Journal of Theoretical and Applied Information Technology
15th June 2017. Vol.95. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN EFFICIENT DESIGN MODEL VALIDATION FOR THE QUALITY SOFTWARE DEVELOPMENT

[1]MULUGU. NARENDHAR, [2]Dr.K. ANURADHA
[1] Department of CSE, SNTI, Hyderabad, Telangana, India
[2] Department of CSE, GRIET, Hyderabad, Telangana, India

E-mail: [1]narendharm4@gmail.com, [2]kodali.anuradha@yahoo.com

ABSTRACT

Models are the main artifacts of the software development process which generally follows a model-based paradigm. Business process models and data models play an important role in building information systems and to represent the perspective on business knowledge and close depended on relations, which is important to be validated to maintain the fault free and quality software development. Therefore, the design relevant models are acts as the blueprints for the quality assurance in software development. In this paper, we propose an efficient design model validation (DMV) supported utilizing the prescribed "specification language" for the proclaiming construction of transformation attributes based on the invariants, pre and post-conditions for generating partial functions used can be transformed for testing evaluation. We broaden the utilize of specification languages for automatic creation of input test models. We also used the prototypes of the Eclipse plug-in to provide specifications and calculations of metrics in related to models supported by the Eclipse Modeling Framework.

Keyword: *Design Modelling, Model Transformation, Quality Software Development, Validation Metrics*

1. INTRODUCTION

Model transformation is a pillar of model-driven engineering (MDE), it must be expanded by means of the right engineering standards to ensure accuracy. However, generally of the model conversion technologies are now focused on implementation phase support, and there are few specifications for requirements, design, or conversion tests. The outcome, conversions are often hacked, manipulated, difficult to maintain, inaccurate, or buggy. There are many requirements modeling methods and requirements model validation methods, and other requirements modeling methods from dissimilar points of observation in software systems reflect dissimilar modeling conceptions.

The prerequisites modeling methods that are aggressively attracting much consideration in intellectual fields include "object-oriented methods", "side-oriented methods", "function-oriented methods", and "goal-oriented methods" [21]. Model verification supports are different for different demand models such as verification method supported on semantic analysis [1], verification method supported on the inference of state machine [2], verification method supported on ontology constraint [3], [4]. To resolve requirements validation issues from other ways to a certain extent. The concept of model-based software development is becoming additional popularity because it guarantees to improve the effectiveness and superiority of software development. It is prudent to solve artifact quality problems during early software development phases, such as the quality of the models you have already joined. In particular, in model-driven software development, models are the main artifacts in which overall software product quality assurance depends heavily on the quality assurance of the software model involved.

The quality of the software model consists of several dimensions. The paper considers a model quality assurance processes that focus on the syntactic dimensions of model quality. Syntax quality aspects are all aspects that can be found only in the model syntax. This includes, of course, not only stability with language syntax definitions [11] but also other aspects such as conceptual integrity using the same patterns and standards in comparable modeling situations and adherence to modeling rules specifically defined

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

for software projects. These quality aspects and other quality aspects are discussed in [1], for example, where authors propose taxonomies for software model quality.

The model-driven approach to software development and software development industry are acquisition increasing concentration. Traditional development techniques that tend to look on implementation, unlike the model-driven software development at all stages of the software development process, it focused on the use of models. As a result of these changes the way the software are designed and maintained, and the test has changed significantly. Model development supported on the "Unified Modeling Language" (UML) [8] and used in numerous researchers "state machine diagrams", "use case diagrams", and "sequence diagrams" as UML diagrams for using in test cases. The main improvements of this model are supported by testing techniques, software testing activities for the early stages of the development process moving and design test cases are self-determining of the specific design implementation to increase productivity and quality [4], [5].

Model transformations indicate how the elements of the source metamodel are transformed from the elements of the objective metamodel [13]. The source metamodel fully indicates the complete input model set, which is the input domain of the transformation. In this circumstance, the suggestion is to assess the suitability of the test model for the scope of the source metamodel. For example, a test model must instantiate everyone class and each relation in the source metamodel at the majority once. The following provides test compliance criteria supported on the scope of the source metamodel. We also examine the automated creation of test models that meet these principles [7].

