Journal of Computer and Communications, 2017, 5, 49-59
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

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

How to cite this paper: Thu, E.E. and New, N. (2017) Transforming Model Oriented Program into Android Source Code Based on Drools Rule Engine. *Journal of Computer and Communications*, **5**, 49-59.
https://doi.org/10.4236/jcc.2017.53006

Received: February 17, 2017
Accepted: March 10, 2017
Published: March 13, 2017

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 paradigm to reduce the ever-present tension between model-centric and code-centric development styles. UMPLE is a model-oriented programming language that supports modeling using a textual notation just like other high-level programming languages. Therefore, Umple 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 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 Umple source code as input file. The parser parses the Umple file and constructs the syntax tree model. Syntax tree is a tree representation of the abstract syntactic structure of source

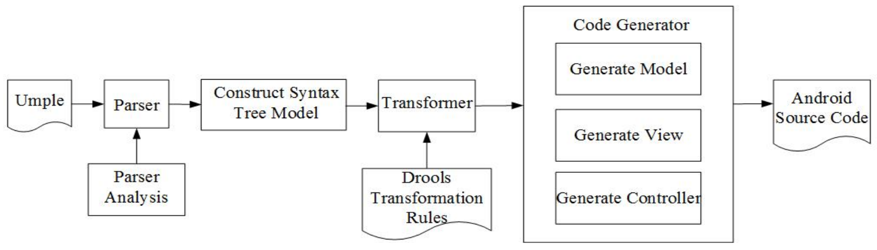


Figure 1. Model transformation framework architecture.

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

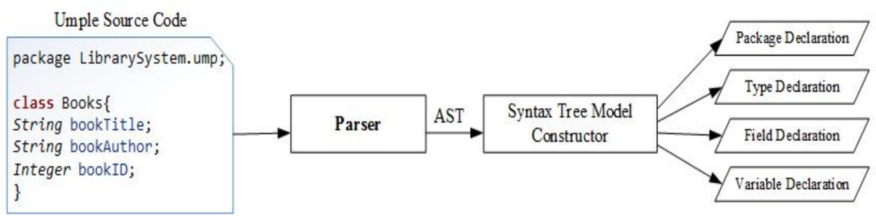
The main architecture of parser is composed with visitor-based parser which use for extracting abstract syntax tree (AST) from input model and code generator which generate the respective android source code using extracted AST model of input file [13]. AST traverse and collect the information of source 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

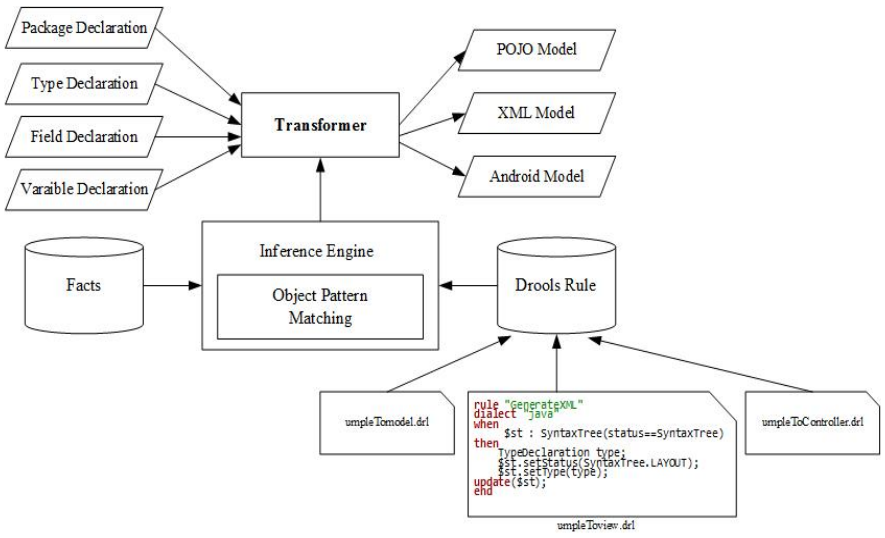
Code generation can be defined as the technique in which we write program that create another programs. After the object pattern matching process, the transformer fires the rules and generates corresponding model for code generator. The code generator receives the POJO model for model layer, XML model 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



```
package LibrarySystem.ump;

class Books{
String bookTitle;
String bookAuthor;
Integer bookID;
}
```

Figure 2. Parser component.



```
rule "GenerateXML"
dialect "java"
when
     $st : SyntaxTree(status==SyntaxTree)
then
    TypeDeclaration type;
    $st.setStatus(SyntaxTree.LAYOUT);
    $st.setType(type);
update($st);
end
```

Figure 3. Transformer component.

