Analysis and Clustering of Model Clones: An Automotive Industrial Experience

Manar H. Alalfi, James R. Cordy, Thomas R. Dean
School of Computing, Queen's University, Kingston, Canada
Email: {alalfi, cordy, dean}@cs.queensu.ca

Abstract—**In this paper we present our early experience analyzing subsystem similarity in industrial automotive models. We apply our model clone detection tool, SIMONE, to identify identical and near-miss Simulink subsystem clones and cluster them into classes based on clone size and similarity threshold. We then analyze clone detection results using graph visualizations generated by the SIMGraph, a SIMONE extension, to identify subsystem patterns. SIMGraph provides us and our industrial partners with new interesting and useful insights that improves our understanding of the analyzed models and suggests better ways to maintain them.**

I. Introduction

In todays automotive industry, models are widely used to generate production software code. A modern automobile may have 100 million lines or more of production software source code on board, and up to 80% of the code deployed on up to 100 embedded control units can be generated from models specified using domain-specific formalisms such as Matlab/Simulink [1]. The size and complexity of software in the embedded domain, especially in the automotive software systems, is expanding rapidly, while the innovation cycle length is decreasing with high cost pressure and large numbers of product-line variants. Consequently, software development in this domain has adopted a highly reuse-oriented approach, where general purpose domain specific libraries with elements such as PID-controllers are being reused in the manufacture of many new software components. Dealing with this complexity and the high frequency of software reusability requires sophisticated software tools to manage the massive amounts of information used by engineers in software development projects.

This rapid growth has led to challenges that are already well known from classic programming languages. In particular, the presence of copied program elements or "clones", which can affect productivity and software maintenance, is also manifested in models. Thus the identification of common or similar elements in different parts of the software is important to the model-based development process.

To address this need, we have developed a method and toolset called SIMONE [2] that uses clone detection for the analysis and formalization of subsystem similarity in industrial models. In this paper we present our first experience in the analysis and pattern extraction of subsystem patterns in a set of production Simulink automotive models, using visualization techniques to provide insight. Our method is intended to assist reuse in model development in a number of ways: standards and consistency analysis or enforcement in model maintenance; failure and change propagation in model maintenance; and verification and test optimization in model testing. We envisage this work as a fundamental enabler for the future of higher level modeling, model transformations and meta-modeling frameworks, customized to specific domains.

II. Approach

This paper describes our first experience in applying our analysis method in an empirical study of an set of actual production models from our industrial partners at General Motors, using pattern mining and clone detection technologies to discover a catalog of repeated subsystem patterns. We have automated much of this first step using subsystem clone classes from our tool SIMONE, a near-miss clone detector for Simulink models [2], from which we derive a first approximation of the pattern set. Our plan is to organize these discovered patterns into a taxonomy with the goal of covering all of the subsystem patterns in the models. In this paper we introduce SIMGraph, a graph visualization extension for SIMONE results, which helps us to visualize and understand Simulink subsystem clones and patterns in a more intuitive and understandable way. In the following sections we will discuss in more detail the phases of our analysis, and our case study analyzing a set of industrial automotive Simulink models.

III. Clone Identification

Our analysis consists of three phases. In the first, "discovery" phase of our analysis, our primary goal is the discovery and identification of common subsystem patterns in an example production model set obtained from our industrial partners at General Motors.

For that purpose, we have used our previously-developed method for leveraging text-based code clone detectors to find near-miss clones in graphical models, and have demonstrated SIMONE [2], an implementation for Simulink models based on the NICAD clone detector [3]. In that work, we outlined the challenges of using a parser- and text-based method on graphical models, described our solutions using filtering and sorting of the textual representation, and compared our results to a state-of-the-art graph-based method, showing that our near-miss detection can find meaningful subsystem clones that graph-based methods can miss. Our approach generalizes to

