Towards Qualitative Comparison of Simulink Model Clone Detection Approaches

Matthew Stephan, Manar H. Alafi, Andrew Stevenson, James R. Cordy
School of Computing, Queen's University, Kingston, ON, Canada
{*stephan,alalfi,andrews,cordy*}*@cs.queensu.ca*

***Abstract*—In this position paper we briefly review the Simulink model clone detection approaches in literature, including a new one currently being developed, and outline our plan for an experimental comparison. We are using public and private Simulink models to compare approaches based on clone relevance, performance, types of clones detected, user interaction, adaptability, and the ability to identify recurring patterns using a combination of manual inspection and model visualization.**

***Keywords*-clone detection; Simulink; comparison**

I. Introduction

Clone detection at the code level is a heavily researched and studied topic [1]. Clone detection of higher level representations, or *model clone detection*, is much newer in comparison. As the use of models in software development becomes more and more prevalent, the need to identify problems at these higher levels is increasingly important. While the underlying problem of code and model clone detection is the same in that we are looking for identical or similar artifacts, the different nature of the software representations necessitates solutions with different approaches.

In this paper, we outline our plans for a comprehensive qualitative comparison of existing approaches for model clone detection. Our primary goal is the assessment of both strengths and weaknesses in various areas. We begin by briefly describing the problem of model clone detection and listing the various approaches that have been applied thus far, including a new one currently under development. We continue by outlining the comparison we plan on performing in Section IV including the model sets we will be using as our sample sets and the various criteria being considered.

II. Problem Definition

Model clone detection involves finding multiple segments of higher-level software models that are similar or identical with respect to some measure of similarity. The particular measure will depend on the types of models being analyzed, the domain of the models, and the end goal and users of the clone-detection analysis. Like code clones, there are three types of model clones that can be discovered: Type 1 or *exact clones*, Type 2 or *renamed clones*, and Type 3 or *near-miss clones*. Exact clones are connected sets of model elements that are identical to one another based on the similarity measure. Renamed clones are identical except for differences in names and values of labels and elements. Near-miss clones are those that are are similar but may allow for small structural or connectivity differences up to a certain difference threshold.

For the purposes of this paper and our experiments, we will be focusing on Matlab Simulink models[1]. Simulink models are quite prevalent in the domains we are interested in, and the majority of model clone detection work thus far has been on Simulink. In the following section, we discuss the Simulink model clone detection approaches from the literature that we intend to evaluate.

III. Approaches

In general, all approaches begin by normalizing the models in order to filter out information that is not relevant to the particular measure of similarity being used.

Deissenboeck et al. [2] have extended the *ConQAT* tool suite to detect exact subsystem clones in Simulink models. They detect clone pairs using a breadth-first search of the model graph and an inspection of the nodes' neighbourhoods. Clone pairs are then clustered into clone classes by representing the pairs in a graph and identifying sets of connected pairs as clusters. Pham et al. [3] attempt to improve on *ConQAT* by providing *eScan* and *aScan* in order to detect exact, and near-miss clones, respectively. Their improvements utilize graph mining work and Simulink-specific properties.

Recently, Petersen [4] developed a model clone detector called *Naive Clone Detector* for exact clones. Like the previous approaches, they use graph-based model properties and Simulink information, but by contrast, they use a top-down approach. Specifically, they identify a relatively small number of large fragments and then match these against the other large fragments to determine if there are any "cloned core fragments", and then proceed to deeper levels.

In our own work [5], we are currently extending *NiCad* [6], a tool for detecting near-miss clones in source code, to detect model clones in Simulink. By analogy with NiCad's code clone detection strategy, we proceed by (1) building a grammar for the Simulink model representation; (2) identifying "granules" in the grammar, that is, the structural sub-units to be compared; (3) normalizing the sub-model granules to sort internal elements and strip out any extraneous or irrelevant information; and (4) running the LCS-based clone detection algorithm of NiCad, which allows for differences up to a given threshold.

[1] http://www.mathworks.com/products/simulink/

IV. A Comparison Experiment

The primary goal of our comparison is to compare the strengths and weaknesses of each approach and to evaluate the potential of the *NiCad*-based one that we are developing. We are particularly interested in the question of which methods may be better suited to identification of frequent sub-model patterns in large model sets, a goal of our industrially-sponsored research. And, of course, we hope to identify areas for improvement and continued research in model clone detection.

