Towards a Cognizant Virtual Software Modeling Assistant using Model Clones

Matthew Stephan
Department of Computer Science & Software Engineering, Miami University, Oxford, Ohio, USA
stephamd@miamioh.edu

***Abstract*—We present our new ideas on taking the first steps towards cultivating synergy between model-driven engineering (MDE), machine learning, and software clones. Specifically, we describe our vision in realizing a cognizant virtual software modeling assistant that uses the latter two to improve software design and MDE. Software engineering has benefited greatly from knowledge-based cognizant source code completion and assistance, but MDE has few and limited analogous capabilities. We outline our research directions by describing our vision for a prototype assistant that provides suggestions to modelers performing model creation or extension in the form of 1) complete models for insertion or guidance, and 2) granular single-step operations. These suggestions are derived by detecting clones of the in-progress model and existing domain, organizational, and exemplar models. We overview our envisioned workflow between modeler and assistant, and, using Simulink as an example, illustrate different manifestations including multiple overlays with percentages and employing variant elements.**

***Index Terms*—model driven engineering, model clones, model clone detection, machine learning, software modeling**

I. Introduction

Model-driven engineering (MDE) is a software development approach employing high level formal abstractions as first class artifacts [1]. There is notable adoption of MDE in industry [2], the research community, and education. Due to a variety of social, organizational, and technical factors, there is still much room for innovation and increased model quality [2], despite the demonstrable benefits and success stories of those employing MDE.

Cognification is one way that has enhanced software development. For example, machine learning (ML) techniques are used by software engineers employing traditional code-based approaches to estimate development effort [3] and assist with coding [4]. Analogous techniques have yet to be significantly exploited in MDE. As pointed out at the "2017 Grand Challenges in Modelling" workshop, this inadequacy in the state-of-the-art must be addressed by the software community in order to support adoption and evolution of MDE [5]. One aspect of cognification we believe to be especially promising and valuable is the notion of a modeling bot, which can act as a virtual software modeling assistant. By comparing developers' incomplete models to similar domain models and exemplars, it can assist them by identifying inconsistencies and recommending additions or changes [5].

In this paper, we describe our vision for this research direction by describing our approach involving a virtual assistant that analyzes an incomplete model/system undergoing development by looking for similar models in the form of type 3, near-miss, clones [6] from the same project/organization, same domain, and best practice exemplars, and provides suggestions based on those models. We establish the impact and value of this innovation by contrasting it with current practice, and then present our new ideas involving how to achieve synergy between MDE, ML, and software clones.

This material is based upon work supported by the National Science Foundation under Grant No. 1849632 .

II. Related Work

Techniques for assisting with traditional source code development have been successfully implemented by research and industry alike. Proksch et al. [4], Raychev et al. [7], and Asaduzzaman et al. [8] are notable examples that use Bayesian networks, language-based techniques, and method call and call site contexts, respectively. A popular example of code completion is Eclipse's code recommender system, which suggests code based on API usage statistics. All of these are successful examples to draw inspiration from for our vision.

We discuss the state-of-the-art in model assistance to position our ideas, and demonstrate their pertinence and potential impact. Dyck et al. provided suggestions on how a UML model recommender should appear based on a UML tools survey [9]. Segura et al. created an assistant for modeling language meta model development by querying textual data, such as XML and RDF [10], which is different than our task of model development, as is the data being queried. Sen et al. [11], Mazanek et al. [12], Pati et al. [13], and Steinmann and Ulke [14] all employ meta model modeling language/grammar analysis to compute suggested potential model modifications. All of this model-completion research differs from our directions and approach in that our virtual modeling assistant makes suggestions based on knowledge derived from existing models. This includes the work most related to ours, providing auto-completion suggestions based on UML activity patterns [15]. It uses past editing operations to identify completion opportunities, whereas our approach identifies opportunities based on structurally similar models.

The most closely related concept in industry is the 2018a version of Matlab Simulink, which contains a new feature that performs proprietary model clone detection to facilitate reuse and library replacement[1]. While this method also employs model clone detection, it is explicitly for refactoring *completed* Simulink models. Our envisioned technique is intended for

[1] https://www.mathworks.com/help/slcheck/ug/identify-subsystem-clones-and-replace-them-with-library-blocks.html

use in model creation and editing *incomplete* models (model fragments), which is a different activity. Additionally, their similarity detection algorithm is both proprietary and is limited to having ”different block parameter values” only. Thus, many Type 3 clones [6], such as those having additional blocks or models using different types of blocks, will not be identified using their built-in tool.

