Model Driven Development of Mobile Applications Using Drools Knowledge-based Rule

Ei Ei Thu
Universities of Computer Studies
Mandalay, Myanmar
eieithuet@gmail.com

Nwe Nwe
Universities of Computer Studies
Mandalay, Myanmar
nwenwemdy08@gmail.com

***Abstract*—Nowadays, the significance of automatic transformation of object oriented code from design models has increased due to its benefits such as cost reduction and time efficiency. Model driven development (MDD) absolutely needs automatic approach and method to generate system from model. This paper contributes the efficient Drools rule-based transformation approach to automate the mobile application development process. In our approach, the textual model Umple considered as primary artifacts that drive the whole mobile application. The consistency of source and target model and the assessment of transformability are the critical issues in model transformation domain. In this paper, we attempt to address these issues with measuring the accuracy of consistency between source and target model and assessing the transformability using object oriented metrics. Results reveal that our approach achieved the high accuracy of consistency and sound quality of transformability.**

Keywords—model transformation; Drools rule-based; textual model

I. INTRODUCTION

The mobile application development industry is increasingly growing up due to the intensive use of applications in mobile devices. Therefore mobile apps development faces production problem in short time-to-market and to achieve high productivity. It motivates the applying of model driven development (MDD) in mobile apps development domain. Model-Driven Engineering (MDE) is a software development process that has gained popularity in the recent years. Unlike traditional software engineering processes, MDE is centered on models, instead of code. By using model transformations, model can be translated from one language to another, resulting in a separation of program architecture and execution platform.

The adoption of MDD can simplify the development of mobile apps, reducing significantly technical complexity and development costs. Model driven Engineering (MDE) is a software engineering discipline in which models play a central role throughout the entire development process. Model transformation is one of the prominent features and the rising research area of MDE. MDE promotes models to primary artifacts that drive the whole development process. The success of MDE generates a strong need for techniques and method for developing model transformations.

In this paper, we contribute rule-based model driven approach for mobile applications based on Umple model. Umple is a model-oriented programming language that supports modeling using a textual notation just like other high-level programming languages. It blurs the distinction between code and modeling, enabling modelers and coders to collaborate on models in a way similar to code-based collaboration. It is a multi-faceted technology allowing users to integrate modeling into software development straightforwardly [1]. The proposed efficient transformation rules are constructed on Drools knowledge based. Drools is a business rule management system with a forward and backward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm [2]. It is the leading open-source rules engine written in java.

Maintaining consistency between model and the corresponding code becomes challenging [4]. Recently, some researchers have proposed MDD approaches for mobile application development [3] [5] [9] [10]. These prior researches just have focused on meta-model building and defined transformation rule specifications in template-based approach. Although there are already some approaches to model-driven development of mobile applications, there is still need to tackle in consistency and generated source code quality issues. Additionally, we address the following issues in the model driven mobile apps development:

1) Measuring consistency between source and target model
2) Evaluating the transformability with object-oriented source code quality measurement

In order to address these issues, we proposed efficient Drools transformation rules that can generate Model, View, and Controller (MVC) pattern of mobile application. The main results of the model transformation are plain old java object (POJO) class for model layer, XML file for view layer and android activity class for controller layer.

The remainder of the paper is organized in the following way: section 2 surveys the related work. Section 3 introduces our contribution of knowledge based model driven development architecture in the design process. Section 4 presents experiments and evaluation of proposed approach. Finally, we conclude and discuss our research perspectives in section 5.

978-1-5090-5756-6/17/$31.00 ©2017 IEEE
SERA 2017, June 7-9, 2017, London, UK

II. RELATED WORK

There is increasing attention towards the generation of source code from modeling languages. Several researchers propose model driven approach for the different aspects of mobile applications. Son,H.S, et.al [3] [5] defined the meta-model and model transformation rule for model driven android apps development. Omar, E.B, Brahim,B [6] also defined ATLAS transformation rules for UML sequence diagram to generate enterprise java bean code (EJB). Parada, A.G et.al [8][9] presented enhance code generation tool for android source code based on UML class and sequence diagram. Steeg, C.C. and Gotz, F. [7] specified meta-model with Ecore and transformation rules with Xpand templates for entity relationship diagram to generate android SQLite database model.

