Generating Android Intent implementation code using EMF and DSL

Ashley Davison-White, ashw@itu.dk
Jorge Y. Castillo Rodríguez, jorgeycrodriguez@gmail.com
Marrtin Rylander, mryl@itu.dk
Agnieszka Majkowska, aga.majkowska@gmail.com

April 24, 2013

Abstract

This paper describes the process of creating a development tool for programmers using the Android platform, specifically in Intent code generation. Intents are abstract ways of describing processes to be performed, and the implementation code can be verbose and tedious to write. While the coding of intents is not difficult to comprehend the specifics of data types and settings for each Intent makes the process error prone. The tool we will be describing in this paper is one that allows for the programmer to insert the generated code of an Intent call at a desired location in an editor in Eclipse. The tool comes in the form of a plug-in to be installed in Eclipse, which is generated based on a Domain Specific Language created using Ecore EMF models and Xtext generated grammar. The models done in the DSL is transformed into models in the Java Abstract Syntax Tree(AST) using the Eclipse Java Development Tools(JDT)before being injected in to Eclipse plug-in view.

1 Introduction

An Android application can contain zero or more activities. When the application has some activities needed to be run, it is important to be able to transit from one activity to another to perform another task either with

or without information from the first activity. In Android, the navigation between activities is possible thanks to Intents.

Intents are asynchronous messages which provide a facility for performing late runtime binding between the code in different applications and allow Android components to request functionality from other components of the Android system. Intents can also be used to signal to the Android system that a certain event has occurred.

Basically an Intent is a passive data structure holding an abstract description of an action to be performed. For instance, lots of Android applications allow their users to share some data with other people e.g. via Twitter application. It is possible to send data to one of this components through an Intent.

1.1 Motivation

Motivation of the project is to make implementing Intent while developing Android applications easier for the programmer. We want to create a user friendly Eclipse plug-in which allows an user to choose an intent from the intents list and to put the generated code of chosen intent with default parameters settings into a proper place in the code. The code can be modified by an user e.g. it is possible to change the values of parameters.

1.2 State of the art

An example of a current technology for dealing with Model2Text transformation is the open source Acceleo project [5], development of which started four years ago. The project aims to ease and speed up the writing of tedious framework specific completion code, and is able to create code generators specific to several frameworks like Android for one.

The Acceleo provides an abstract syntax for generating concrete code and offers editor features such as highlighting, content assistance and error detection.

The code generator works using a template file written in Acceleo syntax that reads in a model and generates simple .java files. The code generator can be further customized by specifying blocks in the template that must be implemented by the user after generation.

Another example of Model2Text transformation this time specifically for Android devices is project called Gplad (Graphical Programming Language

for Android Devices) [6]. It is a domain-specific language that allows to generate code in a blocks-based programming interface, to obtain solutions in Common programming languages.

It is created to make simple programming without a need to code. After creating graphical solution, it is possible to obtain the code in Java or SL thanks to use of Intelligent Agents as JADEAndroid.

1.3 Problem Description

The specific problem we aim to solve surrounds decreasing development errors, and increasing productivity.

The code required to initialize Intents, whilst relatively simple in its basic form, is prone to producing errors because of the customizability and extensive optional parameters and uses. An Intent call in a simple case will use just a few lines of code, whereas a fully extended and defined Intent could, through using additional data and specific parameter settings, grow to be quite verbose.

1.4 Goals

Our goal is to generate the Java code handling intents in Android applications by using a DSL model.

1.5 Methodology

individual steps of action taken, intents work, dsl, code transformations, iterate

We aim to solve the problems surrounding Intent usage by analysing the Domain-specific Language surrounding Intent Filters and later develop this DSL into the Eclipse Modelling Framework. This EMF model will then allow us to create an Eclipse plugin which will generate stubs of code required for Intent use. This automatic code generation will hopefully significantly reduce the possibility of development errors.

We will start the development of our plugin with a small subset of Intents which we will specifically choose for their common usage and variance. We will start by developing our plugin to use Model To Text (M2T) technologies to convert our DSL and models into text that can be injected into the Eclipse editor window. Later in the project, time and scope allowing, we may instead

choose to use Model To Model (M2M) to build our framework and Intent code stubs.

To evaluate the usefulness of our plugin we will run group testing with non-Android developers, and experienced Android or phone developers. We will mark our plugin successfulness around two key areas: reduction in time taken to perform specific tasks, and reduction in coding errors that occur during development. If our plugin is successful, in comparison to development without the plugin, there should be a significant reduction in both of these areas.

