Evolution of Model Clones in Simulink

Matthew Stephan, Manar H. Alalfi, James R. Cordy, and Andrew Stevenson

Queen's University,
Kingston, Ontario, Canada
{stephan,alalfi,cordy,andrews}@cs.queensu.ca

Abstract. A growing and important area of Model-Based Development (MBD) is model evolution. Despite this, very little research on the evolution of *Simulink* models has been conducted. This is in contrast to the notable amount of research on UML models, which differ significantly from *Simulink*. Code clones and their evolution across system versions have been used to learn about source-code evolution. We postulate that the same idea can be applied to model clones and model evolution. In this paper, we explore this notion and apply it to *Simulink* models. We detect model clones in successive versions of MBD projects and, with a new tool, track the evolution of model clones with respect to their containing clone classes. When there is a change in classification of a model-clone, we investigate what specifically evolved in the model to cause this classification change.

Keywords: model evolution, model clone detection, model clone evolution, Simulink

1 Introduction

Understanding software model evolution in **Model-Based Development** (MBD) is important as it can improve our ability to adapt to change, allows us to refactor more efficiently, and increases the quality and amount of analysis we can do on MBD projects. While still a relatively young area, there is a notable amount of work that discusses the evolution of UML models [5, 8, 10]. In contrast, there is very little research related to the evolution of *Simulink* models, a data-flow modeling language that is widely used in the automotive and communication industries, as well as other embedded areas. *Simulink* models differ significantly from anything in UML, with the most analogous diagram being a UML activity diagram, which is still quite different.

As noted in [13], there are a number of instances where code clones are used in order to perform source-code evolution analysis. Specifically, once a relationship can be established between two versions of a system, it can be employed as a means to understand the evolution of the system. Such a relationship can be realized by extracting code clones from different versions and then identifying and analyzing similar groups and how they have changed. We believe the same holds true for models. Thus, as a first step towards understanding *Simulink*

model evolution, we introduce the notion and use of *Simulink* **model clone evolution** (MCE). We focus on *Simulink* because there is sparse research on its evolution, it is of interest to our industrial partners, and *Simulink* **model clone detection** (MCD) is the most mature form of MCD.

In this paper, we use our near-miss MCD tool, *SIMONE* [1], to detect clones from successive versions of both industrial and publicly available models. Using a new tool we develop, we are able to track the evolution of a model-clone class' clone instances throughout multiple versions. When there is a change in the classification of the model-clone instance, we then delve deeper into the model itself to see and illustrate what exactly has evolved that caused this change and then explain it. The paper begins by providing background in Sect. 2 and defining key terms in Sect. 3. We then present the tool we developed for this work in Sect. 4 and our experiments with it, along with examples of models and their evolution, in Sect. 5. We present related and future work in Sect. 6 and conclude in Sect. 7.

2 Background

2.1 Simulink

Simulink models consist of 3 levels of granularity: whole models, (sub) systems, and blocks. Models contain systems, and systems contain other (sub) systems and blocks. This is similar to source files: models are like programs; systems are like methods, functions, and classes; and blocks are like statements in traditional programming languages. An important characteristic of Simulink models is "All block names in a model must be unique and must contain at least one character." [1]

2.2 Clone Genealogies

Kim et al. [9] define the notion of genealogy for code clone groups as the way in which a collection of clones evolves over multiple versions of a system. The clone group evolution they describe is in terms of code snippets, which are comprised of both text and location. For Kim et al.'s genealogies, each code clone group contains identical (exact) code clones which are matched to other clone groups based on textual similarity. Saha et al. [14] later consider clone groups containing non-identical (near-miss) code clones and match them to other groups by matching functions (code blocks) containing the code clones. We extend and modify the concepts and approaches from these works in order to apply them to *Simulink* models.