These approaches vary in several ways. Their approaches differ in the choice of input model and transformation rules. Most of these approaches are based on graphical modeling languages and some are based on textual modeling languages. In contrast to our approach, the previous approaches are focused on predefined meta-modeling while our approach automatically parse and extract syntax from input model. And our approach specified transformation rules in object pattern matching approach. JUSE4Android [10] [11] is the most related study with our proposed approach. It is also based on textual modeling languages. Unlike our approach, it is adding annotation into JUSE model and transform into android source according to the predefined meaning of annotation. Therefore their generated source code contains some unnecessary file in the project.

The author [12] proposed the approach for empirical evaluation of model driven engineering in multiple dimensions. Their case studies included qualitative (expert judgments) and quantitative data (metrics) evaluations. They supposed that the productivity and defect detection rate are the popular metrics for measuring automation degree of MDD process. According to the literature, the comparative study for measuring the transformability of MDD generated source code using object oriented metrics has not been conducted earlier. Thus we obtain more reliable findings.

III. RULE BASED MODEL TRANSFORMATION FRAMEWORK

Model transformation is a central concept in model-driven development approaches, as it provides a mechanism for automating the manipulation of models. A transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into target language. 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.

In this section, we provide an overview of the framework and their underlying architecture. The architecture is divided into three major parts corresponding to the main capabilities of the proposed framework. These components are parser, transformer and code generator. The transformer deploys the knowledge base rule engine. The parser receives an input model, written in Umple language, tokenizes it and passes it to the next component transformer. The transformer processes the tokens previously obtained and transforms them into internal representation consistent with target source code model using predefined set of Drools mapping rules. Then the code generator translates the internal representation into target artifacts; source code as Java, XML and android activity class. Each component is tested independently to ensure that the input is processed correctly and the output is produced valid. Figure 1 shows the overall architecture of the proposed system.

A. Rule-based Inference System

In this paper, we present a rule-based model driven approach to generate android application from text-based modeling language. Rule languages and inference engines incorporate reasoning capabilities in mobile application development systems. The proposed model transformation rule is based on Drools rule inference engine [2]. It improves the generation of mobile applications source code and introducing new concern in model driven mobile engineering. The core of the Drools suite is an advanced Inference Engine using an improved Rete algorithm for object pattern matching. Rules are stored in the production memory, while facts are maintained in the working memory. The production memory remains unchanged during an analysis session, i.e. no rules are added or removed or changed. The contents of the working memory on the other hand can change. Facts may be modified, removed or added, by executing rules or from external sources. After a change in the working memory, the inference engine is triggered and it determines which rules become "true" for the given facts. If there are multiple selected rules, their execution order will be managed via the Agenda, using a conflict resolution strategy.

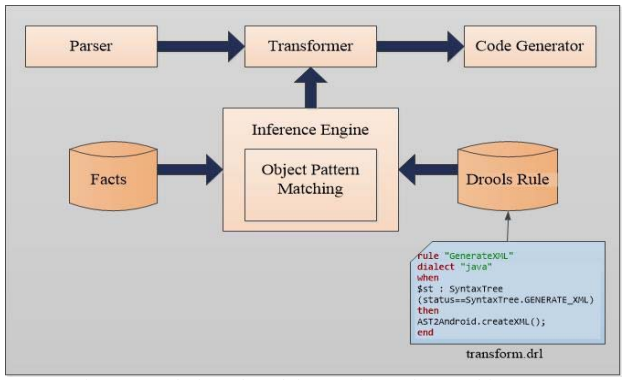

Figure 1 Rule-basedModel Transformation Architecture

B. Drools Transformation Rule

Drools rules are defined using Java-like language. The proposed rule engine consists three portions: umple2model, umple2view and umple2controller according to android model, view and controller perspective. Table 1 shows the sample form of drools transformation rule for simple variable

declaration. 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. 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.