III. Research Vision

To describe our new research idea and direction at a high level, we overview our envisioned approach by describing the general work flow of a modeler in conjunction with the process followed by a virtual software modeling assistant using model clone detection. We summarize this in Figure 1. The process begins with a modeler creating a new model or working on an existing incomplete model. They either explicitly ask the virtual modeling assistant for suggestions or, ideally, the suggestions appear in their modeling environment automatically and relatively unobtrusively. The virtual modeling assistant analyzes the incomplete model, X, and attempts to find similar models, Y, based on using near-miss model clone detection on its knowledge base. The knowledge base is formed from models from the same project and organization, the same domain, and best practice exemplars. An important consideration and challenge is questioning where the models for the latter two of these sources will come from and if they will be sufficient. With the increasing prevalence of MDE [2], we believe model repositories, such as MDEForge [16] and the Lindholmen Dataset [17], company internal repositories, and popular open source repositories that contain models, such as GitHub and SourceForge, will continue to cultivate a rise in available model artifacts. Once the modeling assistant determines the set of similar models, Y, it analyzes Y to determine what suggestions to provide the modeler. Based on inspiration from analogous code completion research [4], [18], we will base this on prevalence statistics, user rankings/input to help avoid perpetuating poor modeling practices, and context analysis. Lastly, the modeling assistant presents suggestions in a form conducive to the modeler. We have two different ideas for the nature of these suggestions which is the focus of the next section: having an virtual modeling assistant suggest 1) a complete model, for guidance or direct insertion; or 2) granular, single operation, suggestions.

A. Technical Details and Prototype

To illustrate our vision and the directions we envision this research taking, we outline our proposed prototype employing the synergy among MDE, ML, and software clones: the Simulink Virtual Modeling Assistant (SimVMA). SimVMA will guide Simulink developers in their native Simulink environment as they create and extend their models. We focus on Simulink as it has the most mature techniques for model clone detection [19] and is quite popular and growing in automotive, aerospace, and many other embedded domains. Additionally, modelers often create their Simulink models in a modular and structured fashion. However, the framework and technologies we develop in this work will not be Simulink-specific and should be generalizable to other model types. We specifically plan on using the Simone model clone detector [6] developed by Alalfi et al. According to evaluations [20], Simone is 1) the most adept Simulink clone detector for detecting type 3 near-miss clones, which are similar models according to a specific similarity definition, and 2) capable of handling large model collections very efficiently. Simone measures model similarity accurately, and is able to tackle various model comparison challenges including similar elements with different names [19]. To realize our vision initially, we will define and create different and independent capabilities for SimVMA, which we now overview in the two sections that follow.

1) Suggesting Entire Models/Systems: One capability is having SimVMA analyze an incomplete system being worked on by a modeler, and presenting similar systems that they can import/clone, as a whole, into their workspace for editing. We demonstrate this in our mock-up in Figure 2 whereby the model on the left represents a system in progress, and, after the modeler asks SimVMA for recommendations, SimVMA presents the two model systems on the right. The modeler could then either click on one those models to auto-complete their incomplete model, or keep them displayed for inspiration. These example models are completed power window models from the Simulink demonstration set[2]. This ability to insert entire systems in place of an incomplete system is useful as it is analogous to snippet matching in source code [21]. It differs from the state-of-the-art since it will detect clone system suggestions with different block types and additional/deleted blocks (Type 3) instead of just different parameters, and can be used for model completion/extension instead of library extraction.

2) Single Operation Suggestions: This direction of our research vision involves us realizing a modeling assistant that provides step-wise suggestions to modelers, similar to Eclipse’s code recommender, that are based on ML and software clones. Specifically, SimVMA will analyze model set Y to provide user suggestions and rank them based on their prediction confidence. Using frequency as an example, 25% of systems from Y did A, 15% did B, 10% did C, and so on. A, B, and C will be some form of model modification. In ML and reasoning terms, all model clones will be considered the knowledge base, and the current model and its potential additions/edits will be the data undergoing prediction. When it comes to model completion suggestions, past research analyzing industrial and open-source Simulink model repositories and edit histories indicate suggestions will belong to one of eight modifications classes [22] identified by Stephan et al. Using Bruch’s [18] information categories to avoid common biases, we will employ edit operation frequency as the number of occurrences of a model element in similar models. Additionally, we will allow for weighted rankings of model sources in the knowledge base as a user-configuration

[2] https://www.mathworks.com/help/simulink/ug/power-window-example-case-study.html