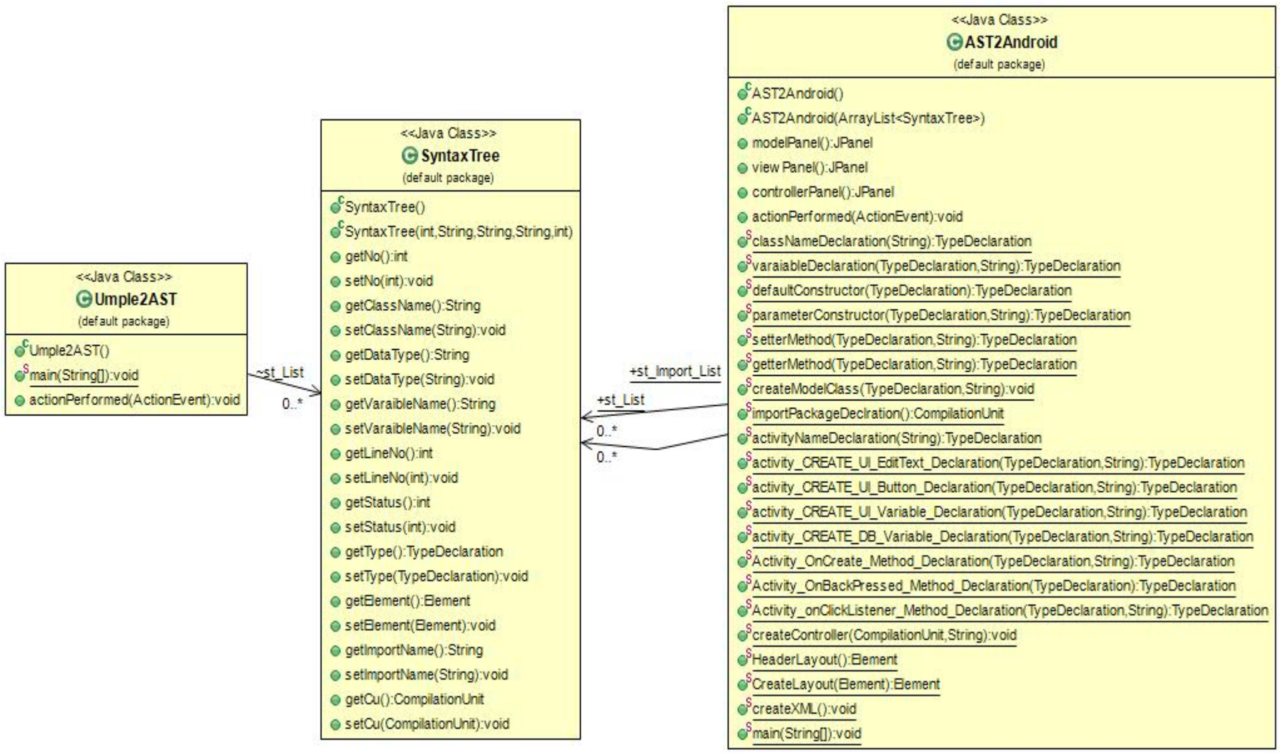

<<Java Class>>
Umple2AST
(default package)
Umple2AST()
main(String[]):void
actionPerformed(ActionEvent):void
~st_List
0..*
<<Java Class>>
SyntaxTree
(default package)
SyntaxTree()
SyntaxTree(int,String,String,String,int)
getNo():int
setNo(int):void
getClassName():String
setClassName(String):void
getDataType():String
setDataType(String):void
getVaraibleName():String
setVaraibleName(String):void
getLineNo():int
setLineNo(int):void
getStatus():int
setStatus(int):void
getType():TypeDeclaration
setType(TypeDeclaration):void
getElement():Element
setElement(Element):void
getImportName():String
setImportName(String):void
getCu():CompilationUnit
setCu(CompilationUnit):void
+st_Import_List
+st_List
0..*
0..*
<<Java Class>>
AST2Android
(default package)
AST2Android()
AST2Android(ArrayList<SyntaxTree>)
modelPanel():JPanel
view Panel():JPanel
controllerPanel():JPanel
actionPerformed(ActionEvent):void
classNameDeclaration(String):TypeDeclaration
varaiableDeclaration(TypeDeclaration,String):TypeDeclaration
defaultConstructor(TypeDeclaration):TypeDeclaration
parameterConstructor(TypeDeclaration,String):TypeDeclaration
setterMethod(TypeDeclaration,String):TypeDeclaration
getterMethod(TypeDeclaration,String):TypeDeclaration
createModelClass(TypeDeclaration,String):void
importPackageDeclration():CompilationUnit
activityNameDeclaration(String):TypeDeclaration
activity_CREATE_UI_EditText_Declaration(TypeDeclaration,String):TypeDeclaration
activity_CREATE_UI_Button_Declaration(TypeDeclaration,String):TypeDeclaration
