a real Android device connected via a wireless network, you need to setup your application with Main Container parameters (host and port). In order to do so click on the Menu *button and select* Configure WADE. *Then enter the Main Container host and port as depicted in*

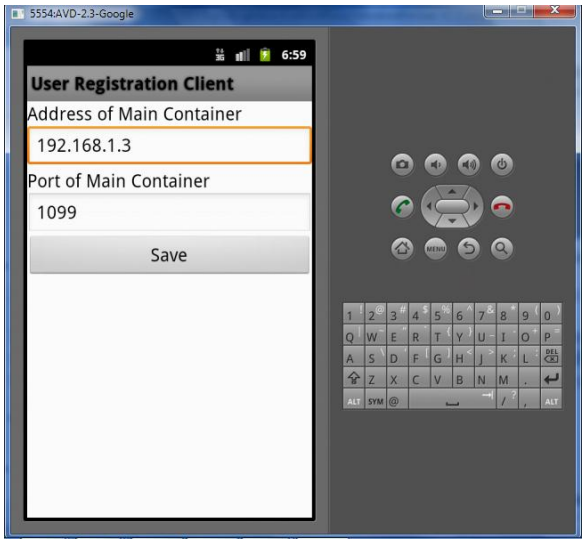

Figure 5.

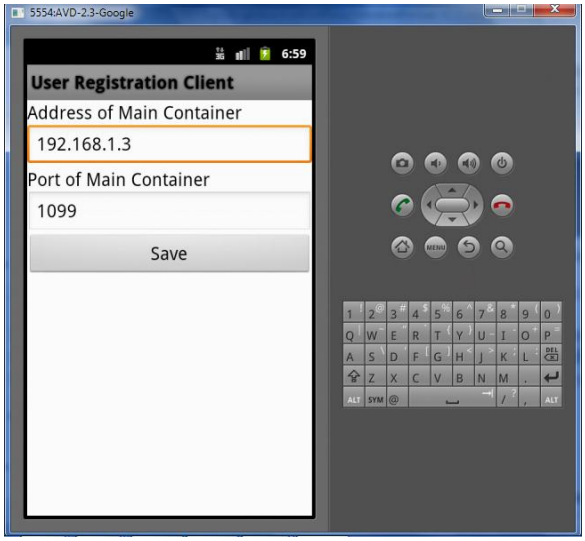

Figure 5. The configuration form.

4) Activate the User Registration interactive procedure by clicking on the *Launch User Registration Workflow* button. This actually starts the execution of the `UserRegistrationWf` workflow that will be described in Section 4. From this point onward, all views appearing on the emulator/device are fully driven by the workflow running in the server part of the application.

The first step of the workflow performs an interaction where the user is requested to insert a nickname, his/her first name and his/her family name as depicted in Figure 6.

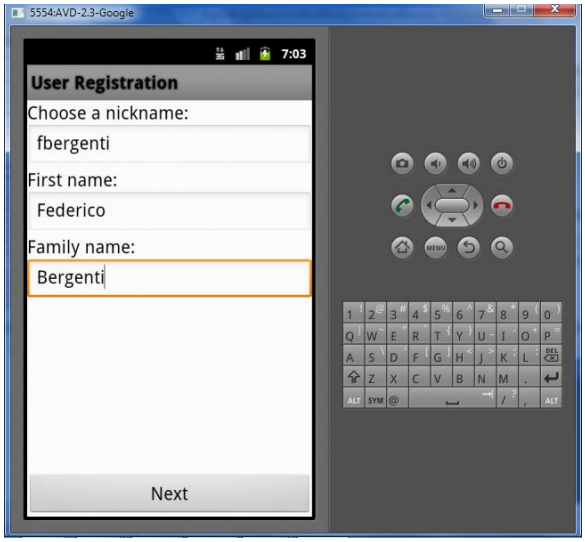

Figure 6. The first interaction of the User Registration workflow

Type in the requested information using the available Android tool and, once completed, click on the *Next* button to make the registration procedure continue.

The rest of the procedure is self-explanatory and it closely follows the description provided at the beginning of this section.

4 THE WADE INTERACTIVITY PACKAGE

In previous section we showed how to use the User Registration application in practice. In this section, and in the following one, we describe how such an application is implemented exploiting the features of the WADE interactivity package.