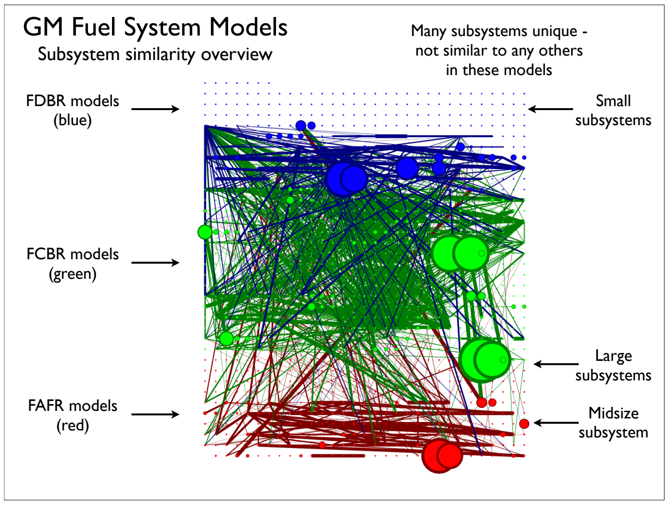

Fig. 1. Subsystem Similarity in Fuel System Models

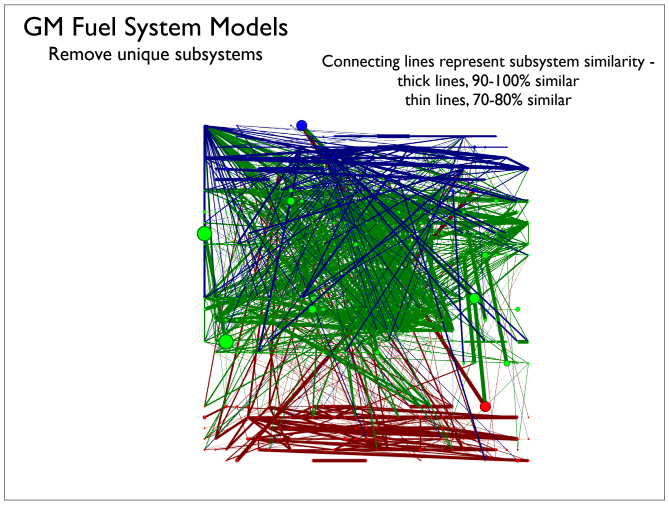
GM Fuel System Models
Remove unique subsystems
Connecting lines represent subsystem similarity - thick lines, 90-100% similar
thin lines, 70-80% similar

Fig. 2. Subsystem Similarity with Unique Subsystems Removed

other modeling languages, including UML-based ones, with only customization of the filtering and sorting algorithms.

IV. Subsystem Clone Clustering

In the second, "clustering" phase, we aim to organize and generalize the identified concrete subsystem clone classes into a formal taxonomy of generic patterns that can cover the models of the example model set. SIMONE generates an initial clustering of similar subsystems into model patterns based on the NICAD algorithm for clustering near-miss clones into "clone classes", using a dynamic clustering based on the size and percentage difference between identified subsystem clone pairs. In this clustering, a database of clone pairs ordered by size is generated after which the clustering is formed without any post processing. Clone classes are then formed by selecting the largest remaining clone as an "exemplar" or distinguished representative of its clone class, and gathering all clone pairs within a given difference threshold (30% in this experiment) into the cluster.

V. Subsystem Clone Visualization

The NICAD clone detector generates clone reports in two textual formats, XML and HTML. It generates two kinds of reports, one that presents cloned subsystems as pairs, and another that presents them as clusters ("clone classes") as described above. While these textual reports are useful for analyzing small similarities in source code, they are not the right format for understanding large-scale similarity in graphical models. Thus we have developed SIMGraph, an extension to SIMONE, which presents clone results as graphs. We used source transformation technology to transform SIMONE's NICAD textual reports into graph files that can be imported by Gephi [4], an open-source tool for visualizing and analyzing large network graphs of complex systems. Gephi has many interesting features, including the ability to present dynamic and hierarchical graphs, as well as interactive filtering and clustering features.