activity_CREATE_UI_Variable_Declaration(TypeDeclaration,String):TypeDeclaration
activity_CREATE_DB_Variable_Declaration(TypeDeclaration,String):TypeDeclaration
Activity_OnCreate_Method_Declaration(TypeDeclaration,String):TypeDeclaration
Activity_OnBackPressed_Method_Declaration(TypeDeclaration):TypeDeclaration
Activity_onClickListener_Method_Declaration(TypeDeclaration,String):TypeDeclaration
createController(CompilationUnit,String):void
HeaderLayout():Element
CreateLayout(Element):Element
createXML():void
main(String[]):void

Figure 4. Code generator structure.

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.

Umple	String bookTitle;		
Drools Transformation Rule	**Rule** "VariableDeclaration" **Dialect** "java" **when** $st : SyntaxTree(status==SyntaxTree.VAR_DECLARE) **then** TypeDeclaration type=AST2Android.Variable_Decl($st.getType()); CompilationUnit cu=$st.getCu(); $st.setStatus(SyntaxTree.ACTIVITY_CREATE); $st.setType(type); $st.setCu(cu); **update**($st); **end**		
	POJO (Model Layer)	XML (View Layer)	Android (Controller Layer)

Generated Code	String bookTitle;
	PublicvoidsetBookTitle(String bt){
	bookTitle=bt;}
	public String getBookTitle(){
	return bookTitle;}
	}
		<EditText android:id=
		"@+id/ txtbookTitle"
		android:layout_height=
		"wrap_content"
		android:layout_width=
		"wrap_content"/>
			private EditText txtbookTitle;
			private String bookTitle;
			txtbook-
			Title=(EditText)findViewById(R.id.txtbook
			Title);

Generated UI			
Generated Persistency Layer	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

