Submodel Pattern Extraction for Simulink Models

[Extended Abstract]

James R. Cordy
School of Computing
Queen's University
Kingston, Ontario, Canada K7L 3N6
cordy@queensu.ca

ABSTRACT

Long before MDE became a popular method for software development, Simulink was firmly established as a tool for model–driven development of hybrid industrial systems. While practical and expressive for model creation and reuse, Simulink lacks for good abstraction mechanisms, and copy–paste–modify is a standard paradigm in Simulink development, leading to large numbers of variants of similar submodels. SIMONE is a framework and tool for automatically identifying, classifying and formalizing submodel patterns in Simulink models, using near-miss clone detection to find similarities and model merging to identify points of variance. The result is a set of submodel patterns which can be used as a reference for variance in the models, supporting consistency analysis, test optimization, fault identification and the disciplined generation of new subsystem instances using feature selection.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*model driven engineering*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*reverse and re-engineering*

General Terms

Design, Measurement, Verification

Keywords

Model–driven engineering, model patterns, Simulink

1. INTRODUCTION

Model-driven engineering (MDE) is rapidly growing as a software development technique in industry. Simulink[1] is an established tool for the modelling and code generation of hybrid embedded systems, which is already widely used in the automotive industry. As is the case for traditional programming languages, copy-paste-modify is a common reuse technique in Simulink model development, and with the wide range of similar products in automobile manufacturing, these copies can easily get out of hand.

The NECSIS Model Pattern Engineering project is aimed at addressing this issue by analyzing our industrial partners' large set of Simulink models to discover, catalogue and characterize the domain- and application-dependent submodel patterns in repeated use in their software development. Our idea is not to be predictive, but rather emergent - we seek those patterns which are in actual use in our partner's production models. The project consists of three phases: *discovery*, in which similar submodels are identified and clustered; *formalization*, in which these clusters are merged into parameterized patterns; and *application*, in which these patterns are applied in practice to aid in tasks such as consistency and best practice analysis, test- and code-generation optimization, identification of potential faults, variance analysis and model product line creation.

2. MODEL CLONE DETECTION

In the first phase, discovery, we are seeking to uncover similar submodels in a large set of Simulink models. The discovery of similar fragments, or *code clones*, in software source code is an established technique with almost two decades of research and several production clone analysis tools available [6]. Code clone research identifies four kinds of code clones: type 1, or exact clones, which differ only in formatting of the source text; type 2, or renamed clones, which differ in formatting and naming of elements in the code; type 3, or near-miss clones, which differ in formatting, naming, and minor additions, deletions or modifications of the code; and finally, type 4, or semantic clones, which are code sections that implement the same functionality in different ways.

Because they are specified graphically, using a two-dimensional network of connected parts, detection of similarity in models is a different problem. Thus clone technology for graphical models such as Simulink has been based on treating the models as the nodes and edges or graphs, where similar connected subgraphs of blocks can be identified by tools such as ConQAT [3]. While this technique works well for exact (type 1) and renamed (type 2) submodel clones, the extension to type 3 (near-miss) clones, those of interest in our work, has proven difficult due to the complexity of approximate subgraph matching.

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

SPLC 2013 Tokyo, Japan

While near-miss (type 3) clone detection is difficult for graphs, it is a firmly established technique in code clone detection, and tools such as NiCad [4] have demonstrated high levels of accuracy in finding groups of almost-similar code functions and blocks in large bodies of source code [5]. Thus in our work in pattern identification in Simulink, we have explored a crazy idea: can textual code clone detection be used to find near-miss clones in graphical models? In particular, can an extension of NiCad be used as the basis of our pattern discovery, to find near-miss submodel clones in Simulink graphical models?

3. SIMONE

SIMONE [1], is a Simulink near-miss submodel clone detector based on this idea. SIMONE leverages the NiCad code clone detector engine to analyze the proprietary internal textual representation of Simulink models. The adaptation of code clone detection to graphical models poses several challenges. First, NiCad is a structure-sensitive code analyzer, and requires a syntactic parse of the code (a "parse tree") in order to work. Thus we needed a grammar for the Simulink internal textual form (".mdl" files). Because there is no published reference for the syntax of the Simulink textual representation, we used grammar inference techniques [7] to infer a syntax for Simulink .mdl files.

Second, NiCad is designed to find similarities at a specific granularity, such as a Java class, method or code block, in order to yield meaningful units as results. Fortunately, Simulink is a hierarchical modelling method, with three clear levels of granularity, *model*, corresponding to a whole program; *system*, corresponding to a hierarchical unit such as a class or method; and *block*, corresponding to a single statement or expression in Java. Since we are interested in submodel patterns, whole models were clearly too coarse-grained, and single blocks clearly too fine-grained. Moreover, if our patterns are to be meaningful to Simulink engineers, then they should be in a form that is recognizable to them. Thus Simulink "systems" were chosen as the target granularity for our submodel clones, and a NiCad "extractor" for Simulink (sub-)systems was crafted to collect the submodels to be compared for similarity.

