Transforming Model Oriented Program into Android Source Code Based on Drools Rule Engine

Ei Ei Thu, Nwe Nwe

University of Computer Studies Mandalay, Mandalay, Myanmar
Email: eieithuet@gmail.com, nwenwemdy08@gmail.com

Abstract

Model transformation is one of the prominent features and the rising research area of Model Driven Engineering (MDE). MDE promotes models to primary artifacts that drive the whole development process. This paper presents the model transformation approach for textual model oriented programs Umple (UML Programming Language) to generate android applications (apps). The proposed approach improved the generation of android source code by using Drools transformation rules and introducing new concern in model driven mobile engineering. The major objective of proposed transformation approach intends to address consistency between source and target model and also intends to handle productivity issues in model driven software development. The main results of model transformation approach are Java class for model layer, XML file for view layer and android activity class for controller layer. Results show that proposed approach achieves high consistency between source and target model and also improves model transformation productivity.

Keywords

Model Driven Engineering, Transformation Rule, Model Oriented Program

1. Introduction

The mobile application development industry is increasingly growing up due to the intensive use of applications in mobile devices. Most of these devices run on Android operating system. Due to increasingly intensive use of mobile apps, the development of mobile apps demands additional worries about the short time-to-market and to improve the productivity [1]. Software engineering process aims to help leading an IT project from the perspective of transforming design
Model to code implementation in a specific platform. Indeed, the model driven architecture (MDA) is a new discipline of software engineering that has emerged to ensure the development of productive model [2].

Model transformation is considered as one of the key technologies in MDE. The transformation rules map the input model to output model. These rules can be specified in transformation language such as ATLAS, Acceleo, Drools [3], etc. Moreover, the MDA allows the code generation from platform specific models by means of generators that automatically transform models into source code for chosen platform [4]. An infrastructure for model-driven development has a high potential for accelerating the development of software applications. MDA does not concentrate on technical detail but lifts software development to a higher abstraction level. A high quality MDA infrastructure can considerably reduce the time to market in consequence.

In this paper, we propose a model transformation framework based on drools rule engine. In our approach, UMPLE’ class diagram considered as a primary model artifact that used to generate android source code. UMPLE blurs the distinction between code and modeling, enabling modelers and coders to collaborate on models in a way similar to code-based collaboration. The result of the transformation considered as model layer, view layer, controller layer and persistency layer. Model layer consists of plain old java object (POJO) class, view layer includes XML for android user interfaces, controller layer comprises with android activity classes and persistency layer considered as apps data management. In summary, the proposed approach transforms Umple class diagram to android mobile apps which contain create-retrieve-update-delete (CRUD) operations based on model-view-controller (MVC) architecture.

2. Related Work

Model Driven Mobile Apps Development

The authors [5] proposed a MDD approach for generating android applications from UML class diagram using Acceleo transformation languages. But their approach generates only two screens for the given class diagram. Their code generation approach is in progress and still need to complete for other mobile platforms. In literature [6], the authors proposed a model transformation approach based code generator for state machine diagram. Their approach generates Enterprise Java Bean (EJB) from model using ATLAS transformation language. Although the authors argued that their approach improves the quality and productivity of MDA concepts, they did not described the any measurements for proposed transformation approach. In literature [7], they proposed code generation method from UML message sequence diagram based on Meta Object Facility (MOF). They proposed meta-model for message sequence diagram and six model transformation rules written by Acceleo. They also described that MDE absolutely needs automatic tools and methods to transform platform-independent model into platform-specific model.

The researchers [8] described the model oriented programming as a new pa-
radigm to reduce the ever-present tension between model-centric and code-centric development styles. UMLPE is a model-oriented programming language that supports modeling using a textual notation just like other high-level programming languages. Therefore, Umlpe is a multi-faceted technology allowing users to integrate modeling into software development straightforwardly.