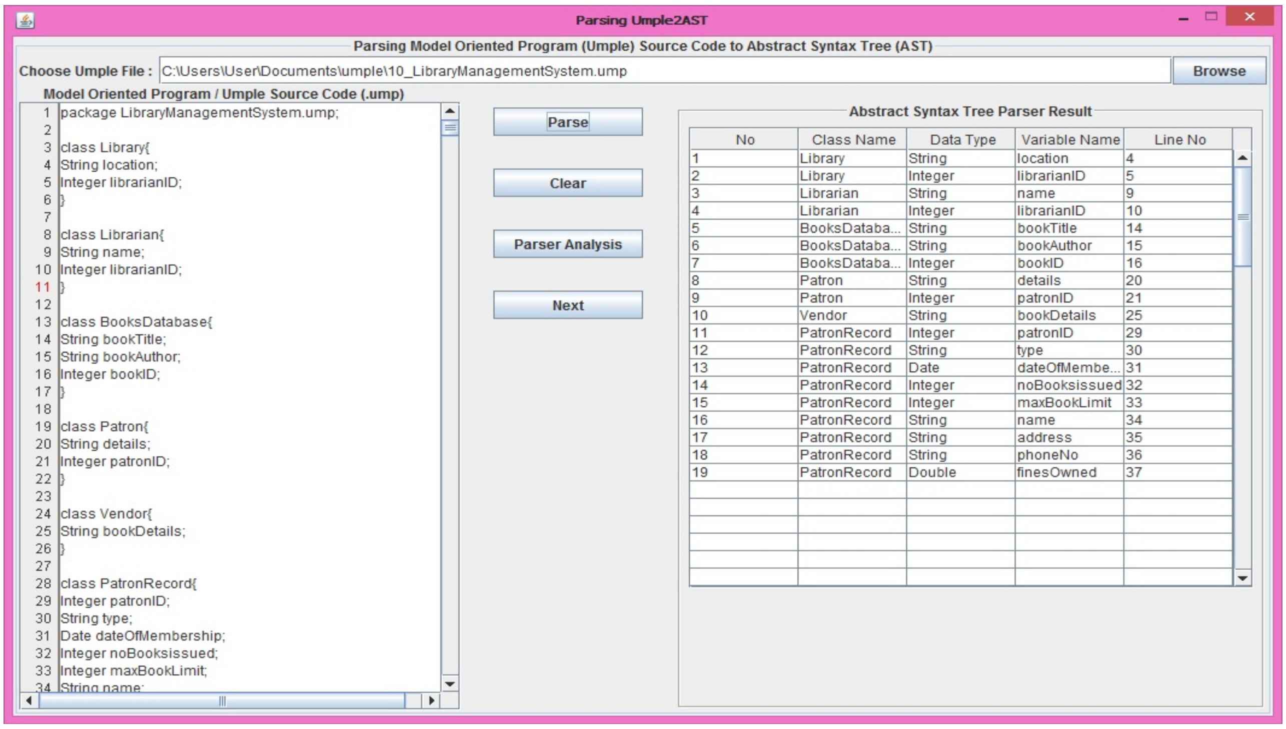
To illustrate the ability of proposed parser, we have conducted the parser statistics with time efficiency and compare with Umple code analyzer [14]. As a case study of transformation process, we choose a Library Management System as Umple input source file. It contains number of 6 classes with 19 attributes.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

$$\text{Precision} = \frac{\text{No.correct consituents identified by parser}}{\text{Total no.consituents identified by parser}} \tag{1}$$

$$\text{Recall} = \frac{\text{No.correct consituents identified by parser}}{\text{Total no. consituents identified by Code Analyzer}} \tag{2}$$