Third, the textual form of Simulink models encodes not only the structure of the model itself, but also a large set of elements and attributes describing the presentation of the model, such as the size, colour, font, and other attributes of model elements; the window size, layout, printing options, and other attributes of the model presentation, and many other options. None of these is relevant to submodel similarity analysis, and a filtering transformation was implemented to filter out presentation elements and attributes from the textual representation of the submodels to be compared.

The most difficult problem to be faced in using a text-based code clone detector on the textual form of graphical models is that there is no defined linear order for the elements of the model in the textual form – the meaning of the graphical model remains the same no matter what order its elements and connections are defined in its textual representation. To address this problem, SIMONE uses a normalizing topological sort of the textual elements before comparison, effectively defining a standard order for the textual form. Both filtering and sorting are implemented as TXL [2] source transformations of the extracted Simulink textual representation.

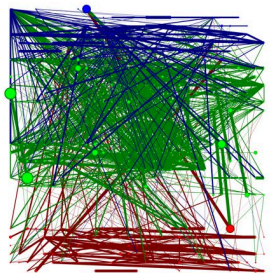

Figure 1: Near-miss subsystem clone relationships in three production industrial automotive models

4. SUBMODEL CLONES

SIMONE has been validated on a large set of publicly available Simulink models, including all of the demonstration models distributed with Simulink itself,[2] and the entire set of models available at Matlab Central[3]. Comparison with the ConQAT state-of-the-art graph-based Simulink model clone detector found that text-based SIMONE finds all of the type 1 (exact) and type 2 (renamed) subsystem clones in these examples found by ConQAT, and additionally finds the many near-miss (type 3) subsystem model clones that we are interested in. These near-miss clones form the basis of our submodel pattern analysis.

As an example, Figure 1 shows a graphical representation of the results of a SIMONE near-miss subsystem clone analysis of three production industrial models (indicated by the three different colours) from our automotive industry partners. The nodes in the graph represent the extracted subsystems, and the edges represent the near-miss clone relationships between them. The size of the nodes indicates the relative size of the corresponding subsystems (100–20,000 lines of textual representation), and the thickness of the edges represents the strength of the similarity (70–100% similar).

5. CLONE CLASSES

The first step in inferring submodel patterns in the model set consists of clustering similar subsystems into connected groups called *clone classes*. Clone classes are created automatically by the NiCad clone detector from the clone relationships by grouping the connected components of the clone relationship graph, yielding clusters of similar subsystems. Figure 2 shows a conceptual view of the clone classes for the production model subsystem clones of Figure 1.

Of course, Simulink engineers are not interested in conceptual views – they are much more interested in being able to use the results of our analysis to assist them in their everyday work in the Simulink IDE. To make these results accessible and actionable, we have created the *SimNav* clone class exploration interface as an extension to Simulink itself (Figure 3). Using SimNav, the engineer can explore a table of the clone classes and bring up the the actual similar subsystems in the Simulink graphical model editor interface they are using to work with the models.

[2]www.mathworks.com/products/simulink/examples.html
[3]www.mathworks.com/matlabcentral

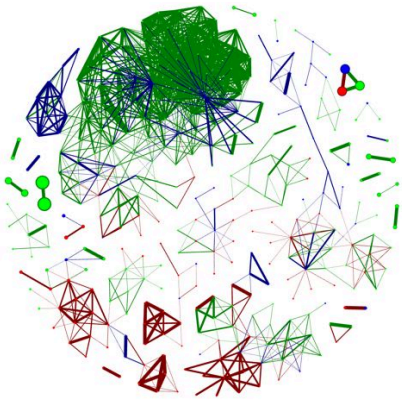

Figure 2: Near-miss subsystem clones of Fig. 1 clustered into clone classes

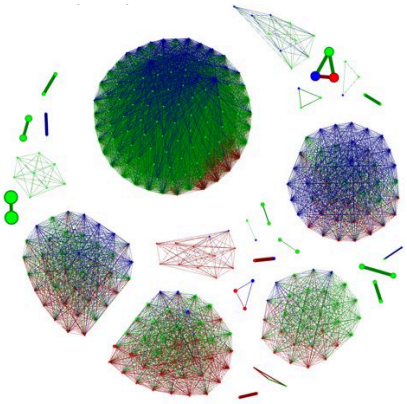

Figure 4: Subsystem clone classes of Fig. 2 partitioned into pattern instances

6. PATTERNS

Clone classes can be used directly as sets of exemplars to represent the subsystem pattern of which they are instances. That is, we can use a subsystem clone class as its own pattern, by using the clone detector to search any new model for subsystems similar to any subsystem of the pattern clone class. Using this definition, we can partition the clone relationships discovered by SIMONE into pattern instances, forming a clear distinction between the patterns. Figure 4 shows a conceptual view of the patterns inferred from the clone classes of the production model subsystem clones of Figure 1.

While this method works well from a practical standpoint, it is somehow dissatisfying from a conceptual one, in that we have not actually characterized the pattern and its variants except by example. What we really want to do is to create an actual Simulink subsystem model that represents the pattern and its variants in a single unit. In current work we are developing *SimPat*, a notation based on merging the elements of the clones in a clone class and distinguishing the points of variation explicitly.