TABLE I. DROOLS TANSFORMATION 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**

IV. EXPERIMENTAL RESULTS AND COMPARISON

In order to evaluate the quality of proposed transformation process, we have conducted the measurement of consistency between source and target model and evaluating the quality of transformability with object oriented metrics. We perform a comparative study on our proposed approach and JUSE4Android [10] [11] research approach by using object-oriented metrics [13] [14], which comprise measures for size and complexity, coupling, cohesion and inheritance. For the size and complexity criteria, number of methods, number of attributes, weighted method and cyclomatic complexity are used to measure. Afferent coupling and efferent coupling metrics are used to measure for coupling criteria. Cohesion is measured by lack of cohesion metrics and number of children and depth of inheritance tree are used to measure for degree of inheritance. Table 3 shows the software quality metrics and expected outcome for comparative study.

A. Parser Consistency Analysis

In this section, we carried out the parser analysis to ensure that Umple models are parsed and tokenized as expect. It helps to provide confidence that the proposed parser is correctly tokenized and processed into an internal representation consistent with the source model. The Parseval metrics is used to calculate precision and recall over the constituents in parse tree. We computed the precision and recall by using the following equation (1) and (2). Precision assesses the proportion of the constructs (type, package, attributes, LOC) identified that are valid by the ground truth Umple static code analyzer [18]. Recall assesses the proportion of the independently identified constructs that are found by or approach. True positive (TP) means correctly identified by proposed parser, false positive (FP) means misidentified by proposed parser and false negative (FN) means missed by proposed parser. Table 2 and Figure 2 show the result for the analysis of proposed parser. The analysis has been conducted over 18 Umple model projects. Additionally, these projects contain over 90 classes, 280 attributes and 600 lines of code (LOC). According to the results, the proposed parser achieves the desire consistency between source and target artifacts.

$$Precision = \frac{TP}{(TP+FP)} \quad (1)$$

$$Recall = \frac{TP}{(TP+FN)} \quad (2)$$

TABLE II. PARSER ANALYSIS PRECISION AND RECALL RESULTS

	Precision	**Recall**
No. Classes	100%	97.36%
No. Attributes	100%	98.31%
LOC	99.38%	98.14%

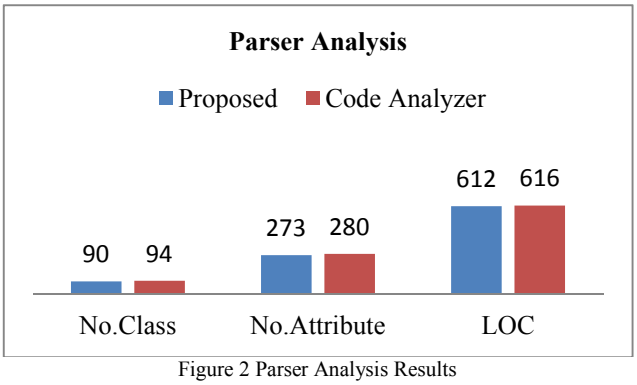

Figure 2 Parser Analysis Results

B. Evaluating the Quality of Transformation

In order to ensure the proposed model transformation approach quality, we assessing the quality of model transformation from generated source code quality. As a preliminary study, we perform the comparative study between proposed approach's generated source code and prior approach's generated source code [10] [11] by employing five commonly used object oriented metrics. Metrics are important to understand, steer, and control the development of complex software. Moreover, metrics can be used as indicators of software quality. By investigating on 18 projects, we attempt to address the following issues:

1) Which approach can help generate classes of small size and low complexity?
2) Which approach can help write classes of high cohesion and low coupling?
3) Which approach can help create good hierarchies?