In keeping with our previous work on comparison of code clone detection techniques [1], we will focus on qualitative comparison. We will assess each approach's ability to detect exact, renamed and near-miss clones independently. Specifically, for exact and renamed clones, we will compare *ConQAT*, *eScan*, *Naive Clone Detector*, and our *NiCad* approach after setting the *NiCad* difference threshold to 0.0, with and without renaming normalization. For near-miss Simiulink clones, there is only *aScan* and our *Nicad* approach. However, *aScan* is not publicly available and other authors have had trouble implementing it [7], so *NiCad* may have to be evaluated by itself.

Due to large differences in output formats, results from the various methods will be compared primarily by hand in this first experiment, assisted by the *ConQAT* Simulink *Model Quality Assessor*[2], an Eclipse plugin that allows model clone detection results to be visually inspected in context given a clone representation in the appropriate format.

A. Example Models

Similarly to the early comparison of *ConQAT* versus *eScan* [3], we will be primarily relying on models that are publicly available from Matlab Central[3], in conjunction with models available from our industrial partners. We choose the same public models that were used by Pham et al. [3] and by Deissenboeck et al. [7] in order to have an existing basis of comparison. Neither *Naive Clone Detection* nor our *NiCad*-based approach have yet been tested on these models.

B. Dimensions of Evaluation

Before beginning a qualitative comparison or evaluation, it is important to be clear on the qualities to be compared. Our first two aspects are the same ones used in the preliminary work done by Deissenboeck et al. [7].

1) Relevance: Relevance, or precision, of the clones retrieved is a key area of concern in model clone detection. Deissenboeck et al. identify five qualities of relevance that we will use in our evaluations: *node size, clone weight, relative weight, interface weight* and *interface node size*.

2) Performance: Performance is an important measure of usability and scalability. We will also consider the usual tradeoff between recall and performance.

[2]http://www.cqse.eu/?page_id=81

[3]http://www.mathworks.com/matlabcentral/

3) Clone detection type: We will consider which types of clones (1, 2, and 3) [1] the approaches are able to detect.

4) User interaction required: It is important to consider the amount of manual interaction required by each approach to get the desired results. For example, some approaches require multiple iterations, possibly with user feedback, to discover relevant clones while others require none.

5) Adaptability: Adaptability refers to the ability of an approach to deal with change. For example, if a tool is tied to a specific model version or if we are interested in different granularities of clone pairs, we must consider how much work is required to adapt the approach to the new situation.

6) Model pattern granularity: Most of the tools attempt to extract clone classes in order to identify potential libraries for future reuse. Which approaches are well-suited to finding useful patterns or antipatterns at appropriate granularities?

V. Conclusion

We have briefly introduced the state of the art in model clone detection approaches, including one currently under development, and outlined our plans to compare them. We described the sample sets of models that we will be using and the criteria we will be focusing on in our evaluation. We hope to have early results ready for the workshop.

Acknowledgments

This work is supported in part by NSERC, as part of the NECSIS Automotive Partnership with General Motors, IBM Canada and Malina Software Corp. Benjamin Hummel has been very helpful in providing advice and the assistance with model sets and *ConQAT*.

References

[1] C. Roy, J. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470–495, 2009.

[2] F. Deissenboeck, B. Hummel, E. Jurgens, B. Schatz, S. Wagner, J. Girard, and S. Teuchert, "Clone detection in automotive model-based development," in *ICSE*, 2009, pp. 603–612.

[3] N. Pham, H. Nguyen, T. Nguyen, J. Al-Kofahi, and T. Nguyen, "Complete and accurate clone detection in graph-based models," in *ICSE*, 2009, pp. 276–286.

[4] H. Petersen, "Clone detection in Matlab Simulink models," Master's thesis, Technical University of Denmark, 2012, iMM-M.Sc.-2012-02.

[5] M. Alafi, J. Cordy, T. Dean, M. Stephan, and A. Stevenson, "Near-miss model clone detection for Simulink models," in *IWSC*, 2012, (to appear).

[6] C. Roy and J. Cordy, "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *ICPC*, 2008, pp. 172–181.

[7] F. Deissenboeck, B. Hummel, E. Juergens, M. Pfaehler, and B. Schaetz, "Model clone detection in practice," in *IWSC*, 2010, pp. 57–64.