In this paper, we deal with the problems by extending the domain functionality with a set of input test models from the transformation specification requirements. As a result, we guarantee the specification range of the specification. The input model is premeditated by means of the "SAT resolution technique" for the ATL expressions created in specification [9] and can be selected from a range of seven levels to achieve various levels of comprehensiveness when tested.

The other part of the paper is structured as follows. Section-2 reconsider the related works which looking on to the existing advancement in the model transformation testing [18]. Afterward, Section-3 discuss the proposed framework for design model validation and the effectiveness of different coverage criteria and section-4 describes the experiment setup and results based on the certain level of specification coverage. Section-5 draws the conclusion and future prospects of the proposed works.

2. RELATED WORKS

Verification and validation are essential for software development. In the circumstance of model transformation, efforts aimed at verification and validation can be classified into three categories. (1) a work that can use formal languages to implement transformations to ensure or analyze transform belongings such as "termination" or "determinism" [14]; (2) Conversion to formal domain (iii) and transformation testing for analysis such as "Petri net", "Logical rewrite" [10], "Satisfaction (SAT) problem" [15]. The first two approaches can analyze common characteristics such as "termination, determinism, rule independence, rule applicability, or reachability of system state". This paper follows the third methodology, therefore we will rethink the work of model conversion testing, paying particular concentration to the "black box testing approach" because it is the extent of the work to be presented in this paper. There are three major problems in the model conversion test [16], as the creation of the input test model, description of the scope of the verification scope, and function creation.

2.1Creation of Input models

Most of the work on dealing with input test model creation is for black box testing and simply takes into account the characteristics of the input metamodel but does not consider the source of the conversion. In the Fozr instance [20], the author performs an input metamodel and some typical coverage criteria, for example, the automatic creation of an input test model supported on the partitioning and number of classes of attribute values. Using this approach [23], the authors present experiments in which dissimilar input test sets are produced for diverse scheme and rules to obtain a score of variation in the range of 72-87%.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In [9], the creation of the input test model should be done by hand, using an imperative language with arbitrarily chosen attribute values and end-of-connection capabilities. The input model is handcrafted, but it is not necessary limitations of the literature and must comply with the simplified input metamodel. This is so-called sub-model alloys can be used for testing and has become a primary constraint to find a suitable meta model instances solver. Model fragments instead of this sub-model are used to cover some of the models and meta-model criteria, a similar approach is presented in [20].

2.2 Validation Coverage criteria

Nowadays "black box testing" approaches for model transformations do not take into account coverage criteria or support input metamodel coverage of attribute value, the number of related classes and associations, etc. [17]. The disadvantage of this operation is that some variant attributes that were not considered when testing the model (manual or automatic) may remain untested. In turn, the white box testing approach typically adopts a mixture of classic "white box coverage criteria" and "variant-specific test criteria" such as regulation coverage or decision-making criteria. For example, in [24], the author measures the range of determination of an input test set. Nevertheless, there is no arguments of the association among this coverage criterion and the efficiency of the test set.

2.3 Construction of functions

Distinguishes whole functions from partial functions with respect to the third task of the model conversion test. The former is defined as having an output model. For example, the test case for "C-SAW Conversion Language" [26] consists of an initial model and a supposed output model. The incomplete function represents the contract that the input and output models of the transformation must meet. The majority suggestions for incomplete functions utilize the "object constraint language" (OCL) to specify contracts [21]. The approach of [4], [19] is related to the philosophy of the "xUnit framework" and purposes can be particular as "OCL / EOL assertions". There are earlier approaches that allow specification of incomplete functions as "graph patterns" or "model fragments" [6]. In summarized, we can see that some conversion test approaches make available automated test carrying out, but do not manually maintain the input model creation and requirements manually [16]. The previous works look into the automatic creation of the input model but don't take into account the varying degrees of completeness for conversion properties or testing.

In this paper, we present an approach implementing specification-based conversion testing that automates the creation of input test models, functions and carrying out test scripts in the same transformation specification. As a prominent characteristic, you can use the produced model to test the relevant properties of the transformation. In addition, we describe a set of specification exposure condition to facilitate ever-increasing levels of thorough testing.

In this case, we aim to create a test model that guarantees the conversion requirements [12], the full metamodel coverage, i.e., the creation of all meta model illustrations of a definite range. Automated test case creation, as illustrated in this paper, which can also measure up to the technologies other than software testing, for example, "testing of executable language definitions" or a "grammar-based test method" [29].