The WADE interactivity package, i.e., the framework that allows implementing interactive workflow-driven procedures, is composed of three main elements as follows:

- The `InteractiveWorkflowBehaviour` class, i.e., the base class for all workflows implementing procedures that involve user interactions. This class extends `WorkflowBehaviour` (the base class for all WADE workflows) and adds to it the `interact()` method, which actually performs a user interaction step.
- A set of classes that can be used to specify an abstract description of each user interaction step involved in a procedure. This abstract description is fully independent from the visualization technology that is used to create the graphical user interface that the user employs to access the application.
- A module called *WADE Interactivity Front-End* that is specific to the visualization technology that the user adopts to access the application. Such a module is responsible for managing the communication with the agent that is actually executing the interactive workflow and for transforming abstract descriptions received at each interaction steps into concrete GUI elements that are shown to the user.

Currently, WADE provides two implementations of the interactivity front-end: one for the Android operating system that is presented here and another for the ZK open-source framework (see

http://www.zkoss.org) to develop Web application using the AJAX approach, which is outside the scope of this tutorial. Moreover, a third implementation of the WADE interactivity front-end based on the Google Web Toolkit (see https://developers.google.com/web-toolkit) is currently under development.

Figure 7 depicts how the three elements of the WADE interactivity package are used within an Android application to implement interactive procedures.

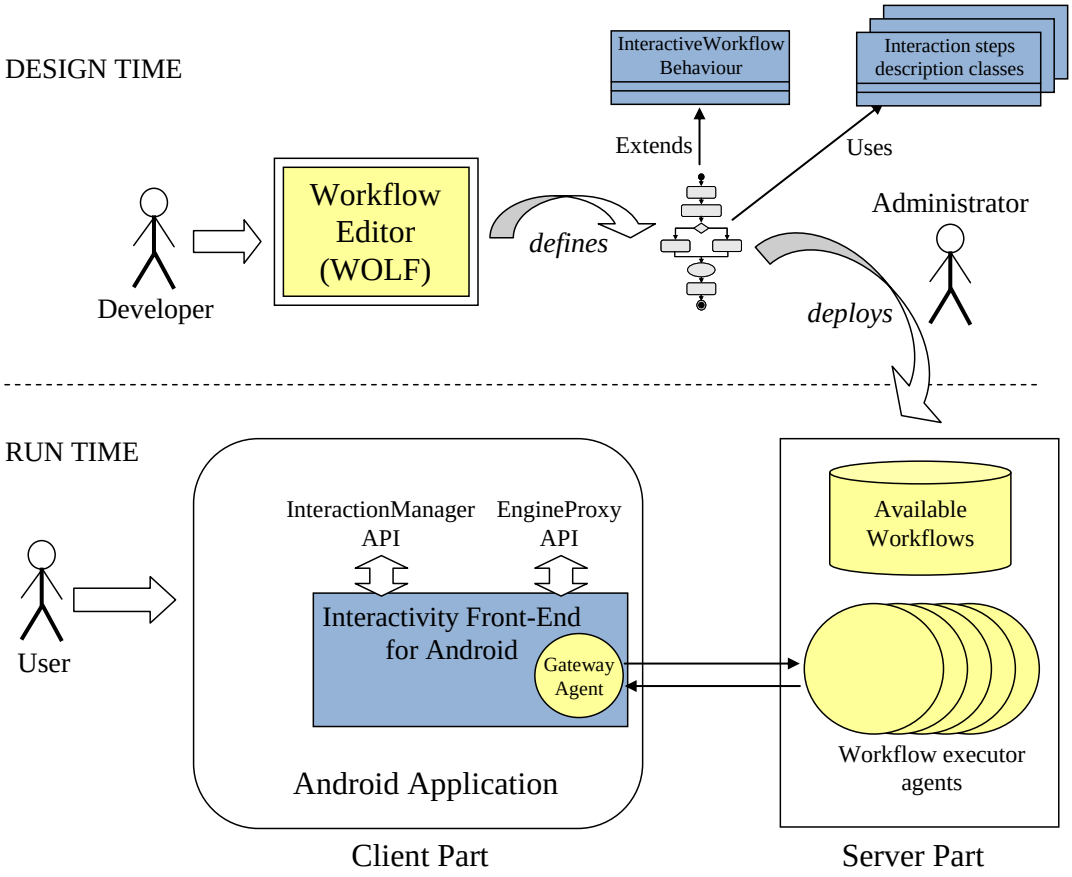

Figure 7. Wade Interactivity package main elements

The Android interactivity front-end mainly provides two APIs that are designed to give access to interactive workflows.