A. *Subsystem Clone Representation*

To create a more intuitive representation of SIMONE results we generated graphs using the following mappings:

- Subsystem Model: Each model is represented by a colour, and its subsystems are coloured to indicate the model they belong to.
- Subsystem Size: Each subsystem of a model is represented by a circle whose size is proportional to the size of the subsystem, larger subsystems represented by larger circles.
- Subsystem Similarity: Subsystem clones are represented by a line joining the subsystem circles, the thickness of which represents the level of similarity between them, ranging from 90-100% (thickest lines) to 70-80% (thinnest).

B. *Levels of Clone Abstraction*

We generated two types of Gephi graphs to represent the results at two levels of abstraction:

- Subsystem Clone Pairs: Subsystem clone results as lines connecting similar subsystems (Figures 1 and 2).
- Subsystem Clone Clusters: Subsystem clone results grouped into categories based on clustering and categorization algorithms (Figures 3 and 4).

VI. Case Study

In this section we present our analysis of a production industrial system obtained from our industrial partners at GM, the Fuel System Simulink models. Applying SIMONE without any normalization other than sorting and filtering to these models extracted 1,091 subsystems from which it identified 245 subsystem clones, initially categorized into 35 clusters. We refined these results and clustering by running SIMONE using a blind renaming normalization, which yields model clones that are similar in structure, ignoring name differences in embedded model elements. The number of clones identified

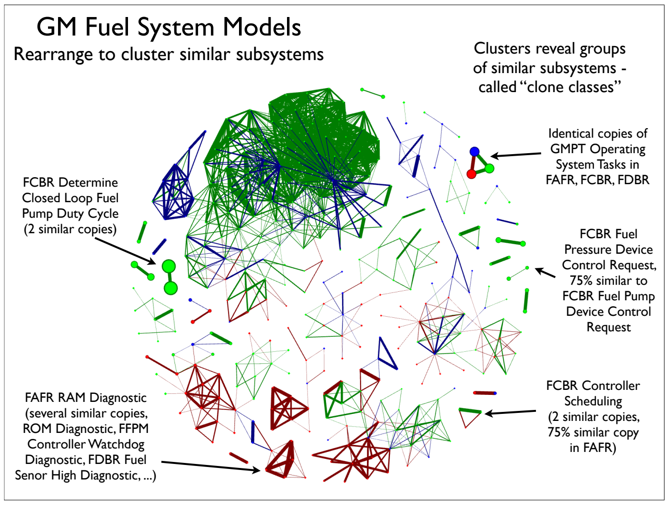
GM Fuel System Models
Rearrange to cluster similar subsystems
Clusters reveal groups of similar subsystems - called "clone classes"
Identical copies of GMPT Operating System Tasks in FAFR, FCBR, FDBR
FCBR Determine Closed Loop Fuel Pump Duty Cycle (2 similar copies)
FCBR Fuel Pressure Device Control Request, 75% similar to FCBR Fuel Pump Device Control Request
FAFR RAM Diagnostic (several similar copies, ROM Diagnostic, FFPM Controller Watchdog Diagnostic, FDBR Fuel Senor High Diagnostic, ...)
FCBR Controller Scheduling (2 similar copies, 75% similar copy in FAFR)

Fig. 3. Subsystem Clone Pairs Clustered by Similarity

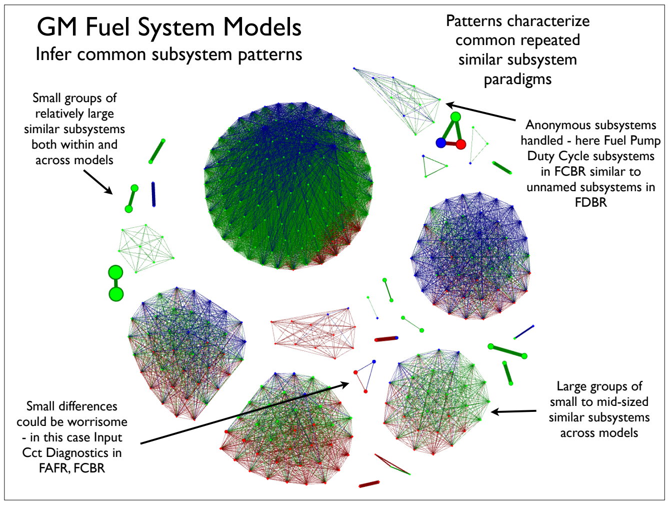
GM Fuel System Models
Infer common subsystem patterns
Patterns characterize common repeated similar subsystem paradigms
Small groups of relatively large similar subsystems both within and across models
Anonymous subsystems handled - here Fuel Pump Duty Cycle subsystems in FCBR similar to unnamed subsystems in FDBR
Small differences could be worrisome - in this case Input Cct Diagnostics in FAFR, FCBR
Large groups of small to mid-sized similar subsystems across models