3. FRAMEWORK FOR DESIGN MODEL VALIDATION

Figure 1 shows our work plan for the proposed Design Model Validation (DMV) approach. In the first action, the designer uses the "pattern-based model" versus the "model specification language" to specify the requirements of the transformation such as pre-requisites, post-conditions, and invariants. Developers are able to use this specification as a guideline to execute transformations utilizing their preferred language, such as "ATL" [27], "ETL" [28], and so on. In fact, with our familiarity, we have found that specifications and implementations are often iterative and sophisticated.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

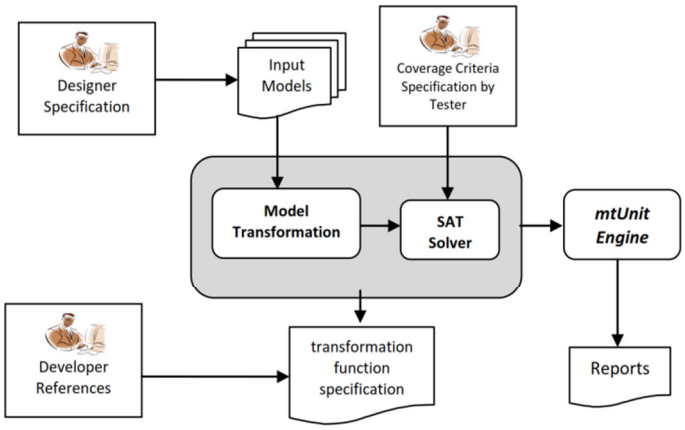

Figure. 1: Framework for Design Model Validation

Starting with the specification, the conversion assessment can automatically create an absolute test suite that can be utilized straight to test the conversion execution. This test experiment suite consists of: (i) a set of input test models that encode invariants and assertions of requirements [7], (ii) a set of input test models that can test specific requirements, (iii) A test script that automates the transform implementation for each test model, uses the function to check the conformity of the results, and reports the errors found using the mtUnit engine [30]. Approximately, the guideline for creating the input test model transforms the conditional and invariant conditions into an "object-constrained language" (OCL) invariant. By combining these invariant conditions, the "Satisfaction" (SAT) solver [15] A model that contains a combination of attributes. How you combine multiple properties to find a model depends on the selected scope type.

Our approach for Design Model Validation having the following steps:

1. We translate the belongings of the standard in the form suitable for model possession,
2. As a result of a specific strategy for creating expressions that require the satisfaction of multiple attributes in the created model,
3. Use the restriction solver to uncover a model that meets all the reliability limitations of the input metamodel and the specific combination of attributes (depending on the selected coverage).

A. Translation of properties in the specification

As a first action, we translate the specification into a language that can automate model creation. In particular, if there are possible solvers to find a model that satisfies the OCL constraint set, OCL is used as the target language [22], so there is no necessitate to parse OCL formulas into other languages in the properties of the specification. Nevertheless, this is our special choice, and we can use the framework in different target languages whenever translations are provided in our specification language.

The specification includes "pre-requisites", "post-conditions", and "invariants", but merely the prerequisites and invariants enclosed the information that is useful in creating the input model. The post-condition describes the attributes of the output model and is used only for function creation, not for the input model.

An invariant expression represents a property of the formation when a specific source pattern appears in the input model, certain target patterns should appear or not appear in the output model. It is, therefore, importance to create an input model that contains an instance of the source pattern and test whether the conversion of this model actually produces an output model that contains the target pattern.

As in list-1 shows, the proposed expression transformed. It iterates through the objects in the resource graph of the most important limit on lines 1 to 3 and makes sure that the inactive source graph does not appear on lines 4 to 7. This

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

code is created for everyone deactivation condition. The function condition in related to the condition that the passed object must satisfy: the continuation of the link specified in the invariant, the inequality for the object of the same type, and the OCL shape, which examines all the terms of the invariant for the input element. Possible conditions of invariants are ignored because they are surrounded by invariants. Additionally, if the invariant is negative, the created representation is the equal, but the non-particle is not preceded because the constant source portion is still a positive value.

```
1 o1.type ::  all-Instances() -> exists ( o1 | . . .
2         oi.type :: all-Instances() -> exists ( o1 | . . .
3                 conditions ( o1, . . . , oi )
4                         < and  not
5                                 oj.type :: all-Instances() -> exists ( oj | . . .
6                                 ok.type :: all-Instances() -> exists ( ok | . . .
7                                         conditions ( o1,...,oi, oj,..., ok ) > * ) . . . )
```