Although there are already some approaches to model-driven development of mobile apps, there is still a gap in model transformation approach for mobile applications. Individual approaches are in moving target. In contrast to our proposed approach, the previous approaches are focused on concrete syntax based process and defined transformation rule specifications in template-based approach. In this paper, our approach will mainly focus on abstract syntax based approach. It traverse and parse source model in visitor based approach instead of defining concrete syntax grammar and specify transformation rules in object pattern matching approach by using Drools rule language [3]. The technical space of model transformations still needs to do 80% to get complete transformations [9].

From the perspective of evaluation, the authors [10] proposed the approach for empirical evaluation of model driven engineering in multiple dimensions. Their empirical studies included qualitative (expert judgments) and quantitative data (metrics) evaluations. They supposed that the productivity and defect detection rate are the effective metrics for measuring automation degree of model driven development process. A cost estimation approach is therefore needed that can incorporate characteristics of MDE that affect economies of scale and effort in application development with the size computation of various artifacts in MDE [11]. Therefore we measure our proposed model driven approach’s automation degree from productivity quality criteria. In order to estimate the code generator effort, we use COCOMOII tool [12].

3. Model Transformation Approach Based on Drools Rule Engine

3.1. Architecture

The main components of model transformation framework are parser, model transformer and code generator. Figure 1 shows the architecture of the proposed model transformation framework. Firstly, it takes Umlpe source code as input file. The parser parses the Umlpe file and constructs the syntax tree model. Syntax tree is a tree representation of the abstract syntactic structure of source

![Figure 1. Model transformation framework architecture.](image-url)
code written in a programming language. The transformer transforms the resulted source model syntax tokens into target model object pattern using Drools mapping rules. And then the code generator generates the android source code according to the MVC architecture.

3.2. Parser and Syntax Tree Model Constructor

The main architecture of parser is composed with visitor-based parser which use for extracting abstract syntax tree (AST) from input model and code generator which generate the respective android source code using extracted AST model of input file [13]. AST traverse and collect the information of source code by using the AST visitor method. Firstly, the AST parser creates the compilation unit (CU) and it accepts the visitor node. The visitor node traverses the source code to extract the package declaration, type declaration, field declaration and variable declaration. The input/output relationship of parser and syntax model constructor is shown in Figure 2.

3.3. Transformer

The transformer is composed with Drools inference engine that is able to scale to a large number of rules and facts. The inference component matches facts and data (syntax tree model). Facts may be modified, removed or added by executing rules or from external sources. The transformer receives the partial pieces of source code (package declaration, type declaration, field declaration and variable declaration) with syntax tree model. This model is transformed using predefined set of mapping rules in Drools Rule engine. The rule engine interprets and executes the mapping rules on the source model and produce target model. The relationships between transformer and Drools rule engine can be seen in Figure 3.

3.4. Code Generator

Code generation can be defined as the technique in which we write program that create another programs. After the object pattern matching process, the transformer fires the rules and generates corresponding model for code generator. The code generator receives the POJO model for model layer, XML model for view layer and android model for controller layer. The generator use the java development tool (JDT-core) to generate POJO class and android class source code. It also uses the JDOM to generate XML user interface file. Figure 4 shows

![Figure 2. Parser component.](image-url)
the internal operation design of code generator. In order to generate java class and android from AST fragment, the proposed generator creates new compilation unit and construct the target language sentence structure by using Java Language Specification (JLS8) application programming interface (API). That API allows generator to create language statements such as Method Declaration, Variable Declaration, Expression Statement declaration, Method Invocation, Super Method Invocation, Field Access, Cast Expression, Class Instance Creation, etc.
3.5. Drools Transformation Rules

A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language. A transformation rule is the smallest entity within a model transformation. A rule contains a source pattern and a target pattern. For each occurrence of the source pattern in the source model, a target pattern is created in the target model.