The proposed approach was compared to existing approach (JUSE4Android) against the software quality criteria. Our experiment is conducted on 18 projects in both Umple and USE model that give solutions to common goal. Table 3 shows the quality criteria for comparative study.

TABLE III. OBJECT ORIENTED QUALITY CRITERIA

Quality Criteria for Comparative Study				
No	**Quality Criteria**	**Metrics**	**Acronyms**	**Expected**
1	Sized and Complexity	No.of Attributes	NOA	Low
		Cyclomatic Complexity	CC	Low
2	Coupling	Efferent Coupling	CE	Low
3	Cohesion	Lack of Cohesion	LCOM	Low
4	Inheritance	Depth of Inheritance Tree	DIT	High

TABLE IV. METRIC VAUES COLLECTED BY ECLIPSE METRICS PLUG IN [19]

Project	**NOA**		**CC**		**CE**		**LCOM**		**DIT**	
	Proposed	Prior	Proposed	Prior	Proposed	Prior	Proposed	Prior	Proposed	Prior
Project_1	1.929	3.11	1.111	2.483	3.667	4.167	0.074	0.255	1.75	1.541
Project_2	0.711	1.907	1.154	1.933	1.667	3.1	0.013	0.165	1.474	1.453
Project_3	0.543	1.772	1.158	1.828	1.333	3	0.014	0.152	1.4	1.418
Project_4	1.6	2.808	1.059	2.243	1.667	3.917	0.098	0.224	1.45	1.458
Project_5	1.22	2.129	1.116	2.005	2	3.182	0.047	0.178	1.537	1.484
Project_6	1.72	2.439	1.111	2.131	3	3.357	0.078	0.204	1.68	1.553
Project_7	0.531	1.667	1.125	1.672	1	2.875	0.023	0.138	1.312	1.375
Project_8	1.22	2.301	1.116	2.091	2	3.545	0.047	0.198	1.537	1.456
Project_9	0.543	1.772	1.158	1.828	1.333	3	0.014	0.152	1.4	1.418
Project_10	1.851	2.85	1.099	2.328	2.667	4.25	0.065	0.234	1.638	1.441
Project_11	2.786	3.183	1.082	2.394	3.667	4.062	0.108	0.248	1.75	1.523
Project_12	2.5	2.684	1.082	2.127	3	3.357	0.102	0.207	1.68	1.553
Project_13	1.293	2.151	1.111	2.005	2	3.182	0.048	0.178	1.537	1.484
Project_14	1.439	2.194	1.102	2.005	2	3.182	0.054	0.179	1.537	1.484
Project_15	1.604	2.446	1.123	2.158	3.333	3.4	0.06	0.209	1.717	1.57
Project_16	1.943	2.545	1.106	2.156	3.333	3.4	0.091	0.21	1.717	1.57
Project_17	1.057	2.146	1.097	1.976	1.333	3.444	0.044	0.181	1.4	1.393
Project_18	2.136	2.46	1.081	2.056	2.333	3.25	0.082	0.19	1.591	1.51

C. *Data Collection*

In order to perform the metrics based comparison, we collected the metric values by using Eclipse Metrics Plug-in [19]. It is an open source metrics calculation tools which measures various metrics and detects cycles in package and type dependencies. Specifically, the data was collected by the following steps. At the first step, we generate the android applications from each proposed and prior approach. In next step, we enable Eclipse Metrics Plug-in on each generate source code that give common solution. Finally, we extracted the mean, standard deviation and maximum metric values for each generated source code. In our comparative study, we collected the average metric values from the proposed and prior generated source code with respect to the quality criteria. Metrics are related to quality criteria because different measures for a metric indicate different quality levels with respect to a given criteria. Table 4 shows extracted metrics values.