2.3 Model Clone Detection

Much like its counterpart, code clone detection, model clone detection entails discovering identical or similar fragments of model elements [4]. We recently

[1] http://www.mathworks.com/help/simulink/ug/changing-a-blocks-appearance.html

developed a model clone detector, called *SIMONE*, that is capable of detecting both exact and near-miss clones in *Simulink* models [1]. Its clone detection algorithm uses a sorted and filtered version of the underlying internal textual representation of the models stored in the Simulink MDL files. This is in contrast to *CloneDetective* [4], which treats *Simulink* models as graphs. Both techniques identify clones and group them together into clone classes, however, in *SIMONE*, a user-specified similarity threshold can be specified, such as 70% for near-miss and 100% for exact clones. There are other less-mature MCD techniques as well [17].

3 Definitions

A *Simulink* clone is essentially a similar subgraph of a larger *Simulink* system and is comprised of *Simulink* blocks (including sub-system blocks) and the lines that connect them. The basic units in our model-clone genealogy are these subgraphs, which we term **model clone instances** (MCIs). The attributes of an MCI are its list of blocks and lines, and its location. Location refers to the specific *Simulink* model and system(s) the MCI is contained in. For example, both *CloneDetective* and *SIMONE* produce XML clone reports that contain this information in some form. A **model clone class** (MCC) is a collection of MCIs grouped together by a model clone detector based on some measure of classification. All the *Simulink* MCD tools we have encountered thus far identify clone classes explicitly.

In order to trace a specific MCI across different versions, we can use the combination of (1) the model containing the MCI, and (2) the fully qualified path to the system (or sets of systems, for clones that span systems) comprising the MCI, that is, the trail of enclosing Simulink (sub) systems that contain the MCI's blocks and lines. Because all blocks, including those of type "subsystem" must have unique names in a *Simulink* model, this is a suitable source of clone traceability. This is analogous with Saha et al.'s code clone mapping where they determine if a code clone fragment is located within a function.

As Saha et al. have noted, with near-miss clones it is not possible to simply map one class to another in successive versions [14]. Analogously to what they do with functions and code clone classes, we extend their ideas to the modeling domain by taking a specific MCC, say MCC_v from version v, and seeing what MCCs in future versions contain MCIs from MCC_v. In contrast, however, while they are interested primarily with counting occurrences of code-clone evolution patterns, we are concerned mainly with how individual model clone instances evolve to cause a change in MCC classification. As such, we need to identify only if MCC_v yields one MCC in a future version, v+1; multiple MCCs in version v+1; or no MCCs in version v+1; and focus on the specific evolution of the MCIs involved in each case.

4 Tool Description

For the first step of our analysis, we developed a tool, called **Simulink Clone Class Tracker** (SIMCCT), that allows a user to select a specific MCC from one version of a system in order to display, in a GUI, what MCCs in future versions contain its MCIs. As input, the program takes in an ordered set of MCD results in XML form, with each XML file representing a different version in the evolution. We used a TXL [3] source transformation to change the XML output of *SIMONE* to a form more conducive to evolution analysis. The same can be done for *CloneDetective*. As demonstrated in a simplified version of the input in Fig. 1, the file contains a list of clones, sorted by classes. Each class contains sources, which correspond to the MCIs. As mentioned previously, these are comprised of blocks and lines, each with their own attributes of interest.

In brief, SIMCCT begins by parsing the input XML file we describe above and extracting the required information, treating each file as a version. It then identifies unique MCIs across all versions using our earlier definition and assigns each a unique ID number. This ID number is used in the GUI to represent the MCI, as a textual ID would be too long and unwieldy. Each time an MCC from the first version is selected by a user, related MCCs for successive versions are discovered and displayed by searching for the MCCs in future versions that contain the MCIs belonging to the selected MCC. So, for example, let us consider an MCC with class ID 4 from a first version, MCC_{v1c4}. It is selected and contains a set of MCIs, MCI_{v1c4}. In future version 'x' and class 'y', MCC_{vxcy} is displayed if MCI_{vxcy} contains any element from MCI_{v1c4}.