List-1: Outline for invariants

Often specifications include constant conditions with the same source and different targets. For example, Task 1 and Task 2 in Figure.1 utilized together with the tasks as sources. The previous indicates how to precisely interpret the job, and the latter identifies the wrong conversion. In this case, you can test the two invariants by creating an input model that contains the tasks. Therefore, duplicate source conditions, that is the same source of key constraints and inactive conditions) are removed from the created OCL representation set. Do not remove sub-assumptions so that size and context conditions test different models.

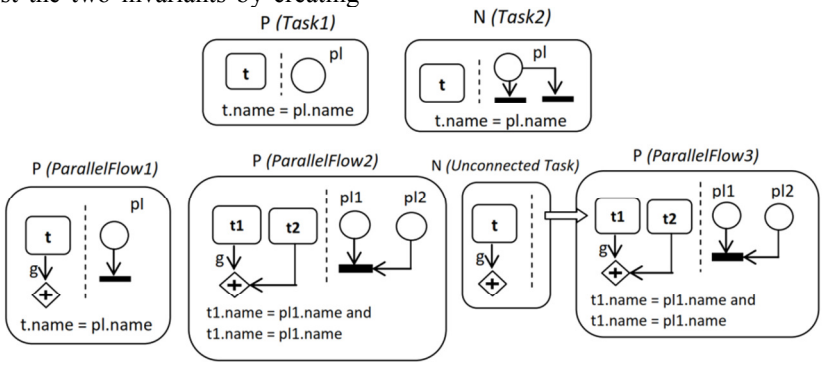

Figure. 2: Some invariants for the transformation

For example, in the invariant ParallelFlow3 in Figure-2, and List-1 lines 1 through 7 in related to the encoding of the immutable source pattern, while lines 8 through 9 encode the inactive state.

```
1 Task :: all-Instances() -> exists( t1 |
2         Task :: all-Instances() -> exists( t2 |
3                 ParallelFlow :: all-Instances() -> exists( g |
4                 SequenceFlow :: all-Instances() -> exists( s1 |
5                 s1.sourceRef = t1 and
6                         and s1.targetRef = g
7                         and t1 <> t2
8                 and not SequenceFlow :: all-Instances() -> exists( s2 |
9                         s2.sourceRef = t2 and s2.targetRef = g )))))
```

List-2: Expression for invariant Parallel Flow

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Finally, the prerequisite specifies the input model requirements for the transformation. That do not meet these prerequisites have to work a transformation of input samples appropriately. The validity of the transformation of the type of input is rarely achieved, but in the case of an external method or the transformation of the application is guaranteed. Therefore, it must meet all the prerequisites for the rule that adopts all of the specification created by the input models. To do this, we create constraints and create as pre-requisite for input on the OCL constraint satisfaction model which was created to increase the input of all the models include an OCL constraint code is shown List-3. The availability of lines 2 through 4 are shown in this expression finds all occurrences of the condition of the occurrence of the primary constraint to each one of them. If there is no expression, the result expression is similar as the invariant expression, and if it is negative, the created appearance is displayed as a leading line by <*not*> as line-1 in list-3.

```
1  < not >
2  < o1.type :: all-Instances() -> forAll( o1 | . . .
3          < oi.type :: all-Instances() -> forAll( oi | . . .
4          conditions ( o1, . . . oi )
5          implies > ?
6                  oj.type :: all-Intances() -> exists( oj | . . .
7                  ok.type :: all-Intances() -> exists( ok | . . .
8                  conditions( o1, . . . , oi, oj, . . . , ok )
9                  < and not
10                         ol.type :: all-Instances() -> exists ( ol | . . .
11                         om.type :: all-Instances() -> exists ( om | . . .
12                 conditions (o1, . . . , oi, oj, . . . , ok, ol, . . . , om) > * . . . )
```

List-3: Outline for pre-conditions

In List-4, it demonstrates the OCL expression created from the prerequisite as represented in Figure-3. A "OneStartEvent" as in line-1, Multiple "StartEvents" as in line 3 to 5, and "PathsForFlow" on lines 7 to 13.