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-

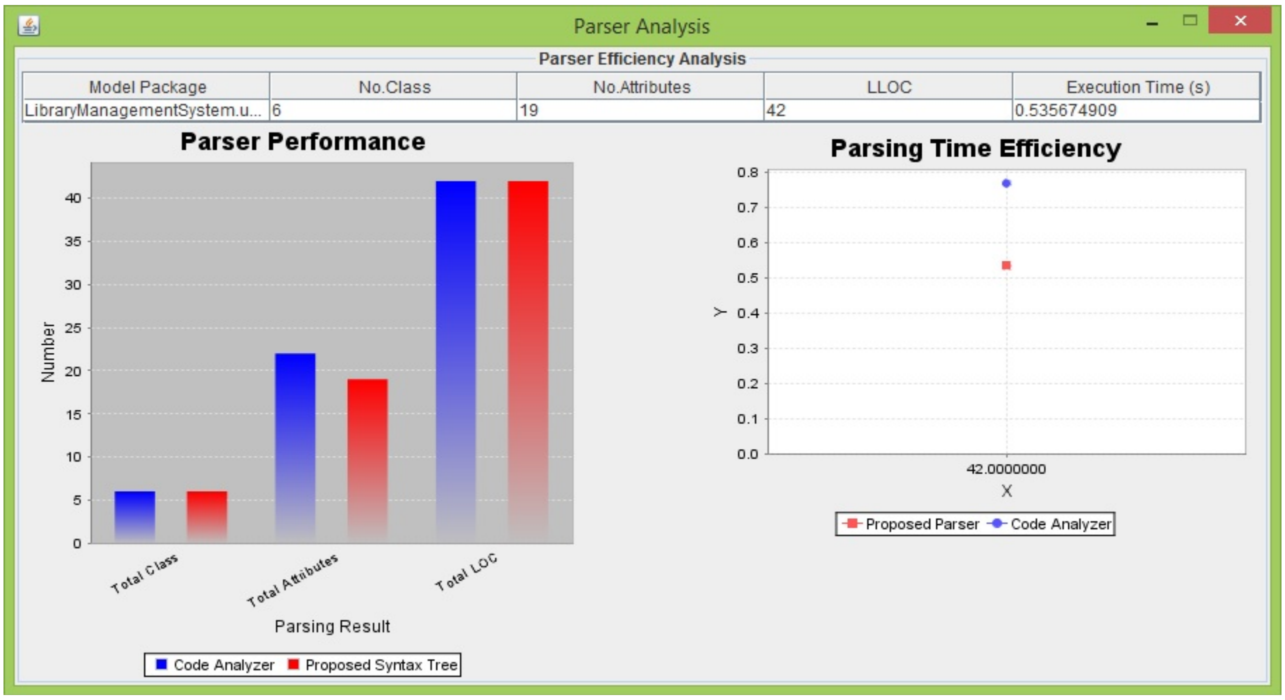

Parsing Umple2AST
Parsing Model Oriented Program (Umple) Source Code to Abstract Syntax Tree (AST)
Choose Umple File :
Browse
Model Oriented Program / Umple Source Code (.ump)
Parse
Clear
Parser Analysis
Next
Abstract Syntax Tree Parser Result

C:\Users\User\Documents\umple\10_LibraryManagementSystem.ump

```
 1 package LibraryManagementSystem.ump;
 2
 3 class Library{
 4 String location;
 5 Integer librarianID;
 6 }
 7
 8 class Librarian{
 9 String name;
10 Integer librarianID;
11 }
12
13 class BooksDatabase{
14 String bookTitle;
15 String bookAuthor;
16 Integer bookID;
17 }
18
19 class Patron{
20 String details;
21 Integer patronID;
22 }
23
24 class Vendor{
25 String bookDetails;
26 }
27
28 class PatronRecord{
29 Integer patronID;
30 String type;
31 Date dateOfMembership;
32 Integer noBooksissued;
33 Integer maxBookLimit;
34 String name;
```

No	Class Name	Data Type	Variable Name	Line No
1	Library	String	location	4
2	Library	Integer	librarianID	5
3	Librarian	String	name	9
4	Librarian	Integer	librarianID	10
5	BooksDataba...	String	bookTitle	14
6	BooksDataba...	String	bookAuthor	15
7	BooksDataba...	Integer	bookID	16
8	Patron	String	details	20
9	Patron	Integer	patronID	21
10	Vendor	String	bookDetails	25
11	PatronRecord	Integer	patronID	29
12	PatronRecord	String	type	30
13	PatronRecord	Date	dateOfMembe...	31
14	PatronRecord	Integer	noBooksissued	32
15	PatronRecord	Integer	maxBookLimit	33
16	PatronRecord	String	name	34
17	PatronRecord	String	address	35
18	PatronRecord	String	phoneNo	36
19	PatronRecord	Double	finesOwned	37

Figure 5. Umple to AST parser.


Parser Analysis

Parser Efficiency Analysis

Model Package	No.Class	No.Attributes	LLOC	Execution Time (s)
LibraryManagementSystem.u...	6	19	42	0.535674909

Figure 6. Parser analysis.

Table 2. Parser analysis result.