In our proposed transformation approach, transformation rules are defined on drools rule engine which is a production rule system and enhanced implementation of the Rete algorithm [3]. Drools rules are defined using Java-like language. The main rules in our proposed rule engine are Umple2Model.drl, Umple2View.drl and Umple2Controller.drl. Umple2Model.drl transforms incoming abstract syntax model (ASM) into plain old java object (POJO), Umple2View.drl transforms ASM into android user interface XML file and Umple2Controller.drl transforms ASM into android activity class. Table 1 illustrates the smallest entity of simple variable transformation process. The second row in table shows the transformation rule that use to transform ASM into variable declaration. The third row shows the resulted code for that variable declaration, the fourth row shows the generated user interface in android screen and the final row shows the generated SQL command that used to create database in generated application.

### Table 1. Drools transformation rule sample.

<table>
<thead>
<tr>
<th>Umple</th>
<th>String bookTitle;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Drools Transformation Rule</strong></td>
<td><strong>POJO (Model Layer)</strong></td>
</tr>
<tr>
<td>Rule “VariableDeclaration”</td>
<td>String bookTitle;</td>
</tr>
<tr>
<td><strong>Dialect</strong> “java”</td>
<td></td>
</tr>
<tr>
<td><strong>when</strong> $st : SyntaxTree(status==SyntaxTree.VAR_DECLARE)</td>
<td></td>
</tr>
<tr>
<td><strong>then</strong> TypeDeclaration type=AST2Android.Variable_Decl($st.getType());</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CompilationUnit cu=$st.getCu();</td>
</tr>
<tr>
<td></td>
<td>$st.setStatus(SyntaxTree.ACTIVITY_CREATE);</td>
</tr>
<tr>
<td></td>
<td>$st.setType(type);</td>
</tr>
<tr>
<td></td>
<td>$st.setCu(cu);</td>
</tr>
<tr>
<td></td>
<td><strong>update</strong>($st);</td>
</tr>
<tr>
<td></td>
<td><strong>end</strong></td>
</tr>
</tbody>
</table>

**Generated Code**

```java
String bookTitle;
_publicvoid setBookTitle(String bt){
  bookTitle=bt;
}
public String getBookTitle(){
  return bookTitle;
}
```

**Generated UI**

![Book Title](Image)

**Generated Persistency Layer**

```sql
db.execSQL("CREATE TABLE IF NOT EXISTS Books Table (BooksID Integer PRIMARY KEY AUTOINCREMENT, book Title String);")
```
4. Evaluation and Analysis

4.1. Parser Analysis

To illustrate the ability of proposed parser, we have conducted the parser statistics with time efficiency and compare with Umple code analyzer [14]. As a case study of transformation process, we choose a Library Management System as Umple input source file. It contains number of 6 classes with 19 attributes. Figure 5 shows the implemented screen of proposed parser.

In order to measure the proposed parser performance, we extract the source file metrics from the input Umple file. The measurable metrics include number of classes, number of lines of code, number of variables and its data type. And then, we also conducted the proposed parser precision and recall using the following equations. Figure 6 depicts the parser analysis for Library Management System case study.

\[
\text{Precision} = \frac{\text{No. correct constituents identified by parser}}{\text{Total no. constituents identified by parser}}
\]

\[
\text{Recall} = \frac{\text{No. correct constituents identified by parser}}{\text{Total no. constituents identified by Code Analyzer}}
\]

At this stage, we measured the proposed parser performance because of the parser accuracy can improve the later transformation and code generation processes. And also, this measurement can validate that whether the target model’ attributes are consistent with source model’ attributes or not. Maintaining consistency between documentation and the corresponding code is challenging task. Table 2 shows the proposed parser gets the average 100% of preci-
Figure 6. Parser analysis.

<table>
<thead>
<tr>
<th>Project Name–Library Management System</th>
<th>Found</th>
<th>Expected</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Classes</td>
<td>6</td>
<td>6</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>No. Attributes</td>
<td>19</td>
<td>22</td>
<td>100%</td>
<td>86%</td>
</tr>
<tr>
<td>No. LOC</td>
<td>42</td>
<td>42</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

sion and 95.33% of recall rate. Therefore, we have the high consistent between source documents and target code.

4.2. Code Generator Analysis

For practical use in model-driven engineering, the quality of model transformations is a key issue. In this paper, we also conducted the quantitative measurement from the perspective of productivity criteria. Because of the productivity criteria is the effective metrics to ensure the automation degree of model transformation. The generated source code result for case study is show in Table 3. Figure 7 shows the implemented screen of code generator.