- The *EngineProxy API* includes classes and methods to perform workflow-related tasks like listing available workflows, launching, suspending, recovering and killing workflow executions, and browse ongoing and completed workflow executions.
- The *InteractionManager API* transparently manages all interaction steps involved in the execution of an interactive workflow and provides callback methods that can be redefined to customize most of the aspects of the interaction management mechanism, e.g., executing application specific code just before or just after the visualization of an interaction step.

4.1 Creating interactive workflows

In order to show how to create interactive workflows, this section describes the `UserRegistrationWf` interactive workflow, which implements the user registration procedure described in Section 3.3. **It**

should be noticed–once again–that, though the user exploits the registration procedure on the Android emulator/terminal, the workflow that implements it is executed by an agent that runs in the server part. As a direct consequence, the `UserRegistrationWf` class resides in the *src* directory of the User Registration Server part of the application. If you open such class with the WOLF workflow editor (see the WADE Tutorial available at http://jade.tilab.com/wade/doc/tutorial/WADE-Tutorial.pdf for details) the flow depicted in Figure 8 should appear.

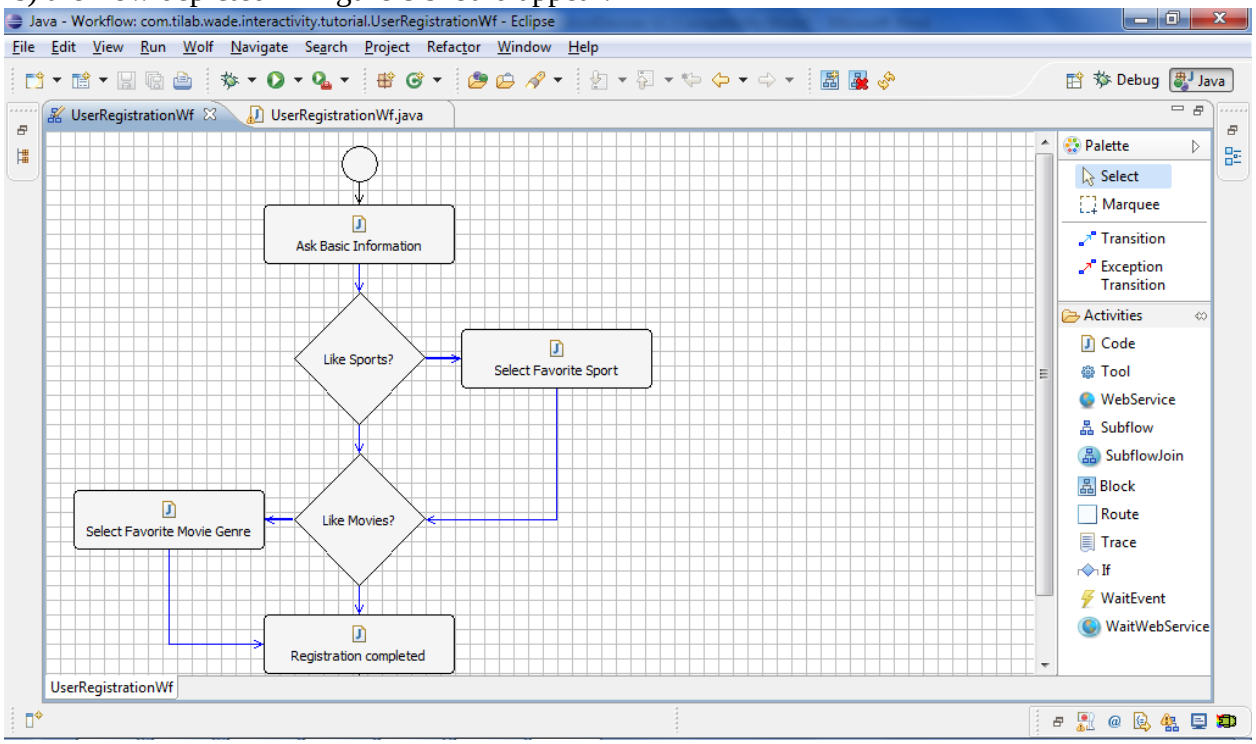

Figure 8. Editing the developed interactive workflow with WOLF visual editor

Switching to the Java editor it is possible to notice that the `UserRegistrationWf` class extends the `InteractiveWorkflowBwhaviour` class rather than the usual `WorkflowBehaviour` class directly. `InteractiveWorkflowBehaviour` is the base class for all workflows that include interactivity steps.