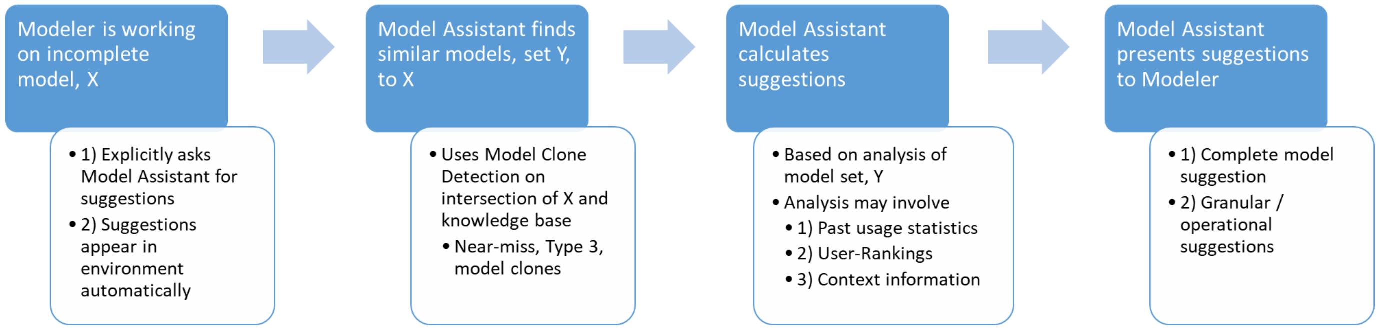
Modeler is working on incomplete model, X
1) Explicitly asks Model Assistant for suggestions
2) Suggestions appear in environment automatically
Model Assistant finds similar models, set Y, to X
Uses Model Clone Detection on intersection of X and knowledge base
Near-miss, Type 3, model clones
Model Assistant calculates suggestions
Based on analysis of model set, Y
Analysis may involve
1) Past usage statistics
2) User-Rankings
3) Context information
Model Assistant presents suggestions to Modeler
1) Complete model suggestion
2) Granular / operational suggestions

Fig. 1. Proposed Workflow of Modeler and Virtual Modeling Assistant

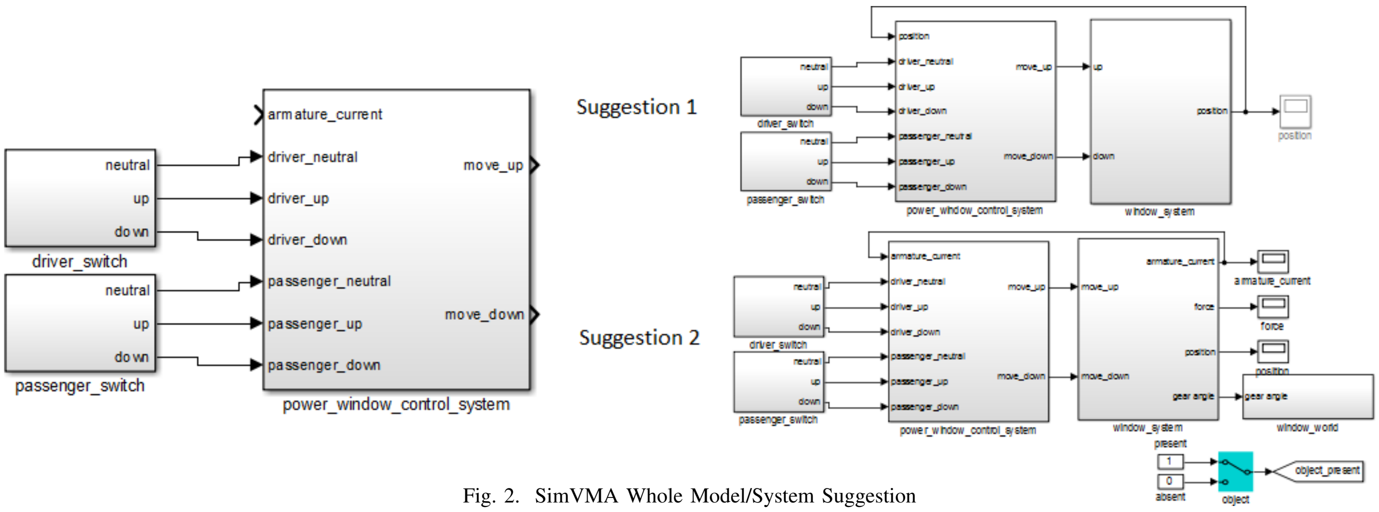

Fig. 2. SimVMA Whole Model/System Suggestion

option of the modeling assistant. For example, a modeler can specify they favour best practice exemplar model clones over project/organization model clones and model clones from the same domain. This should help avoid perpetuating poor modeling practices. We will also consider context information in our data processing and predictions. That is, our approach will assign higher weights/ranking to operations that come from model clones that have a higher similarity value (less difference) to a model than those with lower similarity values (more different), which aligns with Bruch's matching neighbour information concept [18]. This capability is useful for both model creation and existing-model editing, and can also be applied by a modeler after using our entire model insertion capability.