2 Background

2.1 Intents

When it comes to Intents there is a distinction between explicit and implicit types of intents [7]. An explicit Intent is primarily used for launching internal activities since it carries specific information as to what class is to be put on the activity stack and executed. The Intent can be explicitly run via the startActivity() method providing only the launching context and the target class to be executed as constructor parameters.

```
@Override
public void onClick(View arg0)
{
    startActivity(new Intent(FirstActivity.this,
        SecondActivity.class));
}
```

The use of implicit Intents complicates the matter a bit, in that the level of abstraction becomes higher. The Intent is no longer directly associated to an activity but rather a generic action that later, as a result of Intent Resolution, will be mapped to a specific activity or service.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(android.content.Intent.EXTRA_TEXT,
        "News for you!");
startActivity(intent);
```

Since more than one activity or service can be eligible for carrying out a generic action, the Manifest-file of the application will define the conditions upon which to choose the correct activity or service given the context. When

an activity is declared in the manifest the use of Intent Filters serve to inform which implicit actions they can handle.

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

Further implicit intent distinctions can be made by declaring categories as sub-elements to filters.

While the explicit intents are fairly trivial the implicit intents can grow to be rather complex not least in the defining of proper filters and categories. If at the end of the intent resolution more than one suitable activity has been found for carrying out the implicit intent the user will be prompted to decide which activity will be allowed to proceed with the action.

2.2 Tools

EMF. Eclipse Modelling Framework (EMF) is a meta-model framework. A self-describing meta-model in EMF is called Ecore. Any EMF model has tree-like structure, where there is a root element and other elements are contained by the root explicitly or through other elements.

XText.

JDT.

3 Realisation

4 Evaluation

4.1 Method

To evaluate our plug-in we wanted to conduct a proper trial to highlight the benefits of using the code generation from the plug-in which carries out the Intent instantiation for you as opposed to having developers write the code themselves.

We based our test primarily on measuring the difference in time spent creating a group of intents to be inserted in a pre-made Android application

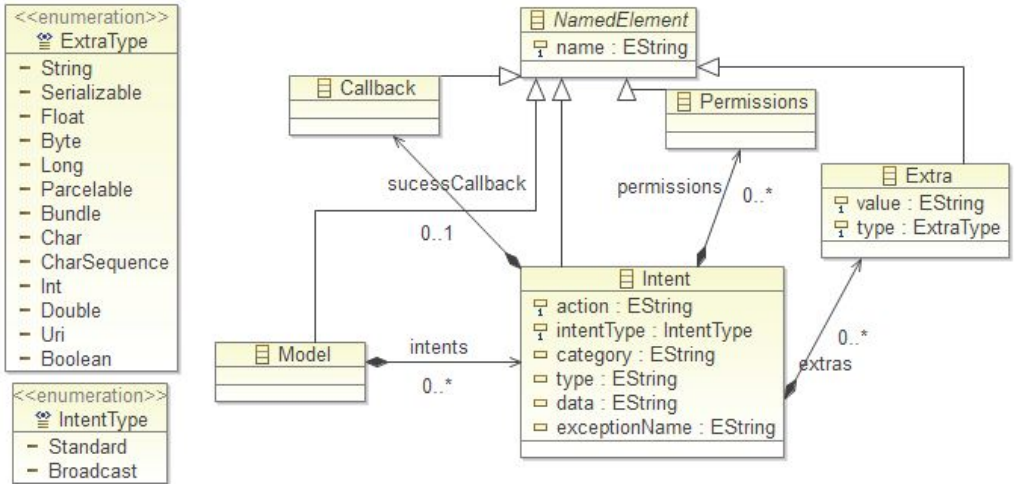

Figure 1: Meta-model of Intent

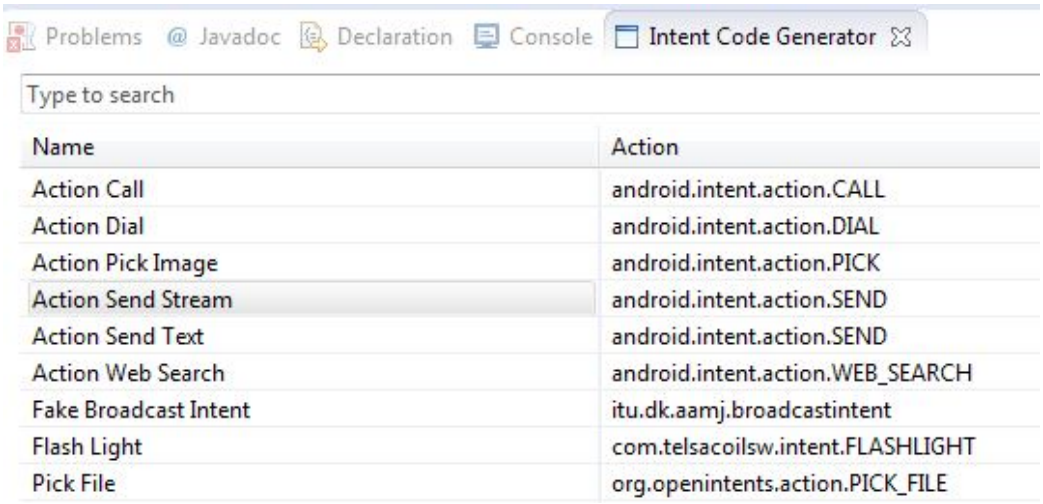

Figure 2: Code generator view

with just one Activity in it. The method stubs are left empty for the trial participants to fill out with the correct code, using either the plug-in or manually. We were briefly contemplating having trial participants building the entire application from scratch but regressed from this idea in that a trial design like that would be introducing to many variables to our test and it would be difficult to dissect the results and clearly establish the effects of

using our plug-in.

The intents required for the participants to create also requires them to insert permissions into the application manifest file, which the plug-in handles for the participants who were instructed to use it.

The participants were split into two groups based on Java programming experience. The least experienced were instructed to use the plug-in and the more experienced were told to do the implementation on their own, with the Google search-engine as their only help.

For the trial we wanted to focus on getting quantifiable data which is why we decided to measure the implementation time, so we can not know for sure if using our plug-in would bring about qualitative improvements as well.

The participants were given a short introduction to the task first, and they were given opportunity to see the application in action on an Android device. After this the subjects were placed in front of a computer and the timer was started. Deciding when the subject was done the task was done by reviewing the code they had written.

4.2 Results

5 Threats to validity

5.1 Internal Validity

5.2 External Validity

6 Use cases