Fig. 4. NICAD Clone Classes: Closure into Subsystem Patterns

with similarity 70%-100% then increases to 1,147 clones clustered into 30 clone classes.

Using SIMGraph, we generated a graph to represent subsystem clone results, which intuitively presents the level of similarity between the identified subsystem clones as well as subsystem sizes. The analyzed Fuel System consists of three main models, FAFR, FCBR and FDBR. When generating the graph, we set the colour attribute to be different for each of these models, thus subsystems of FAFR are coloured in red, FCBR in green, and FDBR in blue. Subsystems of these models are represented as circles, where the circle size corresponds to the subsystem size (lines of text in the internal Simulink representation of the subsystem).

Figure 1 shows the resulting graph. Subsystems that have a similarity relation are connected, and unique subsystems are not connected. Figure 2 shows the graph after we filter out these unique subsystems, since they are not of interest from a clone detection prospective. For example, the large green subsystems on the right of Figure 1 disappear in Figure 2 since they are unique. We can also observe the level of similarity between subsystem clone pairs, represented by the thickness of the lines connecting them. The thicker the connecting line the more similar the subsystem pairs are.

While Figures 1 and 2 show the cloning relation at the level of subsystem clone pairs, a more interesting view, the clustering view, better reveals the similarity between groups of subsystems (Figures 3 and 4). Figure 3 is a layout of the graph of Figure 2 clustered using the Fruchterman Rheingold clustering algorithm to better reveal groups of similar subsystems. Figure 4 on the other hand shows the NICAD clone classes, a closure of the clone pair relations into cliques representing repeated subsystem patterns. This better reveals the relative size and distribution of subsystem clone classes.

VII. Analysis Insights

SIMgraph arranges that the names and file paths of the subsystems are stored as attributes of the nodes of the generated graphs. Thus the identities of the subsystems and models can be explored in Gephi simply by clicking on the nodes of the graph. Using this technique, we can gain deeper insight about what we are seeing.

- *Inferring common subsystem patterns:* The NICAD clustering shown in Figure 4 provides us with an initial insight about model pattern characterization, since similar subsystems have been partitioned into separate categories, with a distinguished element serving as the exemplar of the group. These categories will form the basis of our pattern formalization in building a subsystem pattern taxonomy for the GM models.
- *Identifying similarities across models:* Visualization the clone relationships helps us to identify not only similar subsystems within models, but also across models. This is made possible using the colouring scheme to distinguish the different models. For example, on the left of Figure 3 we can see that there are two similar copies of Determine Closed Loop Fuel Pump Duty Cycle in the FCBR model, since the two circles have the same green colour. On the right of the same Figure we can see three medium-sized circles of different colours connected to each other by a thick line, which reveals that there are identical copies of a medium sized subsystem, the GMPT Operating System Tasks, used in all three models, FAFR, FCBR and FDBR.
- *Handling anonymous subsystems:* This an important feature of SIMONE, by which anonymous (unnamed) Simulink subsystems are identified and grouped with other similar subsystems. This allows engineers to identify these subsystems as instances of other known named subsystems, and possibly name or refactor them to document the relation. For example, on the right of Figure 4 an anonymous subsystem in the FDBR model is identified

as similar to the Fuel Pump Duty Cycle subsystems of FCBR.