As already mentioned, the main enhancement it provides with respect to the `WorkflowBehaviour` base class is the method `interact()` that actually implements an interaction step. Such a method is supplied with an `Interaction` object that contains the following parts:

- An abstract description of the information to be presented to the user with some abstract requirements on the way information is presented, e.g., by indicating how a set of labels should be aligned on the user screen;
- An abstract description of the information that the user is allowed to return in his or her response;
- An abstract description of the constraints that the user response must meet to be considered valid;
- A list of possible abstract actions that the user is allowed to choose as valid responses.

Upon executing the `interact()` method, the workflow is blocked. In the meanwhile an `Interaction` object is passed to the Interactivity Front-End module that is responsible for presenting to the user the information as well as the available response actions. When the user selects one of the available actions, the `Interaction` object (properly filled with the required inputs, if any, as well as with the user

selection) is sent back to the agent executing the interactive workflow and the `interact()` method returns. The received `Interaction` object is made available to the workflow as return value of the `interact()` method.

The following code, corresponding to the `AskBasicInformation` step of the `UserRegistratioWf` shows an example.

```
protected void executeAskBasicInformation() throws Exception {
      // Prepare interaction
      Interaction interaction = new Interaction("User Registration");
      interaction.setId("AskBasicInformation");

      // Prepare main panel
      ListPanel mainPanel = new ListPanel();
      mainPanel.setOrient(ListPanel.VERTICAL_ORIENTATION);
      interaction.setMainPanel(mainPanel);

      TextBox nickNameTextBox = new TextBox(NICKNAME_ID);
      nickNameTextBox.setLabel("Choose a nickname:");
      nickNameTextBox.setText("");
      nickNameTextBox.addConstraint(
            new RegexConstraint("^\\s*\\w{4,9}\\s*$"));
      mainPanel.addComponent(nickNameTextBox);

      ...

      // Prepare actions
      Action next = new Action(NEXT_ID, NEXT_LABEL);
      interaction.addAction(next);

      // Interact and get user response
      Interaction userResponse = interact(interaction);

      ...
}
```

First, the workflow creates a new `Interaction` object and gives it a title (`"User Registration"`) and a unique ID (`"AskBasicInformation"`). The ID needs to be unique only in the scope of the workflow and the name of the activity is often sufficient. This is because **it is strongly suggested to include a single call to the `interact()` method inside each activity** to enable WADE supporting backward tracing through an execution of a workflow.

Then, the workflow creates a vertical `ListPanel` which is used as the *main panel* of the interaction: every interaction has a main panel which is the container of all visual elements. Such a main panel is then filled with the needed visual elements, i.e., abstract descriptions of visual parts of the user interface. In this case, the workflow uses only `TextBox` objects and creates them with a unique ID (passed in constructor) and a label that the interactivity front-end will present to the user. It is worth noting that the IDs of visual elements must be unique only in the scope of an interaction and they can be freely reused across interactions.

Special visual elements intended to provide information back to the workflow, so called information elements, can use `Constraint` objects to provide in-place validation to user input before the information is sent back to the workflow. In this case, the workflow attaches a `RegexConstraint` object to `TextBox` objects to ensure that fields are properly filled before the interaction can terminate. Generally speaking, visualizers make sure that actions are selectable only if all constraints are met.

Finally, just before invoking the `interact()` method, the actions that the user can choose to provide his/her response are added to the `Interaction` object (a single `next` action in this case) .

Once the user has validly compiled a response and he/she has chosen one of the available response actions, the visualizer returns such a response to the workflow instance in terms of a copy of the original `Interaction` object that is now filled with relevant user input.

Once the `interact()` method returns, it is possible to extract the user inputs from the `Interaction` object returned by the `interact()` method as shown in the code snippet below.

```
nickNameTextBox = (TextBox)userResponse.getComponent(NICKNAME_ID);
nickname = nickNameTextBox.getText().trim();
```

First, IDs are used to retrieve the necessary information elements and then actual user inputs are read by means of specific methods of such information elements (`getText()` in case of a `TextBox`).