5 Experiment

We ran SIMCCT on both publicly available models and private models from our industrial partners. The public models include the **Automotive Power**

```
<clones>
    <class classid="#" nclones="#" similarity="#" ...>
        <source file="..."subsystem="..." ... >
             <block path="..." type="..." ...Block attributes.../>
             ...More Blocks...
             <line ...Line attributes"/>
             ...More Lines...
        </source>
         ...More Sources...
     </class>
     ...More Classes...
</clones>
```

Fig. 1: General form of SIMCCT input

Window (PW) System that comes with the *Simulink* example set and a large open-source **Advanced Vehicle Simulator**(AVS) [2].

To start, we analyzed the 3 systems using *SIMONE* with our best-fit [1] settings of 70% similarity and blind-renaming. Table 1 displays statistics about the results. The PW system is a smaller, compact, and simple system. AVS is quite large and complex, and has more clone pairs and MCCs than our industrial system set. Thus, we believe it is a fairly representative and rich system.

After transforming the MCD results into the SIMCCT format, we execute SIMCCT. As mentioned, we are looking specifically to note what MCCs in future versions contain MCIs from a user-selected MCC in an earlier/earliest version, v_1. For each MCC in v_1, its relation to future MCCs with respect to another version can be classified in one of five ways: (1) **1 to 1**; (2) **1 to 1***, which is the same as "1 to 1" except there are additional MCIs, missing MCIs, or a combination of both; (3) **1 to many**, which has no additional or missing elements in future MCCs; (4) **1 to many***, which has additional or missing elements in future MCCs; and (5) **1 to 0**, meaning the MCIs from the original MCC are no longer in any MCC. We can then use this information as grounds to investigate what model evolution has transpired on the MCIs to cause this relation.

Table 2 classifies the MCC evolution we observed in the three systems as it pertains to each MCC's MCIs in each system's first version. In our sample systems, we found no instances of "1 to many", that is, every time an MCC later had its constituent MCIs split into multiple MCCs, there were always additional elements present. For "1-to-many*" relationships, the AVS system had a "1 to 2" and our industrial set had a "1 to 2" and a "1 to 4".

5.1 Examples

We now showcase a set of examples from our experiment demonstrating different cases. We illustrate the examples by extending the representation Göde used for

[2] http://sourceforge.net/projects/adv-vehicle-sim/?source=dlp

Table 1: Systems Analyzed by SIMCCT

System Name	Version #	Model Files	SubSystems	Clone Pairs	MCCs
PW	1	1	18	7	5
	2	1	29	15	5
	3	1	33	23	6
	4	1	25	13	4
	5	1	45	39	6
AVS	r0000	69	861	1916	18
	r0080	69	1621	5693	35
	r0116	72	1714	5951	38
Industrial Set	55	9	977	600	20
	56	9	977	618	21
	57	9	986	624	23

Table 2: Relationship Classifications of MCCs w.r.t. Earliest Versions

System Name	Version	1 to 1	1 to 1*	1 to many	1 to many*	1 to 0
PW	2	1	4	0	0	0
	3	1	4	0	0	0
	4	1	3	0	0	1
	5	1	2	0	0	2
AVS	r0080	12	5	0	1	0
	r0116	9	8	0	1	0
Industrial Set	56	14	4	0	2	0
	57	14	4	0	2	0

the evolution of type-1 code clones [6], with MCCs being rectangles and MCIs being circles. In addition, we provide figures of some of the examples showcasing the specific evolution that has transpired. This is in order to highlight samples of changes that form various evolutionary MCC relationships.

We choose examples from public models as they adequately exhibit the cases and are available to all. We then investigate what evolution has taken place on the models themselves that caused the observed MCE. A reminder, each number within a circle refers to a uniquely identified key that corresponds to a unique MCI across all versions.