D. *Size and Complexity Measurement*

At this stage, number of attributes (NOA) [15] is used to measure the size of a class and cyclomatic complexity (CC) are applied to measure the complexity of a class through calculating the total complexity of its member functions in different ways. Since classes are suggested to be designed as concise as possible, these metrics are expected to be low in their values. CC [16] means the sum of complexity of all methods in a class based on the information flow. We gathered for each class level metric its average metric value of all classes in each projects. In term of class complexity, we observe a significant difference between proposed and prior research. This indicates that our proposed code generator's results are not significantly complex than prior code generator. Figure 3 and 4 shows the comparative results on number of attributes and cyclomatic complexity measurement for size and complexity quality criteria.

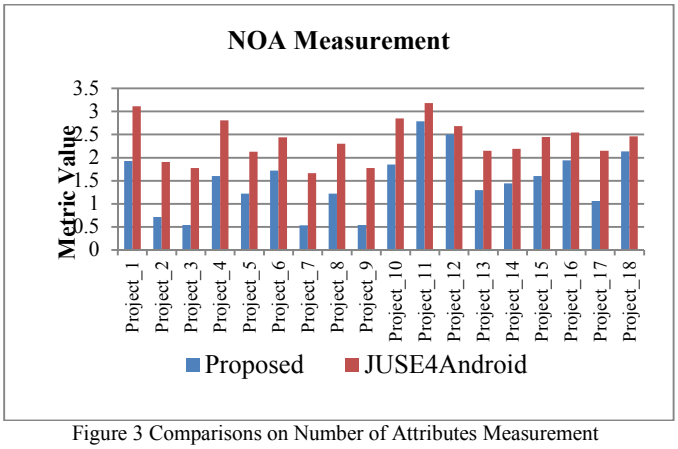

Figure 3 Comparisons on Number of Attributes Measurement

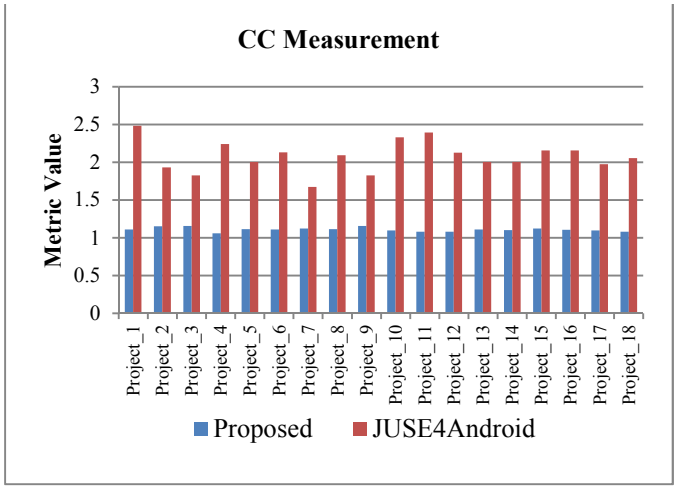

Figure 4 Comparisons on Cyclomatic Complexity Measurement

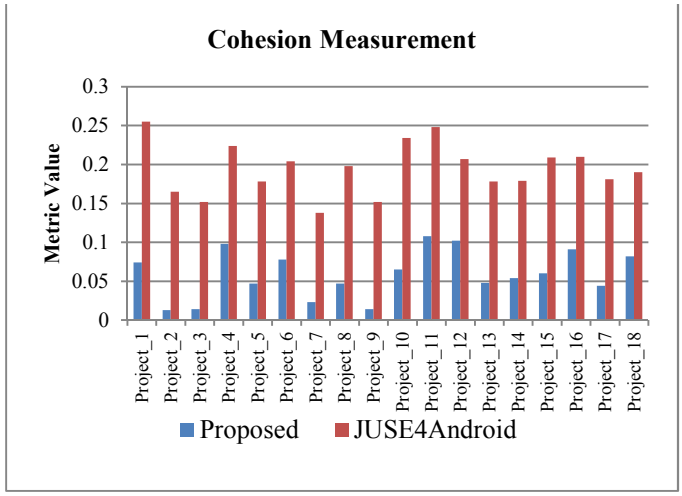

Figure 6 Comparisons on Cohesion Measurement

E. Coupling and Cohesion Measurement

We employ the efferent coupling and lack of cohesion of method metrics to investigate the degree of cohesion and coupling between objects. Efferent coupling (CE) is used to evaluate the coupling of all classes at the system level. Since highly coupled classes are less object-oriented, low metric values are preferable. Cohesion is measured with lack of cohesion in methods (LCOM) [17], which is calculated in different