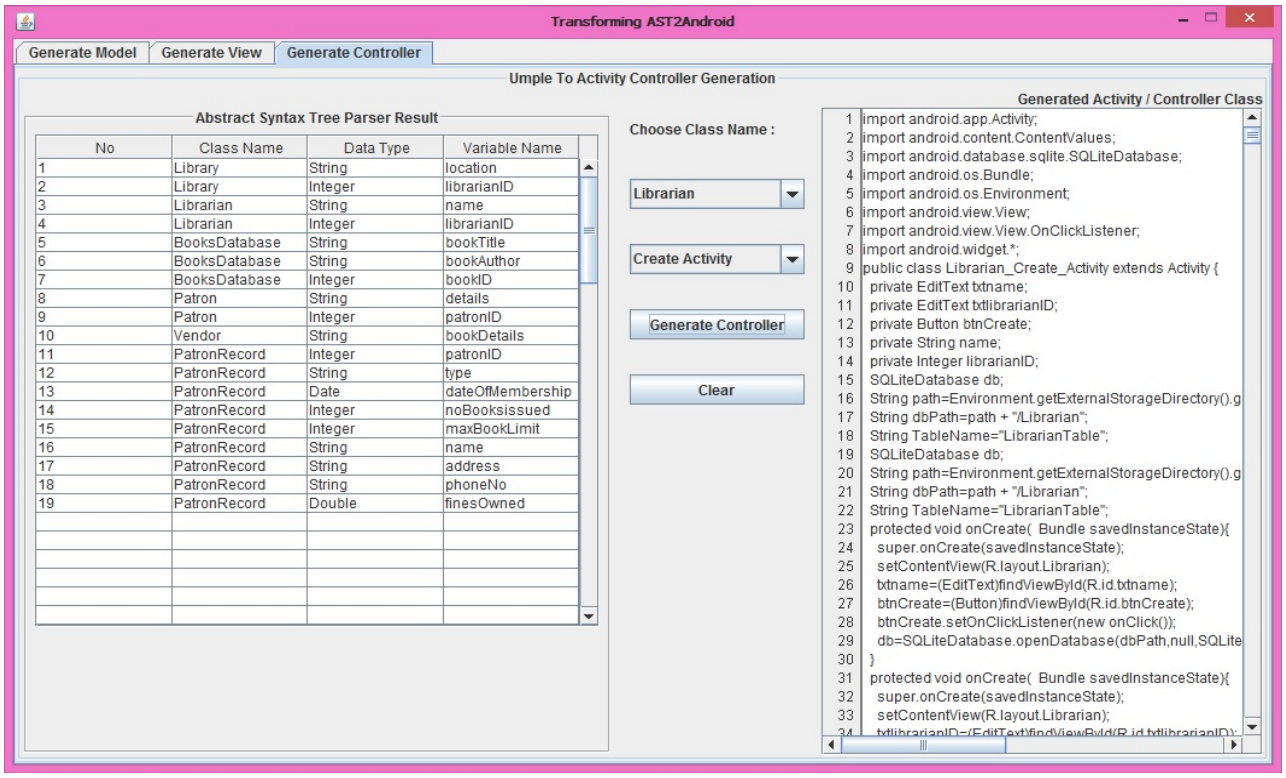
Project Name–Library Management System				
	Found	Expected	Precision	Recall
No. Classes	6	6	100%	100%
No. Attributes	19	22	100%	86%
No. LOC	42	42	100%	100%

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.



No	Class Name	Data Type	Variable Name
1	Library	String	location
2	Library	Integer	librarianID
3	Librarian	String	name
4	Librarian	Integer	librarianID
5	BooksDatabase	String	bookTitle
6	BooksDatabase	String	bookAuthor
7	BooksDatabase	Integer	bookID
8	Patron	String	details
9	Patron	Integer	patronID
10	Vendor	String	bookDetails
11	PatronRecord	Integer	patronID
12	PatronRecord	String	type
13	PatronRecord	Date	dateOfMembership
14	PatronRecord	Integer	noBooksissued
15	PatronRecord	Integer	maxBookLimit
16	PatronRecord	String	name
17	PatronRecord	String	address
18	PatronRecord	String	phoneNo
19	PatronRecord	Double	finesOwned

```
import android.app.Activity;
import android.content.ContentValues;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.*;
public class Librarian_Create_Activity extends Activity {
  private EditText txtname;
  private EditText txtlibrarianID;
  private Button btnCreate;
  private String name;
  private Integer librarianID;
  SQLiteDatabase db;
  String path=Environment.getExternalStorageDirectory().g
  String dbPath=path + "/Librarian";
  String TableName="LibrarianTable";
  SQLiteDatabase db;
  String path=Environment.getExternalStorageDirectory().g
  String dbPath=path + "/Librarian";
  String TableName="LibrarianTable";
  protected void onCreate(  Bundle savedInstanceState){
   super.onCreate(savedInstanceState);
   setContentView(R.layout.Librarian);
   txtname=(EditText)findViewById(R.id.txtname);
   btnCreate=(Button)findViewById(R.id.btnCreate);
   btnCreate.setOnClickListener(new onClick());
   db=SQLiteDatabase.openDatabase(dbPath,null,SQLite
  }
  protected void onCreate(  Bundle savedInstanceState){
   super.onCreate(savedInstanceState);
   setContentView(R.layout.Librarian);
   txtlibrarianID=(EditText)findViewById(R.id.txtlibrarianID);
```

Figure 7. Code generator.

