Searching for Models to Evaluate Software Technology

Francisco Gomes de Oliveira Neto[1]
Software Practices Laboratory
Universidade Federal Campina Grande
Campina Grande, Brazil
netojin@copin.ufcg.edu.br

Robert Feldt, Richard Torkar
Dept. of Computer Science and Engineering
Chalmers and Gothenburg University
Gothenburg, Sweden
{robert.feldt,richard.torkar}@chalmers.se

Patrícia D. L. Machado[1]
Software Practices Laboratory
Universidade Federal Campina Grande
Campina Grande, Brazil
patricia@computacao.ufcg.edu.br

***Abstract*—Modeling and abstraction is key in all engineering processes and have found extensive use also in software engineering. When developing new methodologies and techniques to support software engineers we want to evaluate them on realistic models. However, this is a challenge since (1) it is hard to get industry to give access to their models, and (2) we need a large number of models to systematically evaluate a technology. This paper proposes that search-based techniques can be used to search for models with desirable properties, which can then be used to systematically evaluate model-based technologies. By targeting properties seen in industrial models we can then get the best of both worlds: models that are similar to models used in industry but in quantities that allow extensive experimentation. To exemplify our ideas we consider a specific case in which a model generator is used to create models to test a regression test optimization technique.**

***Index Terms*—Automatic Model Generation, Search-Based Techniques, Model-based Software Engineering Technology.**

I. Introduction

Models are a central concept in any engineering discipline and have found many uses also in software engineering [12]. Models can be used on multiple abstraction levels and in several stages of software development from project, requirements to testing. Besides providing valuable information regarding the product being developed, in many cases they help automate some of the development activities; in fact this is often a driving factor in developing models in the first place.

Several studies have described benefits of using model-based (MB) techniques [13]. However, there are also drawbacks and in particular the area has been criticized for a general lack of empirical evaluation. For example, a systematic review on model-based testing concluded that there was a general lack of empirical evaluation and transfer to industrial use [4]. There are many reasons for why this is the case. In this paper we focus on what we think is a key reason: even if we want and try to perform systematic, empirical evaluation we often cannot find enough number of realistic models to allow conclusive results. Basically, for models to be realistic they should have a size, type and other characteristics that are typical of the models used in organisations developing software. However, these organisations are often reluctant to share their models; the models are a key part of the economic value and competitive advantage of these organisation. And even if a few organisations would give models for research, the number of models so gained might not be numerous enough or diverse enough to allow a systematic and statistically valid evaluation of model-based software technologies.

To address this situation we propose to combine search-based techniques and stochastic model generation with statistical summaries of realistic models. Statistical summaries of actual models used in the software industry allows an effective form of anonymization; most companies would allow us to run scripts on their models in-house and just export the descriptive statistics. Combining this information with stochastic model generators would allow us to generate large number of models that share characteristics with industrial models. However, by adding search and optimization to this mix we can also ensure diverse models, for example finding models that are easy or hard for the technology being evaluated. In general, search-based techniques would allow exploration and visualisation of different spaces of models as well as the difficulty for the technique over this space [6], [7].

To illustrate our ideas, we use a model generator to evaluate a model-based regression test (MBRT) selection technique. Our generated models represents a generic reactive deterministic software, where the system reacts to inputs by showing expected output in a state-based format. After modifying the models we can use them to generate test cases and execute the selection technique. The goal is to explore how the models' characteristics affect the performance of the technique, thus pointing to areas of improvement based on a deeper understanding of both strengths and limitations.

In the following, we present related work on model generation (Section II) and then present an example case of model generation for evaluating a MBRT optimization technique (III). Section IV explores our core idea and present three different types of *search-based model-generation for tech evaluation* (SBMTE) and explores their use on our example case. Finally, Section V concludes.

II. Background and Related Work

Models allow the representation of external, internal, behavioral and structural interactions, improving the understanding of the software. We can use MB techniques to automatically harness information from models for a specific purpose (e.g. test case generation, fault detection and risk analysis). Therefore, it is important to have a well specified model. The quality of the model depends on several variables such as the format of the model used (UML Diagrams, State Machines, Control Flow Graphs, among others) or the expertise of the people responsible for building the model [5], thus, modeling

[1]This work is supported by CNPq grants 484643/2011-8 and 573964/2008-4. First author is also supported by CAPES.

consumes a lot of time and effort. In these cases, an alternative is to automatically generate or transform it from other format.