- *Identifying potential maintenance problems:* The graph visualization can also help engineers identify potential maintenance problems. For example, similar subsystems with the same name across models connected by a thin line indicate that even though the subsystems have the same name they are only 70% similar. The bottom left of Figure 4 shows an example on this case, where the thin connections between the Cct Diagnostics subsystems of FAFR and FCBR indicate that they are not nearly a similar as might be expected.

VIII. Proposed Usage Scenarios

Potential general usage scenarios for our subsystem pattern analysis include: the automated discovery of common idioms, both global to the company and local to a project or domain, model optimization, potential bug discovery (inconsistent changes to instances of a pattern), functional and security compliance analysis, modeling standards enforcement, model library creation and reuse, and reuse of implementation and deployment processes and alternatives.

At a meeting to present our tools and results to GM engineers, they showed a keen interest in our approach, and they found the visual presentation useful as a first-level overview of the results our tools uncovered. While not intended for everyday model clone management (our SimNav interface [5] serves that purpose), team leaders found this overview useful in assessing the overall situation in a system. The meeting also helped our GM industrial partners to identify a more specific list of applications tailored to their own goals:

- Refining / adjusting / changing modeling guidelines to either prohibit common problem constructs, or encourage other constructs.
- Identifying new candidate constructs for addition to the GM block library. The block library not only provides primitives (native Simulink blocks), but also common subsystem constructs for easy re-use (e.g., basic first order lag filters).
- A set of common model patterns provides designers with a choice of alternatives. These alternatives can be explored by designers to identify which take best advantage of MATLAB's optimizations.

IX. Related Work in Clone Pattern Visualization

Dang al el. [6] report their experience at Microsoft research using their XIAO code clone detection approach in a real development setting. Like us, the authors present the importance of presenting clone results to the engineers in an understandable and intuitive way, as well as the need for near-miss clone detection techniques with sorting and filtering capabilities, which will help engineers to identify clones of interest and eliminate any irrelevant clones.

Ader and Kim [7] argue the importance of using graphs to help understand code clone results in the context of parameters and relationships, such as relation degree, size, and colour. They have developed their tool SoftGUESS for that purpose.

Yoshimura and Mibe [8] present their experience in visualizing code clones in an industrial setting, analyzing an enterprise business application at Hitachi. The authors emphasized the importance of tool support for large-scale systems and a suitable visualization of clone results to help stakeholders understand their systems.

All of the above approaches present support for using visualization to understand code clones, while our tool, SIMGraph, is specifically aimed at near-miss model clones.

X. Conclusions and Future Work

In this paper, we presented an initial analysis of production automotive models using our SIMONE model clone detection tool and its SIMGraph visualization extension. We presented a case study and some insights resulting from it, identifying potential usage scenarios with the help of the initial feedback received from our industrial partners. As future work, we plan to refine the initial clustering observed by our tools and to use pattern matching techniques adapted from source code analysis research to encode the taxonomy elements and organize them into a partitioning engine. The pattern set will then be back-validated by applying it to the analysis of the original example model set, and then applied to a larger set of industrial models to test completeness.

Acknowledgements

This work is supported by NSERC as part of the NECSIS Automotive Partnership with General Motors, IBM Canada and Malina Software Corp.

References

[1] M. Jungmann, R. Otterbach, and M. Beine, "Development of safety-critical software using automatic code generation," in *SAE World Congress*, 2004.

[2] 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 *ICSM*, 2012, pp. 295–304.

[3] 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, pp. 172–181.

[4] B. Hauptmann, V. Bauer, and M. Junker, "Using edge bundle views for clone visualization," in *IWSC*, 2012, pp. 86–87.

[5] J. R. Cordy, "Submodel pattern extraction for Simulink models," pp. 7–10, 2013.

[6] Y. Dang, S. Ge, R. Huang, and D. Zhang, "Code clone detection experience at microsoft," in *IWSC*, 2011, pp. 63–64.

[7] E. Adar and M. Kim, "Softguess: Visualization and exploration of code clones in context," in *ICSE*, 2007, pp. 762–766.

[8] K. Yoshimura and R. Mibe, "Visualizing code clone outbreak: An industrial case study," in *IWSC*, 2012, pp. 96–97.