Figure. 3: Some preconditions of the transformation

```
1  StartEvent :: all-Instances -> exists (e | e.outgoing -> size() = 1)
2
3  not StartEvent :: all-Instances() -> exists (e1 |
4          StartEvent :: all-Instances -> exists( e2 |
5                  e1 <> e2))
6
7          Gateway :: all-Instances() -> forAll (g |
8          true
9          Implies
10                 SequencesFlow :: all-Instances() -> exists (s1 |
11                 SequenceFlow :: all-Instances() -> exists ( s2 |
12                 s1.targetRef = g and s2.sourceRef = g
13                 and s1 <> s2 )))
```

List-4: Phrases for pre-conditions in Figure. 3

It does not change the specification and post-conditions; we have created a series of claims that will serve as a test suite.

B. Coverage criteria for input model creation

It is a two-step process for model creation. First, identify the characteristics of each type of input

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

should be created for this type of an OCL expression. The criteria are based on the definitions of the scope of this specification. It is then formed with the input meta-model and the OCL constraint solver is an expression of the code provides a prerequisite. It tries a position to find the right model input enough to give expression OCL needs, and requirements metamodel integrity. For some phrases, it doesn't locate the model in the range specified. In this case, we may intend to expand the search or not to create the models for individual expression.

We categorize few levels of specification coverage for the created test set are increases in the range of exhaustively as, properties, close properties, and t-way closed. Attributes and t-way levels combine source models to create models that can test multiple invariants of specifications. The remaining levels create a model that does not involve the occurrence of a specific invariant.

- *Property coverage*: This is the most comprehensive coverage applicable when the specification's invariant is independent.

- *Closed property coverage*: This criterion is not included in the description of the appearance of some of the sources of change that extend the previous model by creating additional patterns. Interestingly, the change is consistent with the unchanging words of blank samples do not have to be original. Source model shows the result of some type but its absence does not change any result. However, in this way is even more interesting in the input model. The aim of the transformation is not a valid because the rest of the models and variables and post-conditions to yield the integrity of the sanctions should target metamodel.

- *t-way coverage*: Most errors are due to a number of factors or characteristics of software systems. These observations and t-wise testing [25] is a test case creation system which includes all of the possible combinations of t-attributes.

C. Apply of Constraint Solver to locate Models satisfaction

Some expressions created at the coverage level mentioned above may not be satisfactory, and if many combinations are not satisfactory for example, using a thorough strategy, the constraint solver consumes a lot of unnecessary runtimes. For this purpose, one general expression is formulated that covers how many invariants should be considered, and where some of them may be invalidated but not precisely specified. If the condition solver locates satisfactory tasks, then this tasks can infer from the constraint not only the model but also the invariant constructions considered in the invariant.

The solutions already found in the iterative execution can be found in all combinations or when the constraint solver is no longer satisfied until you cannot find it. It can then stop the search because it can not satisfy the other configuration. Table-1 shows a generalized representation of most of the coverage levels provided. It provides the generalized closed t-way coverage and closed combinations are moreover difficult to justify additional overhead.

Table-1 Generalised terms for the dissimilar levels of coverage

Coverage level	Generalised expression
Property	$\bigwedge_{j=1}^{n}(s_j \rightarrow i_j) \wedge vs = 1$
Closed property	$\bigwedge_{j=1}^{n}(s_j \rightarrow (p_j \leftrightarrow i_j)) \wedge vs = 1$
t-way	$\bigwedge_{j=1}^{n}(s_j \rightarrow i_j) \wedge vs = t$

Regardless of the assurance coverage we choose, the model of creation process has some configurable aspects or empirical methods that can influence the dimension and number of models created. For example, when seems for a model to test multiple invariants with non-empty intersections, we can consider different overlapping levels, from non-overlapping, as the source of invariant is segregated to maximum overlapping. Second, if the combination already exists in a previously created model for a more demanding or exhaustive test, we can skip a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

model creation for a specific grouping of properties and minimize the range of the created test set.