Automatic model generation is discussed in different fields of software research. Feng et al. [9] generate models for testing of real-time embedded systems, where the specification provides better visualization, thus facilitating its integration and analysis. In their work, a machine learning approach generates a control flow from execution traces. Huselius et al. present similar work, targeting legacy systems [11].

In both studies, the generation relies on information of traces of execution from a program. The goal is to obtain information about the system from which the model is being generated, such as prototyping of future design, or retrieve data dependencies. In our approach, we target generation of models to execute MB techniques.

Deeptimahanti and Sanyal [3] propose a semi-automatic generation of UML models from natural language specifications through natural language processing techniques. The authors provide tool support (UMGAR) that generates use-case diagrams, analysis class model, collaboration diagram, and design class model. The use case templates used by the authors provides a level of abstraction similar to the models that our strategy generates. However, the generation still depends on human interaction to remove irrelevant classes in diagrams. Also, the generation is for specifications described in natural language (sentences with subjects, verbs, etc.) hampering instrumentation when generation of a large quantity of models is required.

In the model-driven engineering (MDE) field, model transformations need to be tested, and models have to be generated to enable the testing of these transformations. In this context, models are generated from meta-models or Object Constraint Language (OCL) expressions determining possible inputs and outputs of the target model [10], [14]. In these studies, modeling or analysis is required prior to generation, focused on the transformation to be tested.

On the other hand, our goal is to enable a much simpler and yet generic model generation to allow early execution or evaluation of MB techniques in combination with search-based techniques. The inputs of our generator are values describing the models that need to be generated independent of a particular context of application. In turn, these values can be obtained, for example, from models of other projects inside the industry or reported in the literature. It is important to notice that MDE concepts and technology can also be used to support our solution.

III. Example of Generation for MB Testing

This section illustrates the use of a model generator to evaluate a technique for MBRT selection (Fig. 1). The goal is to execute the technique on several test suites and then perform statistical analysis on the results. Since numerous test suites are required to achieve statistically valid evaluation, we developed a generator to create models that, in turn, generate test cases allowing the execution of the technique.

The models are generated and modified according to a set of values (i.e. parameters) expressing characteristics of the desired type of model (e.g. size, number of branches, paths with loops and additions). By varying these values and generating different test suites we analyze the selected test cases and draw conclusions regarding the performance of the technique and the characteristics of the models being modified.

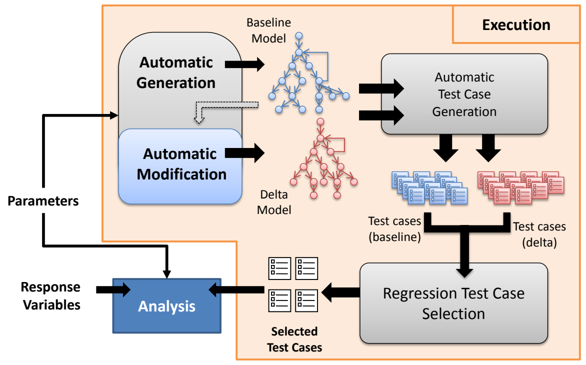

Fig. 1. Using the model generator in an MBT technique for regression test case selection.

A. Automatic Generation Strategy

One of the challenges is to choose a format of model capable of expressing real specifications, which could be constructed automatically. With that in mind, we decided to use a *Labeled Transitions System* (LTS), to generate our models, since the different paths of an LTS can represent sequences of inputs and outputs around the common and alternative flows being tested. Furthermore, they have already been widely used for automatic test case generation and selection [1], [2]. LTS is defined as a 4-tuple $S = (Q; A; T; q_0)$, with: the set of states (Q); a finite nonempty set of labels (A); the transition relation ($T \subseteq (Q \times A \times Q)$); and the initial state (q_0).