Besides `TextBox` elements, the `UserRegistrationWf` shows the usage pattern of other information elements, such as `ExclusiveChoice` as shown in the code snippet below

```
ExclusiveChoice favoriteSportChoice = new ExclusiveChoice(FAVORITE_SPORT_ID);
favoriteSportChoice.setType(ExclusiveChoice.COMBOBOX_TYPE);
favoriteSportChoice.addItem("Soccer");
favoriteSportChoice.addItem("Rugby");
favoriteSportChoice.addItem("Tennis");
favoriteSportChoice.addItem("Baseball");
favoriteSportChoice.addItem("Not in the list");
mainPanel.addComponent(favoriteSportChoice);
```

When the `interact()` method returns the selected choice can be read as below.

```
favoriteSportChoice = (ExclusiveChoice)userResponse
      .getComponent(FAVORITE_SPORT_ID);
favoriteSport = favoriteSportChoice.getSelectedItem();
```

Similarly, the action selected by the user can be read by means of the `getSelectedAction()` method of the `Interaction` class as shown below.

```
Action selectedAction = userResponse.getSelectedAction();
```

4.2 The Interaction Description Framework

Previous section presents an example of implementation of an interaction step. In this section a more comprehensive description of the classes that WADE provides to do that is given. Such classes

collectively form the *Interaction Description Framework* and they are structured in a containment tree as depicted in Figure 9.

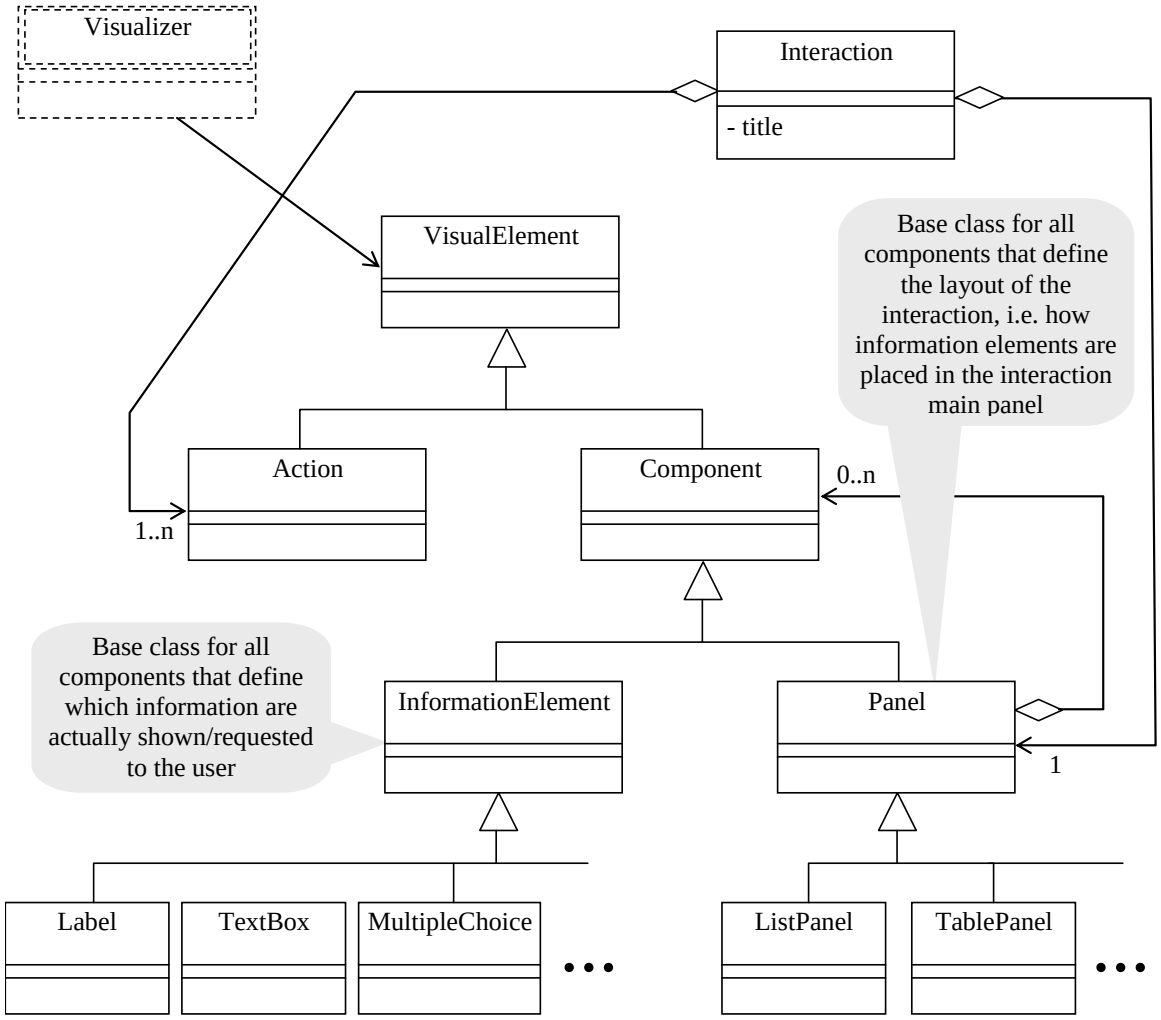

Figure 9. The Interaction Description Framework main classes

The class diagram in Figure 9 sketches the main classes that the developer can use to create an abstract description of an interaction. Such classes are divided into the following major groups:

1. *Information elements*: components that shows and/or require information to the user such as labels, text boxes, multiple and exclusive choices and menus of various types;
2. *Panels*: components that do not manage information on their own, but just contain other components arranged according to given layouts, such as list panels, grid panels and table panels;
3. *Actions*: elements that describe the types of responses the user can select.

According to the MVC pattern, visual elements (no matter if they are information elements, panels or actions) only provide an abstract description (the *model*) of the elements that populate an interaction. The appearance of such elements (the *view*) is under the control of so called *Visualizers* provided by the Interactivity Front-End module.

Finally, each information element can have one or more `Constraint` object attached. Constraints define checks procedures that are applied to user inputs (e.g., checking that a text represents an integer value) and can be seen as a core part the *controller* part in the MVC pattern.

WADE provides a set of general purpose constraints that can be used, e.g., to make sure a mandatory menu has at least an item selected or to warrantee that the text in a text field conforms to a given regular expression.

It is worth mentioning that WADE provides information elements, but not all of them have a suitable visualizer in all Interactivity Front-End modules. This is the case, e.g., for the `Position` and the `Camera` classes that have corresponding visualizers only in the Android Interactivity Front-End module.

5 CREATING AN ANDROID APPLICATION EXPLOITING INTERACTIVE WORKFLOWS

This section practically shows how the Android Interactivity Front-End can be used within an Android application to take advantage of workflow-driven interactive procedures. In order to do that we describe how the application deployed in the previous sections can be compiled and customized.

5.1 Creating the UserRegistration Android Client project in Eclipse

As usual we assume that the **Android Developer Toolkit (ADT) plug-in and the Android SDK with Android Tools at least version 17** have already been installed and that a target AVD supporting **at least platform 2.3.3 (API 10)** has been defined.

NOTE 1 – In previous section you already installed the Android client application on your Android emulator/device. Before continuing in this section, it is strongly suggested that you uninstall it. This is because, since in this section the application will be recompiled in your local environment, very likely the signature of the resulting APK file will be different from that of the `UserRegistrationAndroidClient.apk` included in the distribution package. If this is the case, the installation would fail.

NOTE 2 – Some screenshots may be slightly different depending on the versions of Eclipse and ADT. Those included in this document have been generated using Eclipse Indigo and Android Development Toolkit version 17.

First, you need to create a new Eclipse project for the application. In Eclipse, select "**File** > **New** > **Other...** ". The resulting dialog should have a folder labeled "**Android**" which should contain "**Android Project**" as depicted in Figure 10.

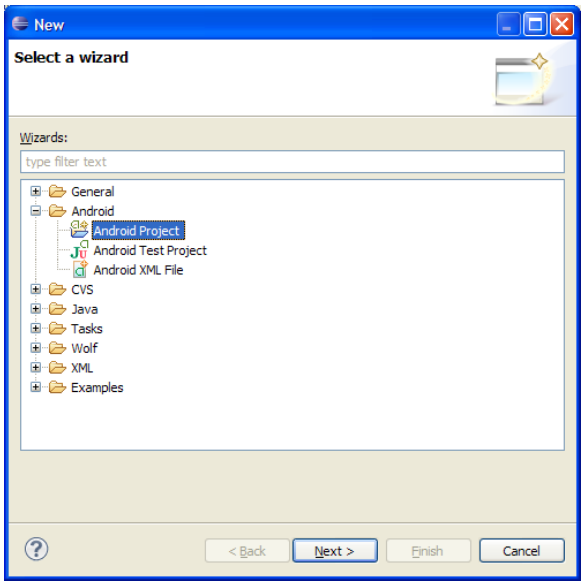

Figure 10. Creating an Android project in Eclipse

Now select "**Android Project**" and click "**Next**".