7 Related Work

Model2Text Transformation. There are plenty of articles presenting different approaches to Model2Text transformation, however we can only highlight few of them because of space limits.

Albert, Munoz, Pelechano and Pastor show in the their article [3] a Model2Text transformation where UML association specification are transformed into C# code. For the project purpose it was necessary to extend UML proposal with association relationships and to create an input models by using a conceptual framework. In addition project required to define

the set of rules to generate the C# code. Model2Text transformation was implemented in EMD and MOFScript tools.

Ugaz in his article [2] presents the process of Model2Text transformation using the Epsilon Generation Language (EGL), the DSLs are implemented in MetaDepth. EGL transforms a model created in DSML and formalism into the code. The formalism is the one of Role-Playing Games. The target text is a code in Java for Android application framework. The author focuses on 3 fundamental DSM elements: a domain-specific language, a domain-specific code generator and a domain-specific framework.

Text2Model Transformation. Breslav focused in his paper [4] on creating textual syntax for Domain-Specific Languages (DSL). The main concept is to represent analysis of textual syntax as a sequence of transformations, which is made by using abstract syntax trees (ASTs). the author divided the transformation process into 2 parts: Text2AST which is handled by openArchitectureWave and AST2Model proposed by the author.

FSML. Framework-Specific Modeling Language (FSML) is a special category of Domain-Specific Language (DSL) that is defined on top of an object-oriented application framework, they model abstractions and rules of application programming interfaces (APIs). FSMLs help developers understand, analyze, create, migrate and evolve application code by showing how applications use APIs.

FSMLs are an explicit representation of the domain-specific concepts provided by framework APIs. FSMLs are used for expressing framework-specific models of application code, which describe instances of framework-provided concepts that are implemented in the application code. In an FSML each concept instance is characterized by a configuration of features, which represents implementation steps or choices. FSML concept configuration describes how the framework should be completed in order to create the implementation of the concept (how the concept should be implemented in the code).

Such models may be connected with the application code through a forward (the generation of code from FSMs by successively executing transformations for code pattern addition), a reverse (the automatic retrieval of FSMs from application code by detecting feature instances in the code) mapping enabling round-trip engineering (RTE).

Antkiewicz and Czarnecki wrote an interesting paper concerning FSML [1], where they present the concept of FSMLs with round-trip engineering support. They focus on few challenges related to this topic: knowing how to write framework completion code, viewing the design of the completion code

and the migration of the code to the new framework API versions.

8 Conclusion

9 Future Work

all these things that Torsten had an idea of but we never did, like detecting intents, doing sth with manifest etc.

References

[1] Michał Antkiewicz and Krzysztof Czarnecki, *Framework-Specific Modeling Languages with Round-Trip Engineering*. University of Waterloo, 2006.

[2] Rafael Ugaz, *DSM model-to-text generation: from MeatDepth to Android with EGL*. Antwerp University, 2013.

[3] Manoli Albert, Javier Munoz, Vicente Pelechano and Oscar Pastor, *Model to Text Transformation in Practice: Generating Code from Rich Associations Specifications*. Department of Information Systems and Computation Technical University of Valencia, 2006.

[4] Andrey Breslav *DSL development based on target meta-models. Using AST transformations for automating semantic analysis in a textual DSL framework*. St. Petersburg State Univeristy of Information Technology, Mechanics and Optics, 2006.

[5] Acceleo Home Page *http://www.acceleo.org/pages/home/en*. April 2013.

[6] Gplad Project Page *http://code.google.com/p/gplad/*. April 2013.

[7] Android Developer *http://developer.android.com/reference/android/content/Intent.html*. April 2013.