Figure 2 shows an example of an LTS representing different flows in a simple application (inputs and outputs are marked by "?" and "!", respectively). The sequences of inputs and outputs determine the behavior being modeled, since flows can branch into alternative paths and join again in common scenarios. Each path in a LTS represents a test case.

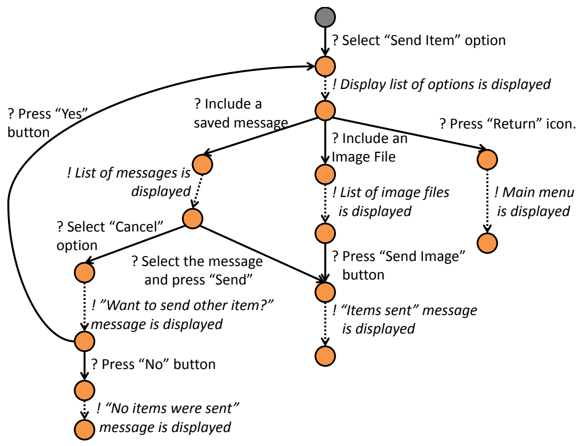
? Select "Send Item" option
! Display list of options is displayed
? Press "Yes" button
? Include a saved message
? Include an Image File
? Press "Return" icon.
! List of messages is displayed
! List of image files is displayed
! Main menu is displayed
? Select "Cancel" option
? Select the message and press "Send"
? Press "Send Image" button
! "Want to send other item?" message is displayed
! "Items sent" message is displayed
? Press "No" button
! "No items were sent" message is displayed

Fig. 2. Example of an LTS as a specification model.

LTSs provide a powerful level of abstraction, and have simple structures capable of expressing a variety of applications.

Some tools use LTS as an input format, and it can be transformed from other model formats (e.g. sequence diagrams) [1]. The states and transition also provide an intuitive idea about the scenarios that can be executed, allowing an easy understanding of what is being specified.

B. *Parameters and Structures*

The transitions in the LTS represent the execution of a step of the software going from one location to another. From now on we will refer to locations as states of the LTS[2]. The parameters for generation are the quantity, size[3] and structures of the LTS.

Firstly, a sequence in the specified size is generated (i.e. the main flow), then paths with loops, branches and joins (Fig. 3 (a), (b) and (c) respectively) are added in arbitrary locations of the LTS. That process continues until the specified number of LTS are generated. In the end, transitions are carefully added and removed from the generated model to obtain different versions and perform regression test case selection. With that idea it is possible to automatically generate several different LTS graphs. Despite being different, the parameters provide some level of control allowing the generation of similar LTS.

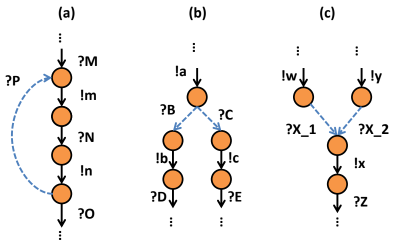

Fig. 3. Different structures used to generate an LTS. (a) Path with loops; (b) Branch or alternative flows; (c) Joins or common flows;

C. *About the Generation*

Models such as the ones we generate can be used for statistical analysis, since the parameters can be changed obtaining different results. An example here is to investigate the scalability of a technique's performance as the size of the model grows. The parameters can be adjusted to generate bigger and more complex models allowing for the investigation of how the technique behaves with such models. The results can then be used to improve the technique.

The generation can also be based on parameters obtained from real models, e.g. analyzing models from a repository. This allows the generation to be targeted to a specific type of model, but using inappropriate values can affect the representativeness of the models being generated. On the other hand, many models have common entities that need to comply with meta-models, for example, UML use case or sequence diagrams. These common entities can then be used as parameters for comparison of different models generated.

[2]The state in a LTS differs from the idea of a state in a Finite State Machine because it does not represent a system state (e.g. "Running" or "Waiting for input"), instead it represents a state of the execution flow.

[3]In this work we consider the size to be the number of transitions in the main flow specified. For example, a size 4 will yield an LTS with 4 inputs and 4 outputs in the main flow.