4. EXPERIMENT EVALUATION

The proposed framework is implemented by Eclipse EMF tool supported for the model framework. This section describes some experiments to measure the efficiency of the input model created to detect conversion failures. Our goal is to identify coverage criteria that can detect a large computation mistakes with the little attempt, that is with a minute set of test data. In this experiment, we focused on two types of coverage such as property and 2-way. These are the effortless coverage criterion and accordingly, the most precious and smallest test dataset creation time. So it is interesting to know if the test suite created by even the simplest criteria can detect a large number of errors.

4.1 Setup

The test bed for the experiment is a series of "ATL model" conversions that consist of 120 lines of code conversion executed for the executing examples as well as presenting the variations of the ATL variants repository. Converts a class schema model into a relational model from the 107 line of database model and another from BibTeX are converted to an XML-based format for DocBook having a 261 line of codes [31].

Table-3: Input meta-models and its Quantity of specifications in the tested

	Specification			Input meta-model		
	#pre.	#inv.	#pos.	#classes	#assoc.	#inh.
(a)	3	10	1	6	4	5
(b)	12	10	5	18	9	15
(c)	3	18	4	21	1	31

(a) Class-to-Relational, (b) BPMN-to-Petri nets, (c) BibTeX-to-DocBook

The specification of the executing illustration was written ahead of the implementation, but in the other case, the specification was written in the documentation provided in the ATL variants repository after implementation. This document contains a very detailed description of the natural language conversion rules that encode the pattern. Since we did not add anything that is not in the documentation to the statement, the completeness of the specification for the transformation at the ATL variants repository depends on the documentation available.

Table-3 collects the number of pre-conditions, post-conditions, and invariants of the final specification and the size of the input metamodel for every one case. The "BPMN metamodel" is the mainly difficult in expressions of the number of relationships among classes, but the class diagram meta model is effortless of the three with little relevance to the "BibTeX metamodel", but it uses inheritance profoundly.

4.2. Results

In the specification, we derived a set of tests for each scope type in the properties and 2-way. This experiment was performed on Intel Core i3 having 6 Gb RAM. The number displayed near the graphic line in related to the number of models created from the numeral models searched for a number of appearances, the solver does not find the explanation model in the specified range. In general, the more invariants the more models will be created, resulting in a longer time for creation. Nevertheless, the size of the input meta-model and the number of pre-conditions to be specified to have a great impact on the model search. That is why the time "BibTeX-DocBook" specification is shorter than the time of "BPMN-to-Petri". Even more than the previous specification invariants, input metamodel describes the size of more models is created. In the first case, a number of pre-conditions to solve each model is lesser than the each model search.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

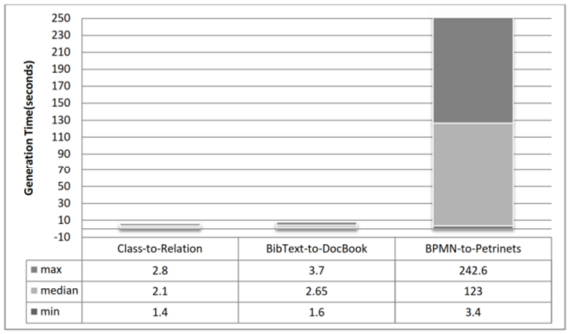

Figure. 5: Model Creation Time For Property Coverage

Figure. 5 illustrates the median of the period it obtained for the tool to locate every tested model utilizing the attribute scope. We utilized metric values as a metric to reduce the consequence of anomalies. This median is 2.1 seconds for the Class-to-Relational specification, 2.65 seconds for the "BibTeX-DocBook", and 123 seconds for the "BPMN-Petri" network. As mentioned earlier, the variation in model creation time is mainly the size of the input metamodel and the number of prerequisites for the specification. For "BPMN-to-Petri", the highest creation period was 242.6 seconds at a time, 8 models were created in less than 9 s, and 5 models were created in 18 s.