way to reflect the interactions between member functions. A low LCOM value is expected. To summarize, the core observation from the coupling metrics is that our proposed code generator results is loosely couple than other prior approach. Figure 5 sows the degree of object coupling measurement between proposed and prior JUSE4Android approach. According to the Figure 6, we find the cohesion metrics show completely different results. From this reasoning, we conclude that our proposed approach generator' source code is tend to be more cohesive than JUSE4Android approach.

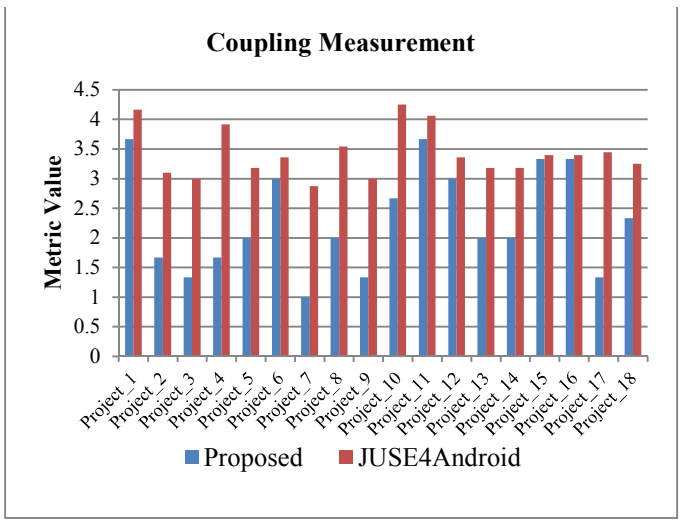

Figure 5 Comparisons on Coupling Measurement

F. Measurement for Inheritance

The inheritance metrics give us information about the inheritance tree of the system. Inheritance is a key feature of the object-oriented paradigm. Inheritance metrics measure various aspects of inheritance such as depth and breadth in a hierarchy and overriding complexity. Depth of inheritance tree (DIT) [17] is class-level metric, which expresses class inheritance through the depth of type inheritance. Since it is suggested to build hierarchical type tree in the object-oriented systems, the high inheritance metric values are expected. DIT is a class viewpoint metric generating values per class. It follows nested outward projection as it traces outgoing parent links in the inheritance chain. In the degree of inheritance measurement, we observe that the results does not reveal significantly outperform in each approach. Thus, we can conclude that both proposed and JUSE4Android approaches have the same degree level of inheritance. Figure 7 shows the comparison results for depth of inheritance tree.