Also, general metrics from graph theory can be used to explain properties of the models. An example is the use of eccentricity and other distance measures in social networks, or clustering of states and transitions to compare models and identify modifications. This provides different scenarios where our generator can be integrated and used. Currently, the example and generator described in this work are currently implemented in the LTS-BT tool [2].

IV. Search-Based Model Generation for Technology Evaluation (SBMTE)

We propose a simple, 2-dimensional model of approaches to evaluating model-based software engineering technology, as shown in Fig. 4. The approaches vary on two main dimensions: realism of models (along vertical axis in the figure, note that here only two approaches, both of high realism, are actually shown), type of search employed (varies from left to right on horizontal axis, left-most and right-most shown). Note that all of our approaches assume the existence of a stochastic model generator that can generate models of the right type given values for or distributions over model parameters[4]. In the example in Section III, the model parameters are the number of states, joins, branches and paths with loops.

The realism dimension is used to denote variation along a continuum of different levels of realism. Only the most realistic level is shown in the figure. At this level *model property extraction* is used to get summary and descriptive statistics about models actually used in industry. This might be easier for some aspects of models (nodes, connections etc.) than others (constraints and semantic information in the models, for example). We propose that the researchers construct extraction scripts that industrial partners can apply on their models and send back characteristic model parameters (*script-based property extraction*). For lower levels of realism the researchers have basic descriptive statistics about model parameters only from a few companies or guess them from existing literature relevant for the model types being studied. At the lowest level of realism the researcher has no information about characteristic models; however, such *zero realism SBMTE* can still give valid technology evaluation if combined with search.

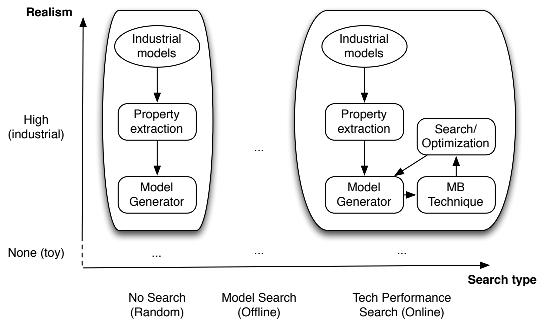

Fig. 4. Two-dimensional model of approaches for SBMTE.

[4]This restriction is not essential though, since a simpler model generator can be combined with search and thus act as a more adept generator.

For the search-type dimension we propose that at least three different levels should be distinguished. At the basic level no optimizing search is used, essentially this amounts to a random search given the model generator. Without realistic model parameters this could be used to evaluate our MBRT technique by generating many small, often called toy, models ensuring the technique is feasible. By getting realistic model parameters from industrial partners a 'random search SBMTE' could be used to collect convincing evidence that a technology would scale to and work for actual industrial systems. However, by going to the next level of search we could search for models with specific combinations of parameters that may not be directly available among the industrial models from which we extracted typical properties. Basically the search would allow us to explore the space of possible models around the region for which we have (high realism) or have not (low realism) information about typical models. As noted by Feldt in other studies, interactive visualisation of high-dimensional data is likely to be key in making this exploration useful [7].

The next level of SBMTE would use performance measures of applying the technique itself on the generated models to guide the search in the model space. This 'online SBMTE' would allow specific models and areas of the model space to be found for which the technique performs well or badly, or have shorter/longer execution times. This could be essential in allowing the proper evaluation of a MB technique and allow visual comparison and trade-off between different MB techniques. However, since many MB techniques are time consuming and the technique is 'in the loop' of the search it is likely that this type of SBMTE would be costly. Thus it is meaningful to distinguish it from model-focused SBMTE where only characteristics about the models themselves are used in the search. We propose to call this 'offline SBMTE'.