Power Window - Model Clone Class 3: This example is presented in Figs. 2 and 3. It contains MCC3, which begins with two MCIs, 5 and 6, that are 81% similar. In version 2, represented by the part underneath the dashed line in Fig. 3, MCI6 has 2 additional blocks and no longer belongs to any MCC. Conversely, MCI5 is simplified by replacing three blocks with one and is now 71% similar to MCI7 and other MCIs, causing a reclassification with them. Starting in version 4, MCI5 is simplified even further by removing more blocks and no longer belongs to any MCC. As such, MCC3 has a "1-to-0" relation to versions 4 and 5.

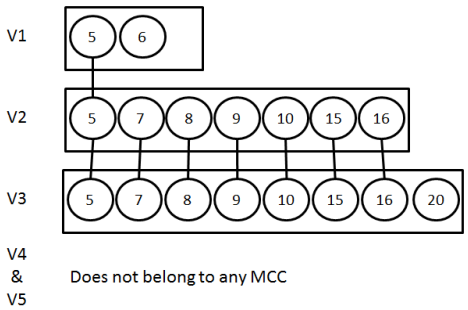

Fig. 2: PW MCC3 SIMCCT Trace

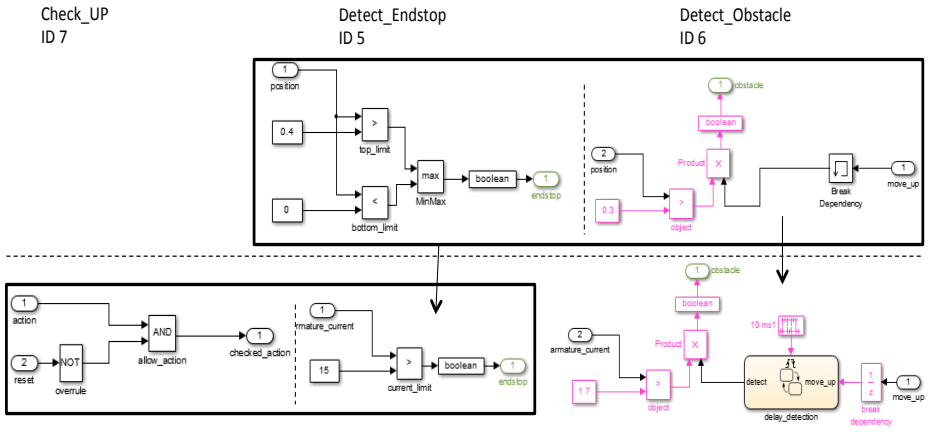

Fig. 3: Sample Models from PW MCC3

Power Window - Model Clone Class 2: We demonstrate, in Figs. 4 and 6, a "1-to-1*" variant where a single MCI is removed from MCC2 in version 1 while the remaining MCIs remain grouped together. In version 1, the three MCIs; 2,3, and 4; are 74% similar to each other with MCIs 3 and 4 being identical in this case. Fig. 6 shows the evolution of MCI2, the Window_System. As shown, it was changed significantly from version 1 to 2 in terms of its ports, the amount and types of blocks, and its lines. This was done in order to include power electronics and to incorporate bodies, joints, and actuators. It changed again in version 4, albeit it not as radically, but it was not enough to reunite it with the original MCIs from MCC2 from version 1.

Advanced Vehicle Simulator - Model Clone Class 7: Fig. 5 presents an example of a "1-to-many*" MCC trace. In version 1, MCI17, which is the system "Energy Storage <ess> RC", is 71% similar to MCIs 16,18,19. In the next version, MCI17 becomes 76% similar to MCI325 and is reclassified with that.