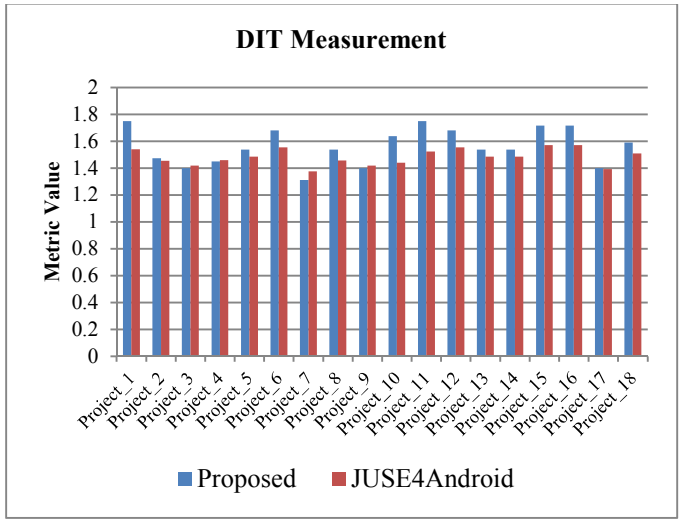

Figure 7 Comparisons on Depth of Inheritance Measurement

G. Results And Discussion

The investigation results based on object-oriented metrics show the following findings:

1) To summarize, the initial observation from class size and complexity metrics is that our proposed approach' generated classes size are significantly lower than the prior approach. Therefore, our proposed approach can help to generate classes of small size and low complexity.
2) The prior approach generated classes are significantly more likely to couple than our approach and our approach is more likely to cohesive than prior approach. Therefore, our approach can help to write classes of high cohesion and low coupling.
3) There is no significant result is revealed in DIT metrics. Both DIT values are likely to be nearly the same. In some cases, our proposed approach slightly helps to create good type hierarchies. Figure 8 shows the overall comparative measurement for each metrics over 18 projects.

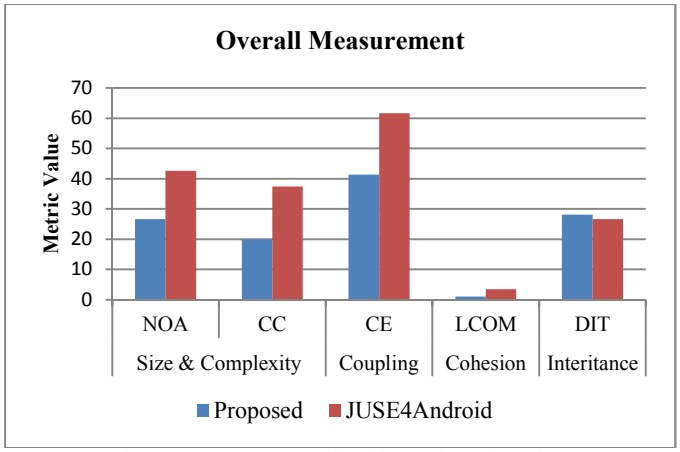

Figure 8 Measurement with Object Oriented Metrics

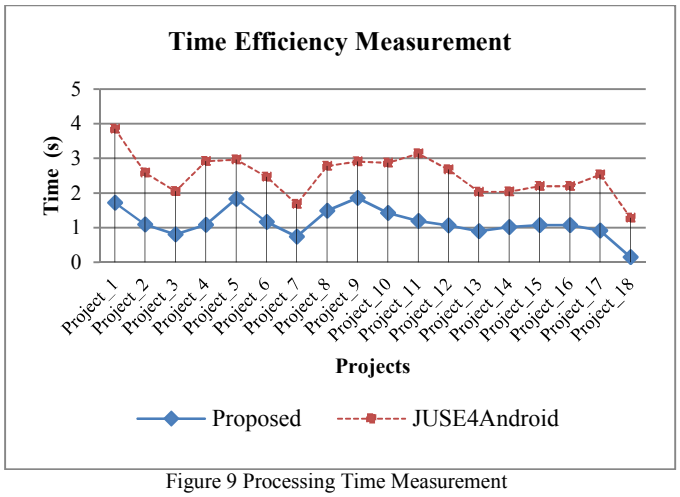

Figure 9 Processing Time Measurement

H. Processing Time Measurement

We have instrumented our proposed approach with timers to measure the time taken to process an input file and produce the target source code. Figure 9 summarizes the execution times in seconds taken to transform source model to target source code over 18 projects. The proposed approach is implemented in Drools Rule language, JDOM, Java and running on 4G RAM with Core i7 (2.0GHz). The prior approach is implemented with USE 3.0, Java and JDOM.