For the MBT example in Section III we initially only had a 'low realism', 'no search' approach to evaluation, i.e. in the lower, left part of our SBMTE model in Fig. 4. By developing our SBMTE model we clarified our thinking and are currently exploring improvements to our MBT technology evaluation along both dimensions. We have secured access to realistic industrial models from which we will extract salient properties. But we are also integrating the model generator with a search-based optimization framework, built on Differential Evolution (DE) and Particle Swarm Optimization (PSO), so we can do offline SBMTE. Once we have done high-realism, offline SBMTE we will be able to consider online SBMTE of the MBT. However, alternative solutions might give similar benefits, for example fractional factorial designs of experiments in combination with offline SBMTE might give enough evaluation power, much as it has alleviated the need for meta-optimization in other types of research [8].

V. Conclusion and Future Work

This paper discusses the idea of combining automatic model generation and search-based strategies for investigating MB software technology. We showed how the generator can be used with a MB testing technique, and then discussed the evaluation of MB technologies considering the realism of the models generated, and the type of search employed. Our thinking about different approaches to combine these technologies lead to a clarifying framework that has helped us better structure and plan our ongoing work. We believe that this SBMTE framework, with its two dimensions of realism and search type, has general value and can be useful for other researchers evaluating MB software technology.

Even though we exemplified our model on a specific type of model, labeled transition systems, for the evaluation of a specific MB technique, regression test case selection, the SBMTE framework is not limited to any particular model of technique. Different parameters to LTS models or wholly different models can be investigated (e.g. activity diagrams and use case diagrams). Modifications of specification models can also benefit from our approach, since SBMTE can search among the space of models to determine, for example, possible changes that can be performed, or changes that have specific characteristics. In the software engineering literature, MB techniques have been proposed for several purposes (e.g. specification, architecture and testing) and for different levels of the software (e.g. unit, integration and system) yielding a variety of combinations where our approach can be explored.

References

[1] E. G. Cartaxo, W. A. Andrade, F. G. Oliveira Neto, and P. D. L. Machado. Test case generation by means of uml sequence diagrams and labeled transition systems. In *ISIC. IEEE International Conference on Systems, Man and Cybernetics, 2007*, pages 1292–1297, 2007.

[2] E. G. Cartaxo, W. A. Andrade, F. G. Oliveira Neto, and P. D. L. Machado. LTS-BT: a tool to generate and select functional test cases for embedded systems. In *SAC '08: Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 1540–1544, New York, NY, USA, 2008. ACM.

[3] D. K. Deeptimahanti and R. Sanyal. Semi-automatic generation of uml models from natural language requirements. In *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, pages 165–174, New York, NY, USA, 2011. ACM.

[4] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos. A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies*, pages 31–36. ACM, 2007.

[5] I. K. El-Far. Enjoying the perks of model-based testing. In *In Proceedings of the Software Testing, Analysis, and Review Conference*, 2001.

[6] R. Feldt. Generating diverse software versions with genetic programming: an experimental study. *Software, IEE Proceedings-*, 145(6):228–236, 1998.

[7] R. Feldt. Genetic programming as an explorative tool in-early software development phases. *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering*, pages 11–19, 1999.

[8] Robert Feldt and Peter Nordin. Using factorial experiments to evaluate the effect of genetic programming parameters. volume 1802 of *Lecture Notes on Computer Science*, pages 271–282, Edinburgh, 2000. Springer-Verlag.

[9] T. H. Feng, L. Wang, W. Zheng, S. Kanajan, and S.A. Seshia. Automatic model generation for black box real-time systems. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, april 2007.

[10] E. Guerra. Specification-driven test generation for model transformations. In *Theory and Practice of Model Transformations*, volume 7307 of *Lecture Notes in Computer Science*, pages 40–55. Springer Berlin Heidelberg, 2012.

[11] J. Huselius, J. Andersson, H. Hansson, and S. Punnekkat. Automatic generation and validation of models of legacy software. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings.*, pages 342 –349, 2006.

[12] J. Ludewig. Models in software engineering–an introduction. *Software and Systems Modeling*, 2(1):5–14, 2003.

[13] B. Selic. The pragmatics of model-driven development. *Software, IEEE*, 20(5):19–25, 2003.

[14] S. Sen, B. Baudry, and J. M. Mottu. Automatic model generation strategies for model transformation testing. In *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*, ICMT '09, pages 148–164, Berlin, Heidelberg, 2009. Springer-Verlag.