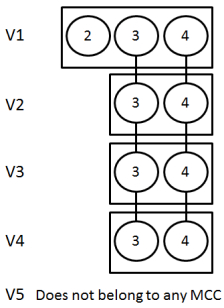

Fig. 4: PW MCC2 SIMCCT Trace

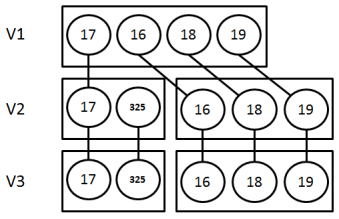

Fig. 5: AVS MCC7 SIMCCT Trace

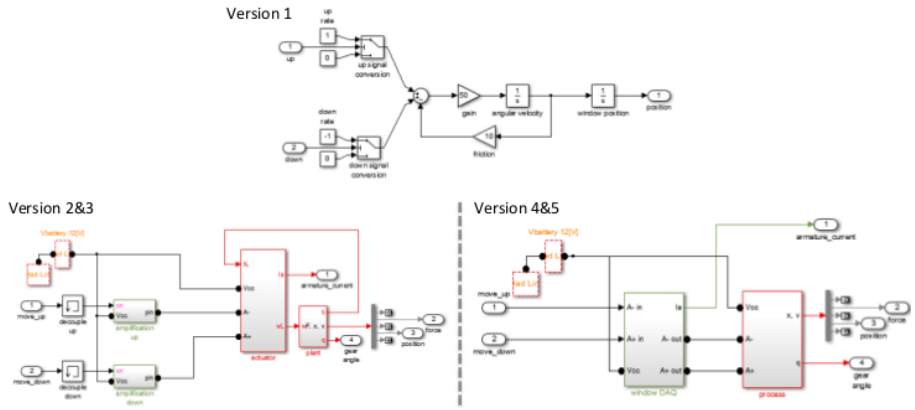

Fig. 6: Evolution of the PW Window_System

MCI17's evolution involved many low-level structural changes, and as such, is not worth showing a diagram of. So, while the model remained the same in terms of layout, there were some key non-visual changes including modification of block types, the addition or explication of ports, and changing a block's key parameters, for example, changing a Gain block's multiplication mode.

6 Related and Future Work

6.1 Related Work

As mentioned already, Saha et al. [14] are focused on counting the occurrences of code clone genealogies. In contrast, we are more interested in reasoning about the changes to models that cause MCE and use the genealogy of the MCCs as the starting point only.

There are some language-agnostic model and metamodel evolution approaches [7, 12] that can track both evolution and co-evolution. However, in order to use techniques like this for MCE, we would essentially have to create a system containing only the clones of interest. As such, we developed a tool explicitly intended to perform model clone class evolution analysis.

There are some model comparison approaches [16, 17] that can find similarities and differences among models for versioning and other purposes, but there are no attempts to explicate the structural evolution of *Simulink* models. That is, to define what are the potential structural changes that can occur to a *Simulink* model and their prevalence. Model evolution and MCE are strongly related to model comparison and versioning, but can be viewed as a longer-term analysis over multiple versions with a focus on how a specific artifact or clone has changed. None of the model comparison techniques we surveyed previously were ideal for tracking MCE.

The only work that deals with any form of *Simulink* evolution is from Tran and Kreuz, who focus on refactoring *Simulink* [11]. Specifically, they look at forms of antipatterns in *Simulink* and discuss tool support for correcting them.

6.2 Future Work

One area of future work is providing better differencing and visualization of differences for *Simulink* models. So far, we've been doing it relatively manually. While there are many model comparison tools [16, 17] that provide visualization functionality, there are none well-suited for our purposes and the provided *Simulink* XML comparison functionality is inadequate as it does not capture the information we desire. As such, devising and automating the differencing and visualization for MCE purposes and incorporating it into SIMCCT would be ideal.

In addition, we are currently working on MCD for other model types, including *Stateflow* and behavioral UML models [2]. We believe our work on MCE can be applied to other model types; Specifically, as long as an MCD tool identifies both MCCs and MCIs, these concepts can be extended and an appropriate clone class tracker can be developed. This is something that we will investigate once our MCD techniques for these other model types are more mature.