One avenue we envision is overlaying various suggestions directly onto the modeler's system in the Simulink modeling environment through scripts. Specifically, as pictured in our manually-created example in Figure 3, we will present different options in different colours with the percentages displayed adjacently. These percentages represent how often these elements/suggestions appeared in similar models. So, in this example, the modeler has created and connected three systems, driver_switch; passenger_switch; and the power_window_control_system, and asks SimVMA how they could proceed. SimVMA presents three annotated options through the modeling interface, sorted vertically by ranking/percentage confidence. The modeler can select one

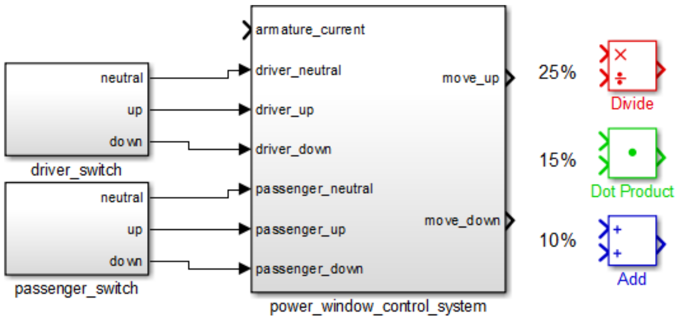

Fig. 3. SimVMA Single-Step Suggestions with Percentages

of the options, triggering SimVMA to connect the existing system to the selected option automatically. The modeler can also disregard the suggestions if they feel none of them are applicable to their needs.

Another option we foresee is using Simulink variability blocks. Continuing our example, SimVMA can add all three blocks to a Simulink Variant subsystem [3]. A Simulink Variant subsystem encapsulates multiple implementations of a subsystem, which can include a single block or more. Only one implementation is active during model execution, which is selected by a modeler programmatically (textually) or at the modeling level. As we demonstrate in Figure 4, the SimVMA suggestion will appear as a single variant block connected

[3] https://www.mathworks.com/help/simulink/examples/variant-subsystems.html

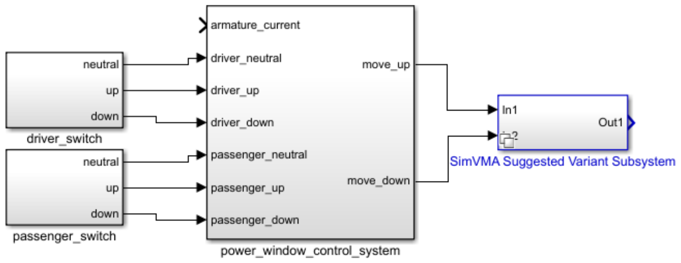

Fig. 4. SimVMA Variant Subsystem Alternative

to the incomplete model. The "SimVMA Suggested Variant Subsystem" will contain internally the three block suggestions shown in Figure 3, ranked in the same way and annotated with the same percentage confidences. One consideration for this option is the possibility of having many Variant subsystems causing an increase in complexity.

IV. Discussions and Conclusion

This research adds value and disrupts current software engineering modeling practice by providing cognitive assistance to modelers that will result in an improvement in software modeling analysis and software model quality. As per source code assistance, this quality improvement will stem from our use of instance prevalence, user ranking/input, and context analysis. Our research directions will allow modelers to leverage past experiences and established models/elements to use in their own development. By providing them this assistance and information, they can make more informed decisions, thus providing society higher quality, more secure, and reliable software. Additional specific impact stemming from this vision and research includes 1) experiments incorporating the tool into educational settings to help teach students modeling based on suggestions; 2) supporting other modeling languages and environments, like UML or the Eclipse Modeling Framework; 3) user studies and interface assessment experiments to evaluate and improve the virtual modeling assistant user experience; and 4) supporting larger scale batch operation suggestions to modelers instead of either single operation or whole-model insertion, which might help provide additional contextual information to modelers, even for single-step edits.