Type "**UserRegistrationAndroidClient**" as Project Name, and flag (if not yet done) the "**Use default location**" checkbox to create the project in your default workspace. Choose the "**Android 2.3.3**" platform SDK. Enter "**UserRegistrationAndroidClient**" as Application name, "**com.tilab.wade.interactivity.tutorial**" as Package Name and choose 10 (android 2.3.3) as Minimum SDK. Unflag the "**Create Activity**" checkbox (since we already have all the sources available, we don't want Eclipse to create new empty classes for us) and click "**Finish**".

In previous step we created an empty Android client application project. Now we have to fill it with the sources and other resources included in the UserRegistration distribution package. To do that copy the content of the *UserRegistrationAndroidClient* folder in the archive you downloaded (see section 3.3 – Client Part), to the folder *UserRegistrationAndroidClient* created in your Eclipse workspace. Overwrite all existing files.

In Eclipse refresh (press F5) your project three and verify that you get a structure similar to that depicted in the following figure.

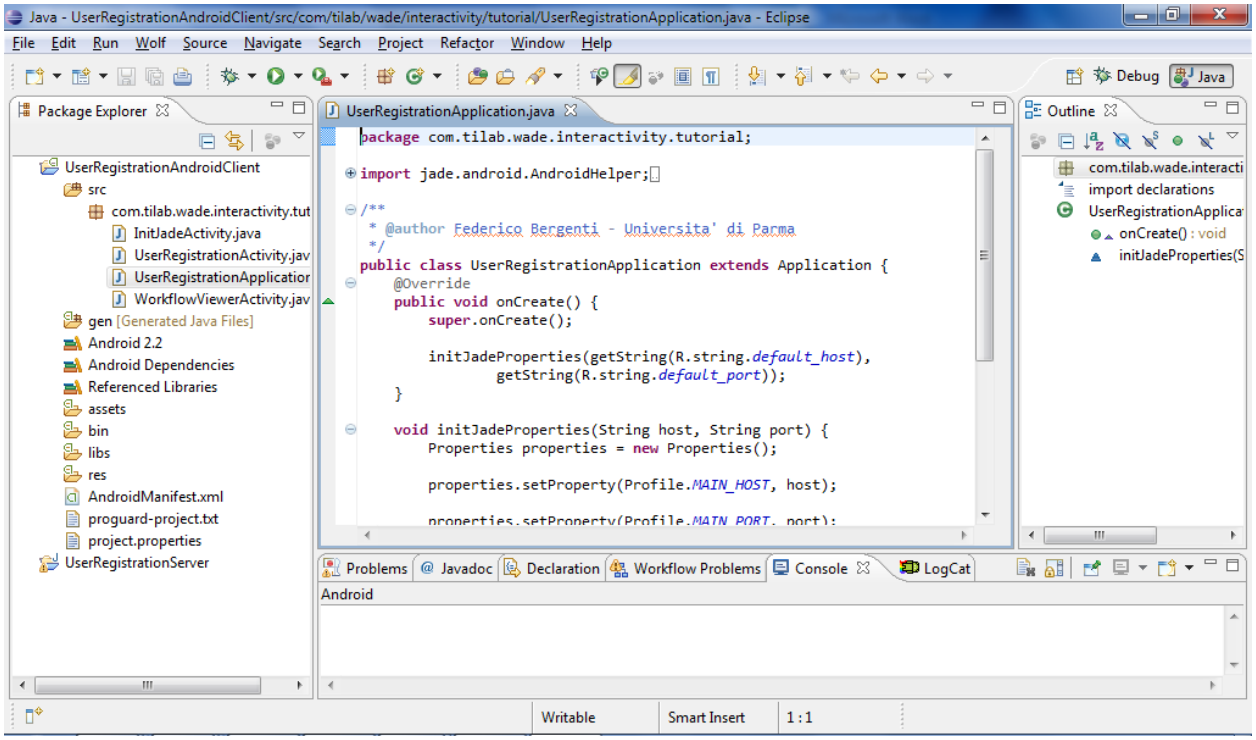

Figure 11. Android client application project structure

Now add the *JadeAndroid.jar*, *interactivityAndroidFE.jar* and the *wadeInterface.jar* libraries to the project build path. To do that open the project "**Properties**" page, select "**Java Build Path**", click on the "**Libraries**" tab and press the "**Add JARs…**" button. In the folder tree that appears go to the *UserRegistrationAndroidClient/libs* directory and select the JAR files. Finally press OK.

- The `interactivityAndroidFE.jar` file contains the actual classes of the WADE Interactivity Front-End module for Android.
- The `wadeInterface.jar` file contains all classes necessary to interact with a WADE application, e.g., to launch and control workflows or detect which workflows are available.
- The `JadeAndroid.jar` file is the JADE version for Android and is necessary to enable agent based communication between the Android application running in the terminal and the WADE based back end running in the network.