In order to calculate the estimated value in time and effort, we use the COCOMOII tool [12]. For this measurement, we set 616 SLOC and set the product cost parameters settings with the default nominal value. The tool showed a needed effort of 2 Person-months, a required 4months’ time frame for 616 SLOC. The proposed transformation approach can generate 616 SLOC within 2 seconds. Therefore, by using our transformation approach we could reduce time and effort in development process by 4 months and effort of 2 Person-months.
4.3. Results and Discussion

In order to ensure the quality of proposed model driven transformation approach, we have conducted relevant measurement with step by step. The proposed transformation approach is developed and successfully tested over 20 UMLE model projects. Firstly, we have measured the proposed parser efficiency with precision and recall. The results show that the parser precision rate from 99% to 100% and also the recall rate from 90% to 100%. Therefore, our proposed parser achieves high consistency between source file and parsed syntax. Table 4 shows the results analysis for parser efficiency, code generation efficiency and cost estimation value for each project. Secondly, we have conducted the statistical analysis for code generation process with number of lines of code metrics, number of files and code generation time for each phase. The main purpose of our proposed approach is to achieve high productivity quality. According to the cost estimation tool results, our approach can significantly reduce the time and developer effort. Therefore the proposed approach also reaches the satisfactory in productivity measurement.
Table 4. Results analysis.

<table>
<thead>
<tr>
<th>No</th>
<th>Project</th>
<th>Parser Efficiency</th>
<th>Code Generation Efficiency</th>
<th>COCOMOII Estimated Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>SLOC</td>
</tr>
<tr>
<td>1</td>
<td>Online Golf Store System</td>
<td>100%</td>
<td>90.67%</td>
<td>797</td>
</tr>
<tr>
<td>2</td>
<td>Online Shopping System</td>
<td>100%</td>
<td>98.34%</td>
<td>853</td>
</tr>
<tr>
<td>3</td>
<td>Banking Structure System</td>
<td>99.27%</td>
<td>100%</td>
<td>630</td>
</tr>
<tr>
<td>4</td>
<td>Patient Information System</td>
<td>100%</td>
<td>97.85%</td>
<td>654</td>
</tr>
<tr>
<td>5</td>
<td>Order &amp; Payment System</td>
<td>100%</td>
<td>97.77%</td>
<td>611</td>
</tr>
</tbody>
</table>

5. Conclusions

Model transformation can be used for different tasks throughout the development process for manipulating models. The field of model transformation is an active research field and it is necessary to exploit the model transformation approach for the development of mobile applications. In this paper, an approach capable of generating MVC-based android source code from Umple class diagram is developed. This approach involves Drools model transformation rules that can address the technical space of MDE. Drools transformation rules make the proposed approach efficient, simple and faster in implementation. The parser analysis results show that the parsed source code is consistent with input Umple file. The evaluation results with COCOMOII estimator show that the transformation approach significantly reduces the effort and improves the productivity. Therefore, the proposed transformation approach is suitable for short time-to-market software development environment.

In this paper, we address the proposed parser consistency and productivity issues in model driven transformation process. As a future work, we plan to address the internal transformation rule quality measurement. The measurement of internal transformation rule quality is still challenging task in model driven engineering process.

References


---

Submit or recommend next manuscript to SCIRP and we will provide best service for you:

Accepting pre-submission inquiries through Email, Facebook, LinkedIn, Twitter, etc.
A wide selection of journals (inclusive of 9 subjects, more than 200 journals)
Providing 24-hour high-quality service
User-friendly online submission system
Fair and swift peer-review system
Efficient typesetting and proofreading procedure
Display of the result of downloads and visits, as well as the number of cited articles
Maximum dissemination of your research work

Submit your manuscript at: [http://papersubmission.scirp.org/](http://papersubmission.scirp.org/)
Or contact [jcc@scirp.org](mailto:jcc@scirp.org)