Table 3. Generated source code result.

Layer	Type	Files	SLOC	Generated Time (s)
Model	Java	6	188	1.98126
View	XML	6	99	0.14854
Controller	Android	6	329	0.29673
Total	Java + XML	18	**616**	**2.42653**

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 UMPLE 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.

No	Project	Parser Efficiency		Code Generation Efficiency Java + XML + Android			COCOMOII Estimated Value	
		Precision	Recall	SLOC	File	Time (s)	Time (Months)	Effort (Person-Months)
1	Online Golf Store System	100%	90.67%	797	27	3.719197	4.8	2.3
2	Online Shopping System	100%	98.34%	853	21	2.058524	4.9	2.5
3	Banking Structure System	99.27%	100%	630	21	1.764526	4.4	1.8
4	Patient Information System	100%	97.85%	654	15	1.977318	4.5	1.8
5	Order & Payment System	100%	97.77%	611	24	1.846651	4.4	1.7
:								

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

[1] Vaupel, S. and Taentzer, G. (2014) Model-Driven Development of Mobile Applications Allowing Role-Driven Variants, MODELS'2014, Philipps-University at Marburg, Gernany.

[2] Poole, J.D. (2001) Model-Driven Architecture: Vision, Standards and Emerging Technologies, ECOOP' Workshop on Meta-modeling and Adaptive Object Models, Hyperion Solutions Corporation.

[3] Browne, P. (2009) JBoss Drools Business Rules. Packt Publishing, 1847196063.

[4] Kriouile, A., Addamssiri, N. and Gadi, T. (2015) An MDA Method for Automatic Transformation of Models from CIM to PIM. *American Journal of Software Engineering and Applications*, **4**, 1-14. https://doi.org/10.11648/j.ajsea.20150401.11

[5] Benouda, H., Essbai, R., Azizi, M. and Moussaoui, M. (2016) Modeling and Code

Generation of Android Applications Using Acceleo. *International Journal of Software Engineering and Its Applications*, **10**, 83-94. https://doi.org/10.14257/ijseia.2016.10.3.08

[6] Bousetta, B., Beggar, O.E. and Gadi T. (2014) A Model Transformation Approach for Code Generation From State Machine Diagram. *IADIS International Journal on Computer Science and Information Systems*' 2014, **9**, 1-15.

[7] Son, H., Kim, W. and Chul, R. (2013) MOF Based Code Generation Method for Android Platform. *International Journal of Software Engineering and Its Application*, **7**, Hongik University, Sehong Campus, Korea.

[8] Omar, B. and Lethbridge, T. (2013) Model Oriented Programming: Bridging the Code-Model Divide, ICSE Workshop on Modeling in Software Engineering, Modeling in Software Engineering (MiSE), University of Ottawa, Canada, 69-75.

[9] Parada, A. and Marques, M. (2015) Automating Mobile Application Development: UML-Based Code Generation for Android and Windows Phone. *Journal of Theoretical and Applied Informatics* (*RITA*), **22**, Pelotas, Brazil.

[10] Mohaghegh, P. (2010) An Approach for Empirical Evaluation of Model Driven Engineering in Multiple Dimensions, MODELPLEX, Oslo, Norway.

[11] Sunkle, S. and Kullkarni, V. (2012) Cost Estimation for Model-Driven Engineering. *Proceedings of the* 15*th international conference on Model Driven Engineering Languages and Systems*. https://doi.org/10.1007/978-3-642-33666-9_42

[12] Abran, A., Deshnarnais, J. and Mohammand, Z. (2015) Productivity-Based Software Estimation Models and Process Improvement: An Empirical Study. *International Journal on Advances in Software*, **8**.

[13] Eithu, E. and Nwe, N. (2017). Model Driven Engineering: Automatic Code Generation of Android from UMPLE, 15*th International Conference on Computer Application* (in Press), Yangon, Myanmar.

[14] https://github.com/umple/umple