In this paper, we have presented our new ideas and visions for a virtual modeling assistant that employs a synergy among MDE, ML, and software clones that can help modelers create high quality software models. Specifically, our proposed approach analyzes modelers' incomplete models and finds similar ones using model clone detection on the intersection of their incomplete models and a knowledge base consisting of models from the same project/organization, same domain, and exemplars. The modeling assistant analyzes these model clones to provide either step-wise guidance or entire model examples to modelers. This differs from the state-of-the-art in that it is not based on language syntax/grammars and is meant to be used during model creation and editing, instead of during refactoring. It is our hope this paper helps accelerate future research on cognifying model-driven software engineering to improve software quality through guided software modeling. Considering our research directions are at an early-stage, we welcome and look forward to feedback from the community as we embark on making this vision a reality.

References

[1] S. Kent, "Model driven engineering," in *Integrated formal methods*. Springer, 2002, pp. 286–298.

[2] J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Science of Computer Programming*, vol. 89, pp. 144–161, 2014.

[3] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *TSE*, vol. 21, no. 2, pp. 126–137, 1995.

[4] S. Proksch, J. Lerch, and M. Mezini, "Intelligent code completion with bayesian networks," *TOSEM*, vol. 25, no. 1, p. 3, 2015.

[5] J. Cabot, R. Clarisó, M. Brambilla, and S. Gérard, "Cognifying model-driven software engineering," in *Federation of International Conferences on Software Technologies: Applications and Foundations*. Springer, 2017, pp. 154–160.

[6] M. H. Alalfi, J. R. Cordy, T. R. Dean, M. Stephan, and A. Stevenson, "Models are code too: Near-miss clone detection for simulink models," in *International Conference on Software Maintenance*, 2012, pp. 295–304.

[7] V. Raychev, M. Vechev, and E. Yahav, "Code completion with statistical language models," in *Conference on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2014, pp. 419–428.

[8] M. Asaduzzaman, C. K. Roy, K. A. Schneider, and D. Hou, "Cscc: Simple, efficient, context sensitive code completion," in *International Conference on Software Maintenance and Evolution*, 2014, pp. 71–80.

[9] A. Dyck, A. Ganser, and H. Lichter, "A framework for model recommenders requirements, architecture and tool support," in *International Conference on Model-Driven Engineering and Software Development*, 2014, pp. 282–290.

[10] Á. M. Segura, A. Pescador, J. de Lara, and M. Wimmer, "An extensible meta-modelling assistant," in *International Enterprise Distributed Object Computing Conference*, 2016, pp. 1–10.

[11] S. Sen, B. Baudry, and H. Vangheluwe, "Towards domain-specific model editors with automatic model completion," *Simulation*, vol. 86, no. 2, pp. 109–126, 2010.

[12] S. Mazanek, S. Maier, and M. Minas, "Auto-completion for diagram editors based on graph grammars," in *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2008, pp. 242–245.

[13] T. Pati, D. C. Feiock, and J. H. Hill, "Proactive modeling: auto-generating models from their semantics and constraints," in *Workshop on Domain-specific modeling*. ACM, 2012, pp. 7–12.

[14] F. Steimann and B. Ulke, "Generic model assist," in *MODELS*. Springer, 2013, pp. 18–34.

[15] T. Kuschke, P. Mäder, and P. Rempel, "Recommending auto-completions for software modeling activities," in *MODELS*, 2013, pp. 170–186.

[16] F. Basciani, J. Di Rocco, D. Di Ruscio, A. Di Salle, L. Iovino, and A. Pierantonio, "MDEForge: an Extensible Web-Based Modeling Platform," in *International Workshop on Model-Driven Engineering on and for the Cloud*, 2014, pp. 66–75.

[17] R. Hebig, T. H. Quang, M. R. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use uml: mining github," in *MODELS*. ACM, 2016, pp. 173–183.

[18] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *Joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 213–222.

[19] M. Stephan and J. R. Cordy, "A survey of model comparison approaches and applications." in *International Conference on Model-Driven Engineering and Software Development*, 2013, pp. 265–277.

[20] ——, "MuMonDE: A framework for evaluating model clone detectors using model mutation analysis," *Software Testing, Verification and Reliability*, p. e1669, 2018.

[21] D. Wightman, Z. Ye, J. Brandt, and R. Vertegaal, "Snipmatch: using source code context to enhance snippet retrieval and parameterization," in *Symposium on User interface software and technology*, 2012, pp. 219–228.

[22] M. Stephan, M. Alalfi, and J. R. Cordy, "Towards a taxonomy for simulink model mutations," in *International Workshop on Mutation Analysis*, 2014, pp. 206–215.