V. Conclusion

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 relatively consistent with input Umple artifacts. In this paper, we address the model driven quality measurement by using object oriented metrics. The empirical results show that the proposed transformation approach can generate source code in small size and complexity, high cohesion and low coupling and good inheritance perspective. Accordingly, our approach obtained the high accuracy of consistency and sound quality of transformability.

References

[1] Badreddin,O, Lethbridge,T.C, Forward,A.: A Novel Approach to Versioning and Merging Model and Code Uniformaly, in MODELSWARD 2014 – Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, 2014.

[2] De Lay,E, Jacobs, D.: Rules-based Analysis with JBoss Drools : Adding Intelligence to Automation, ICALEPCS 2011 – Proceeding of ICALEPCS,Genoble,France.

[3] Son, H.S, Kim, W.Y and Chul, R.Y.: MOF based Code Generation Method for Android Platform, International Journal of Software Engineering and Its application, Vol.7.No3. Hongik University, Sehong Campus,Korea, 2013.

[4] Pallavi, K, Suita, P.U.: Model Transformation: Concept, Current Trends and Challenges, International Journal of Computer Applications, Volume 119-No.14, Nasik, Manarashtra, India, 2015.

[5] Son, H.S, Kim,J.S, Chul, R.Y.: SMTL Oriented Model Transformation Mechanism for Heterogeneous Smart Mobile Models, International Journal of Software Engineering and its Applications, Vol.7. No.3, Sejong Campus, Korea, 2013.

[6] Omar, E.B, Brahim,B.: Automatic Code Generation by Model Transformation From Sequence Diagram of System's Internal Behavior, International Journal of Information Technology, Hassan 1st University, Morocco, 2012.

[7] Steeg, C.C, Gotz,F.: Model Driven Data Management in Android with the Android Content Provider, https://code.google.com/archive/p/mdsd-android-content-provider/ Bingen, Germany, 2011.

[8] Parada,A.G, Lisane,B.: A Model Driven Approach for Android Applications Development, Brazilian Symposium on Computing System Engineering (SBESC), Pelotas, Brazil, 2012.

[9] Parada,A.G, Milena,R.S.: Automating mobile application development: UML-based code generation for Android and Windows Phone, Journal of Theoretical and Applied Informatics(RITA), Volume 22, Pelotas, Brazil, 2015.

[10] Silva, L.P, Abreu,F.B.: A Model-Driven Approach for Mobile Business Information Systems Applications, ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, MODELS 2014, QUASAR/ISTAR/ISCTE-IUL, Lisboa, Portugal.

[11] Silva, L.P, Abreu,F.B.: Model-Driven GUI Generation and Navigation for Android BIS Apps, MODELSWARD 2014, Portugal.

[12] Mohagheghi,p.: An Approach for Empirical Evaluation of Model Driven Engineering in Multiple Dimensions, MODELPLEX, Oslo, Norway, 2010.

[13] Henderson-Sellers, B.: Object-oriented metrics: measures of complexity, Prentice Hall, 1995.

[14] Chidamber, S.R, Kamerer, C.F.: A metrics suite for object-oriented design, IEEE Transaction of Software Engineering, pp-(20)6, 1994: 476-493.

[15] Baowen, X, Di.W.: A metrics-based Comparative Study on Object-Oriented Programming Languages, 27th International Conference on Software Engineering and Knowledge Engineering, SEKE' 2015, USA.

[16] Kocaguneli, Ekrem, et al.: Prest: An Intelligent Software Metrics Extraction, Analysis and Defect Prediction Tool, 21th International Conference on Software Engineering and Knowledge Engineering, SEKE'2009, USA.

[17] Rudiger, L, Jonas, L.: Comparing Software Metrics Tools, International Symposium on Software Testing and Analysis, ISSTA'2008, USA.

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

[19] http://metrics.sourceforge.net/