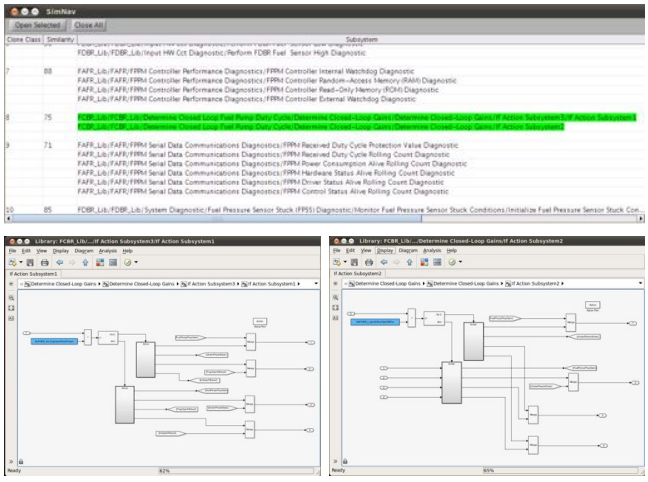

Figure 3: The SimNav subsystem clone class exploration interface in the Simulink model editor

Using Simulink to view these merges yields a representation very close to the the goal we have in mind. Figure 5 shows an example of such a pattern viewed in the Simulink model editor. Our plan is to integrate pattern viewing into SimNav, to highlight the variance in each instance of the pattern and facilitate product variance management.

7. EVOLUTION

Model evolution is an important topic that has been heavily studied for UML models. In spite of its established predominance in industry, however, little research has been done on the evolution of Simulink models. In our ongoing work, we are leveraging our inferred subsystem patterns to study the maintenance evolution of production Simulink models, with the goal of inferring patterns of model evolution analogous to our our patterns of submodel reuse.

Our work is based on tracing the migration of subsystem patterns across versions, using individual pattern instances as the links between the clone patterns of one version and those of subsequent versions. By understanding how common subsystem patterns evolve to split or merge between versions, we hope to characterize the evolution of the models at a higher level. Figure *SimTrace* shows an example of how the submodel patterns of one version have evolved to split into separate patterns in following versions. The diagram is generated by our prototype pattern evolution tool *SimCCT*.

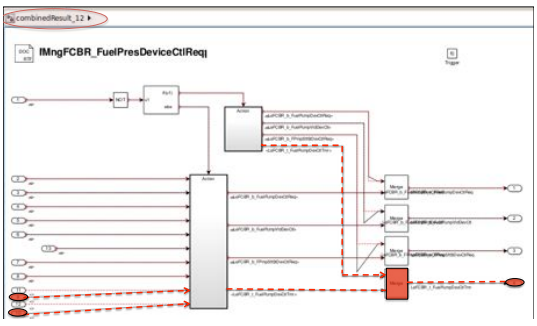

Figure 5: Representing variance in subsystem clone patterns using Simulink

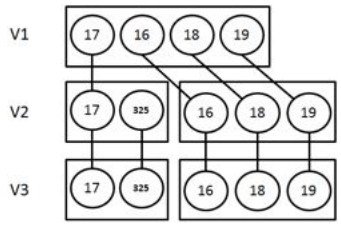

Figure 6: Evolution of subsystem clone patterns

8. CONCLUSION

In this presentation we have outlined a process for inferring emergent submodel patterns in Simulink models using text-based near-miss clone detection to find and cluster similarities. The results are integrated into the Simulink editing environment for use by production engineers in understanding and managing reuse and variance in their models. This work is ongoing, and we are currently focussing on representing these patterns in a generic form that can characterize and identify new instances and their variations in a production Simulink engineering environment.

9. ACKNOWLEDGMENTS

This is joint work with Manar Alalfi, Thomas Dean, Matthew Stephan and Andrew Stevenson of the Software Technology Laboratory at Queen's University. Our work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), as part of the NECSIS Automotive Partnership with General Motors, IBM Canada and Malina Software Corp. The author thanks Joseph d'Ambrosio and Cheryl Williams of General Motors Research & Development for their assistance in obtaining permission to use production GM models as examples in this presentation.

10. REFERENCES

[1] M. H. Alalfi, J. R. Cordy, T. R. Dean, M. Stephan, and A. Stevenson. Models are clones too: Near-miss model clone detection for Simulink models. In *ICSM 2012*, pages 295–304, 2012.

[2] J. R. Cordy. The TXL source transformation language. *Science of Computer Programming*, 61(3):190–210, 2006.

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

[4] C. K. Roy and J. R. Cordy. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *ICPC 2008*, pages 172–181, 2008.

[5] C. K. Roy and J. R. Cordy. A mutation/injection-based automatic framework for evaluating code clone detection tools. In *Mutation 2009*, pages 157–166, 2009.

[6] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495, 2009.

[7] A. Stevenson and J. R. Cordy. Grammatical inference in software engineering: An overview of the state of the art. In *SLE 2012*, pages 204–223, 2012.