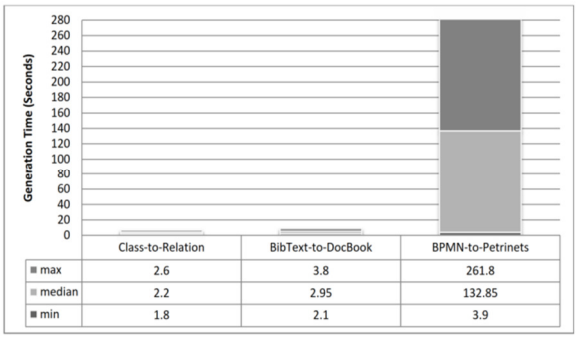

Figure. 6. Model Creation Time For 2-Way Coverage

The graph of Figure. 6 illustrates the similar metric, but for "2-way coverage" model, the locator must believe an invariant pair. The differentiation in creation period with respect to attribute coverage can be ignored in the "Class-to-Relational" and "BibTeX-to-DocBook" specifications. In compare, the median value in our running example is more than the 2-way coverage that is 132.85 seconds in compared to 123.85 seconds. This indicates more model creation time having more models with more creation time. Because the results are similar, it does not display the closed variants of the real estate and the creation time for 2-ways coverage criteria.

Models created using our technique can detect unintentional errors in the transformation implementation, so it is useful even if you use the least comprehensive scope. It is also easy to create and can satisfy additional prerequisites conditions, that is if the model obeys the rules to the meta model and is required for conversion. In this regard, automatic model creation also facilitated to recognize the prerequisites of the input model. The input model is more accurate by checking the function more efficiently in software development.

5. CONCLUSION

The proposed design model validation aims to test the intent of the transformation and allows the created model to test the transformation properties of interest. In addition, models created with our technology to be liable to be smaller. This has the improvement that the analysis model is kept intentionally. This test model is created to test a specific combination of strain invariants to be examined more efficiently by the function. We performed several experiments to automatically create test suites from different variation standards based on various specification-based coverage standards and then measure the efficiency of the created tests.

In the future, we target to conduct more research with larger case studies to assess whether transformation scores are higher for test sets created utilizing dissimilar coverage criteria in these cases. Beginning with the results of this experiment, we can outline to incorporate additional technologies for metamodel-based and white-box-based input model creation. Also one can implement the mechanisms to detect and remove duplicate models from the created test set in an enhanced works.

REFERENCES

[1]. L. Burgueno, J. Troya, M. Wimmer, and A. Vallecillo, "Static Fault Localization in Model Transformations", IEEE Transactions On Software Engineering, Vol. 41, No. 5, May 2015.

[2]. J. S. Cuadrado, E. Guerra, and J. de Lara, "A Component Model for Model Transformations", IEEE Transactions On

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Software Engineering, Vol. 40, No. 11, November 2014.

[3]. M. Schur, A.S. Roth, and A. Zeller, "Mining Workflow Models from Web Applications", IEEE Transactions On Software Engineering, Vol. 41, No. 12, December 2015.

[4]. Claudio Dias-Neto and G. H. Travassos, "Supporting the Combined Selection of Model-Based Testing Techniques", IEEE Transactions On Software Engineering, Vol. 40, No. 10, October 2014.

[5]. F. Belli and M. Beyazit, "Exploiting Model Morphology for Event-Based Testing", IEEE Transactions On Software Engineering, Vol. 41, No. 2, February 2015.

[6]. Nathan Weston, Ruzanna Chitchyan, Awais Rashid, "Formal semantic conflict detection in aspect-oriented requirements", Requirements Engineering, 14(4): 247-268, 2009.

[7]. Felix C., Juan C. Duenas, and R. Garcia-Carmona, "An Autonomous Engine for Services Configuration and Deployment", IEEE Transactions On Software Engineering, Vol. 38, No. 3, May/June 2012

[8]. Joao Araujo, Jon Whittle, "Modeling and composing scenario-based requirements with aspects", Proceedings of the 12th IEEE International Requirements Engineering Conference, Washington, 122-131, 2004.

[9]. Esther Guerra, Mathias Soeken, "Specification-driven model transformation testing", Springer softw Syst Model, 14:623–644, DOI 10.1007/s10270-013-0369-x, 2015.

[10]. Chi-Lun Liu, "Ontology-Based Requirements Conflicts Analysis in Activity Diagrams", Proceedings of the International Conference on Computational Science and Its Applications.Berlin: Springer- Verlag, 2009.

[11]. Jianmei Guo, Yinglin Wang, Pablo Trinidad, David Benavides, "Consistency Maintenance for Evolving Feature Models", Expert Systems with Applications, 39(5): 4987-4998, 2012.