These three files are typically downloaded from the standard JADE and WADE distributions available on the JADE/WADE web site. In this case however we already find them in the UserRegistrationAndroidClient application structure.

Now we are ready to compile (if the *Build Automatically* option is selected Eclipse already did that in background) and run the Android client application. First of all, make sure the User Registration Server part is up and running on your PC. If this is not the case, start it as described previously. Then, from the Eclipse Package Explorer, right-click on the *UserRegistrationAndroidClient* project root folder and choose "**Run as** > **Android Application**".

5.2 The UserRegistration Android Client application structure

The `UserRegistrationAndroidClient` is a simple Android application that uses the features of the WADE Android Interactivity Front-End. In this section we briefly describe its structure and we highlight the pieces of code related to interactive workflows management.

The application is composed of 4 classes.

`UserRegistrationActivity` – This is the Android Activity implementing the initial page of the UserRegistration application. This is a very simple activity including a single button (the Launch User Registration Workflow button) with an `OnClickListener` responsible for actually launching the user registration procedure.

`UserRegistrationApplication` – This is the Android Application class and just provides the `initJadeProperties()` method that tells JADE where to find the Main Container.

`InitJadeActivity` – This is the Android Activity that allows the user to set the Main Container host and port. Such information are then passed to the `initJadeProperties()` methods mentioned above.

`WorkflowViewerActivity` – This is the Android Activity actually responsible for managing the interactive registration procedure implemented by the `UserRegistrationWf` workflow. It should be noticed that the mechanism to visualize the interactions triggered by the `UserRegistrationWf` is fully included in its base class `SimpleInteractionActivity` that is part of the WADE Android Interactivity Front-End. The `WorkflowViewerActivity` just redefine the `getErrorMessage()` method to customize the error message to be shown if some constraint is violated.

5.3 Launching an interactive workflow

The code snippet below, taken from the `UserRegistrationActivity`, shows an example of how an interactive workflow can be launched

```
private void launchWorkflow() {
      WorkflowDescriptor workflowDescriptor = new WorkflowDescriptor(
            WORKFLOW_CLASS_NAME, new HashMap<String, Object>());

      EngineProxy.getInstance().launch(
            workflowDescriptor,
            null,
            true,
            new RuntimeCallback<WorkflowController>() {
                  @Override
                  public void onSuccess(WorkflowController result) {
                        String sessionId = result.getSessionId();

                        Intent intent = new Intent(context,
                                    WorkflowViewerActivity.class);

                        intent.putExtra("SESSION_ID", sessionId);

                        startActivity(intent);
                  }
```

```
                @Override
                public void onFailure(final Throwable throwable) {
                      handler.post(new Runnable() {
                            public void run() {
                                  launchWorkflowError(throwable);
                            }
                      });
                }
          });
}
```

First a `WorkflowDescriptor` is created indicating the workflow to be launched. The `UserRegistrationWF` workflow does not have any parameter so no parameter is set to the `WorkflowDescriptor` object.

Then the singleton `EngineProxy` is used to launch the workflow. It should be noticed that, in order to cope with the specificities of the Android environment, the AndroidInteractivity Front-End provides a dedicated version of the standard WADE `EngineProxy` class. For instance, as usually happens for operations that may take some time, this dedicated version of the `EngineProxy` class provides an asynchronous form of the `launch()` method. The second parameter of the `launch()` method allows specifying a listener that will be notified at the end of the workflow. In this case, since there is no specific action that must be taken on workflow completion, we just pass `null`. The third parameter of the `launch()` method is a `boolean` indicating whether or not the workflow must be executed in interactive mode.

Finally, if the workflow launch is successful, we get back a `WorkflowController` object by means of which it is possible to control the interactions involved in the workflow. Actually this low level mechanism is automatically managed by the `SimpleInteractionActivity` class. All we have to do is to start it (as mentioned in previous section we use an extension of the `SimpleInteractionActivity` class) passing the `sessionId` of the launched workflow in the Intent used to start the activity.

The SimpleInteractionActivity supports several customizations to tailor the appearance of the interactions and to handle events such as the beginning and termination of an interaction in an application specific way. Alternatively one can directly use the WorkflowController passed back by the launch() method callback in combination with the InteractionManager class and create a custom activity where interactions are shown. How to do that however is outside the scope of this tutorial.