In the long term, we plan on enumerating a set of *Simulink* model evolutions as they relate to model clone evolution. The purpose in doing this is to find a sufficient set for performing MCD evaluation in a mutation-based framework as discussed in [15]. So, in addition to all the previously mentioned benefits of observing model evolution, we plan on using this work to help with mutation research intended to realize model-clone tool evaluation.

7 Conclusions

We believe MCE research is quite valuable as it can be a useful tool for better understanding how *Simulink* and other data-flow models evolve. In this paper, we took some first steps towards understanding *Simulink* MCE. We began by defining key terms, including model clone classes and instances. SIMCCT is a tool we introduced that is capable of tracking an MCI's evolution with respect to its containing MCCs across different system versions. We used this tool on three systems, share our findings, and go into details for a few examples. These examples included looking at the model evolution that transpired causing the specific MCE observed. In the future, we plan on automating the differencing and visualization of the model evolution for a given MCE trace as well as enumerating the *Simulink* model evolution steps that cause MCC changes. This and other *Simulink* and data-flow MCE work can go a long way towards improving our relatively underdeveloped understanding of the model evolution of these technologies.

Acknowledgments

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

References

1. Alalfi, M.H., Cordy, J.R., Dean, T.R., Stephan, M., Stevenson, A.: Models are code too: Near-miss clone detection for Simulink models. In: ICSM. pp. 295–304 (2012)
2. Antony, E., Alalfi, M., Cordy, J.: An approach to clone detection in behavioural models. In: WCRE. p. 5 (2013), (to appear)
3. Cordy, J.: The TXL source transformation language. Science of Computer Programming 61(3), 190–210 (2006)
4. Deissenboeck, F., Hummel, B., Juergens, E., Schaetz, B., Wagner, S., Girard, J.F., Teuchart, S.: Clone detection in automotive model-based development. In: ICSE. pp. 603–612 (2009)
5. France, R., Bieman, J.M.: Multi-view software evolution: a uml-based framework for evolving object-oriented software. In: ICSM. pp. 386–395 (2001)
6. Göde, N.: Evolution of type-1 clones. In: SCAM. pp. 77–86 (2009)
7. Herrmannsdoerfer, M., Benz, S., Juergens, E.: Cope-automating coupled evolution of metamodels and models. In: ECOOP 2009, pp. 52–76 (2009)
8. Keienburg, F., Rausch, A.: Using XML/XMI for tool supported evolution of UML models. In: HICSS. vol. 9, p. 9064 (2001)
9. Kim, M., Sazawal, V., Notkin, D., Murphy, G.: An empirical study of code clone genealogies. ESEC/FSE-13 30(5), 187–196 (2005)
10. Mens, T., Lucas, C., Steyaert, P.: Supporting disciplined reuse and evolution of UML models. UML98: Beyond the Notation pp. 378–392 (1999)
11. Minh Tran, Q., Kreuz, I.: Refactoring of simulink models. In: MathWorks Automotive Conference, Stuttgart (2012)
12. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.: Model migration with epsilon flock. In: Theory and Practice of Model Transformations, pp. 184–198. Springer (2010)
13. Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Tech. Rep. 2007-541, Queen's University (2007)
14. Saha, R.K., Roy, C.K., Schneider, K.A.: An automatic framework for extracting and classifying near-miss clone genealogies. In: ICSM. pp. 293–302 (2011)
15. Stephan, M., Alafi, M., Stevenson, A., Cordy, J.: Using mutation analysis for a model-clone detector comparison framework. In: ICSE. pp. 1277–1280 (2013)
16. Stephan, M., Cordy, J.R.: A survey of methods and applications of model comparison. Tech. Rep. 2011-582 Rev. 3, Queen's University (2012)
17. Stephan, M., Cordy, J.R.: A survey of model comparison approaches and applications. In: MODELSWARD (2013)