[12]. Ying Jin, Huaxiao Liu,Peng Zhang, "An approach to analysing and verifying aspect-oriented requirements model", Chinese Journal of Computers, 36(1).63-73, 2013.

[13]. T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin, "Advanced Concepts and tools for In-Place EMF Model Transformation", In Model Driven Engineering Languages and Systems, 13th International Conference, MoDELS 2010. Proceedings, LNCS, Springer, pp. 121–135, 2010.

[14]. T. Mens and Pieter Van Gorp, "A Taxonomy of Model Transformation", Electronic Notes in Theoretical Computer Science, DOI:10.1016/j.entcs.2005.10.021, Elsevier, 2006.

[15]. J.Cabot, R.Clarisó, E.Guerra, J.de Lara, "Verification and validation of declarative model-to-model transformations through invariants", Journal of System Software 83(2), 283–302, 2010.

[16]. Z. Javed, P. A. Strooper and G. N. Watson, "Automated generation of test cases using modeldriven architecture", In Proc. of the ICSE 2nd International Workshop on Automation of Software Test (AST), 2007.

[17]. Tian, Z. Duan, and Z. Duan, "Making CEGAR More Efficient in Software Model Checking", IEEE Transactions On Software Engineering, Vol. 40, No. 12, December 2014.

[18]. B.Baudry, T.Dinh-trong, J.-M.Mottu, D.Simmonds, R.France, S.Ghosh, F.Fleurey, Y.Le Traon, "Model transformation testing challenges", In ECMDA Workshop on Integration of Model Driven Development and Model Driven Testing, 2006.

[19]. Nebut, F. Fleurey, Y. Le Traon, and J. Jezequel, "Automatic Test Generation: A Use Case Driven Approach", IEEE Transactions On Software Engineering, Vol. 32, No. 3, March 2006.

[20]. W. Leungwattanakit, C. Artho, M. Hagiya, Y. Tanabe, M. Yamamoto, and K. Takahashi, "Modular Software Model Checking for Distributed Systems", IEEE Transactions On Software Engineering, Vol. 40, No. 5, May 2014

[21]. M. Fowler, K. Scott, "UML Distilled: A Brief Guide to the Standard Object Modeling Language", Addison-Wesley. 1999.

[22]. Soeken, M., Wille, R., Kuhlmann, M., Gogolla, M., Drechsler, R. "Verifying UML/OCL models using boolean satisfiability, In: IEEE on DATE'10, pp. 1341–1344,2010.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[23]. S.Sen, J.-M.Mottu, M.Tisi, J. Cabot, "Using models of partial knowledge to test model transformations", In: ICMT'12, vol. 7307 of LNCS, pp. 24–39. Springer, Berlin, 2012.

[24]. J.A.M. Quillan, J.F.Power, "White-box coverage criteria for model transformations", In 1st International Workshop on Model Transformation with ATL, 2009.

[25]. S.Oster, , I.Zorcic, , F.Markert, , M.Lochau, "MoSo-PoLiTe: tool support for pairwise and model-based software product line testing", In: VaMoS'11, ACM International Conference Proceedings Series, pp. 79–82. ACM, 2011.

[26]. Y.Lin, J.Zhang, J.Gray, "A framework for testing model transformations", In Model-driven Software Development—Research and Practice in Software Engineering. Springer, Berlin, 2005.

[27]. F.Jouault, F.Allilaire, J.Bézivin, I.Kurtev, "ATL: a model transformation tool", Science Comput. Program. 72(1–2), 31–39 ,2008.

[28]. D.S.Kolovos, R.F.Paige, F.Polack, "The epsilon transformation language", In ICMT 08, vol. 5063 of LNCS, pp. 46–60. Springer, Berlin, 2008.

[29]. R.Lämmel, W.Schulte, "Controllable combinatorial coverage in grammar-based testing", In TestCom'06, pp. 19–38, 2006.

[30]. E.Guerra, , J.de Lara, , D.Kolovos, , R.Paige, , O.dos Santos, "Engineering model transformations with transML", Software Syst. Model. 12(3), 555–577, 2013.

[31]. ATL Transformations Source, "http://www.eclipse.org/atl/atlTransformations/".