Electronic Communications of the EASST
Volume 63 (2014)

Proceedings of the
Eighth International Workshop on
Software Clones
(IWSC 2014)

Handling Clone Mutations in Simulink Models with VCL

— Position Paper —

Hamid Abdul Basit and Yanja Dajsuren

8 Pages

Guest Editors: Nils Göde, Yoshiki Higo, Rainer Koschke
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Handling Clone Mutations in Simulink Models with VCL

Hamid Abdul Basit[*] and Yanja Dajsuren[+]

[*]Syed Babar Ali School of Science and Engineering,
Lahore University of Management Sciences
hamidb@lums.edu.pk
[+]Department of Mathematics and Computer Science
Eindhoven University of Technology
y.dajsuren@tue.nl

Abstract: Like any other software system, real life Simulink models contain a considerable amount of cloning. These clones are not always identical copies of each other, but actually show a variety of differences from each other despite the overall similarities. Insufficient variability mechanisms provided by the platform make it difficult to create generic structures to represent these clones. Also, complete elimination of clones from the systems may not always be practical, feasible, or cost-effective. In this paper we propose a mechanism for clone management based on Variant Configuration Language (VCL) that provides a powerful variability handling mechanism. In this mechanism, the clones will be managed separate from the models in a non-intrusive way and the original models will not be polluted with extra complexity to manage clone instances. The proposed technique is validated by creating generic solutions for Simulink clones with a variety of differences present between them.

Keywords: Model Clones, Simulink, Variability, Clone Management

1 Introduction

Model-based development with Simulink is the state-of-the-art technique in several embedded system domains, where the functionality is specified using models and code is automatically generated from these models. Sometimes, these models are comprised of thousand of elements and are maintained over long periods by the organizations. In these situations, cloning becomes a relevant problem [1]. Separately maintaining multiple similar parts of models could increase costs, and inconsistent changes to cloned parts could lead to incorrect or undesired system behaviour.

Completely eliminating clones from the systems may not always be practical, feasible, or cost-effective. A viable alternative is to perform clone management. This is especially relevant for product line based development of similar systems, where clones represent reusable pieces of functionality, and their integration in a central repository is a basic task for the development of product lines.

To be viable, clone management techniques require a representation of the clones that could provide a powerful parameterization mechanism to capture all kinds of variations that could possibly exist between clone instances, and is robust to evolutionary changes. Model variants exhibit a range of differences. Similar to code clones, type-1 type-2 and type-3 model clones have also been identified [1]. Model clones can also occur across multiple layers [1]. A challenge is to unify all types of possible differences between model clones with a single variability management technique.

So far, no clone management technique has been proposed for Simulink models. A basic requirement for clone management is a powerful variability management technique. Simulink provides a basic variability mechanism with its Variant blocks but this mechanism is only meant for simulation and increases the size and complexity of the models. Another option is to model variability with general-purpose blocks like Switch blocks or if-action blocks for the selection of alternative variants [11][8]. In addition to the above drawback, another problem here is that it is not obvious whether a Switch block is for the selection of alternative variants or to control the signal flow. There is no possibility to remove unnecessary variability information and reduce a variant-rich model to a specific system model [9]. This results in the intermixing of functionality and variability handling mechanisms in the models, violating the principle of separation of concerns [11].

We propose a clone management framework for Simulink models based on Variant Configuration Language (VCL) [13]. VCL provides unrestricted parameterization of text-based artefacts. As Simulink also provides an equivalent textual representation of its models, this does not pose a limitation for the proposed solution. VCL based solution is non obstructive as the variability handling concern is addressed separately from the functionality concern. Due to the powerful parameterization of VCL, we can define variants capturing any kind of differences that could be present in clones. Finally, with VCL we can also define new variation points to a generic structure without affecting the existing variants.

In our proposal, we define separate roles for model developers and model managers with regard to clone management to clarify the description of the process, although the same person can fulfil these roles at different times. VCL based generic clone representations are developed by model managers and stored in a clone repository. Colours are used to tag the different parts of the Simulink models for cloning status and these tags are updated when the cloning status is changed. Finally, we validate our technique by creating generic representations of a number of clones exhibiting a variety of differences.

The rest of the paper is organized as follows: In section 2 we discuss related work and in section 3 we describe the details of our proposed approach for model clone management. In Section 4 we validate our approach. Section 5 concludes the paper and discusses the future work.

2 Related Work

Leitner et. al. [9] uses *pure::variants Connector for Simulink* to handle structural variability in Simulink models. They identify common variability scenarios from the industry, and propose a 3-layered template based mechanism to abstract the variability implementation. Like our approach, they also hide variability mechanism from the developers. However, we do not need any extra blocks to achieve this goal.

Different clone management techniques have been proposed for code-based software systems. Toomim et al. [12] attempts to keep consistency among clone members by linked editing of the clone members. Baxter et al. [3] eliminates clones automatically using macros. Recently, solutions are being searched for manual clone management instead of a fully automatic refactoring tool, as the elimination of clones may not always be viable. Duala-Ekoko et al. [5] use a descriptive language to help track of clones over software evolution. However, this description language is specific to object-oriented languages like Java and C++.

3 Approach

Our approach is based on the Variant Configuration Language (VCL) [13], which offers a flexible and user-defined syntax. It extends the capabilities of the basic C preprocessor (cpp) to better manage software variability. VCL organizes and instruments the base code for ease of adaptation and reuse during development and evolution of system variants. VCL allows instrumenting the source code for customization at any level of details. VCL Processor goes through the base code, executing VCL commands to generate a required system variant. VCL parameters exercise control over the VCL processing separately from the base code. This makes VCL a well-thought out mechanism that is simple, powerful and fully automated.

In our proposal, we differentiate between the roles of model developers and model managers (similar to the role of frame engineers proposed by Bassett [2]). The same person could be playing both roles, but for the sake of identifying relevant responsibilities, we describe the proposal in terms of these distinct roles.

We start with a given set of clones identified in a Simulink model or a group of models. There are tools available for detecting clones in Simulink models [1][6]. SIMONE [1] works with the textual representation of Simulink models and reports clones in the form of clone classes and clone pairs. It detects not only type-1 and type-2 clones but also reports type-3 clones. SIMONE reports only similar subsystems as clones. ConQAT [6] detects clones using a graph matching technique and also reports clones at the block level. However, it only detects type-1 and type-2 clones. Our proposed technique can work with the output from any of these tools, but subsystem level clones gives a more crisp boundary for a reusable element in Simulink, and hence is preferred by our approach.

The responsibility of clone detection lies with the model manager role. From the detected clones, the model manager will decide which clones need to be managed. Various clone related metrics can be used to identify clones that are of importance to the developers and should be consistently maintained [7]. These selected clones are manually converted into VCL representation and placed in a clone repository. In the actual Simulink models, we mark each subsystem that is generated from a VCL managed clone by a unique color, as in [9].

As discussed in [10] and [1], blocks, lines, ports and branches could be reordered in the textual representation of type-1 model clones. For SIMONE, these elements are sorted before clone detection [1]. For our proposed solution, we can safely sort these elements in the generic representation with VCL. Even though, we can generate the exact ordering of these elements for each clone instance, as it was before sorting, but we do not need this extra complexity in the generic representation as the sorted and unsorted subsystems would be functionally and graphically equivalent in the resulting Simulink model. Figure 1 shows the workflow for model manager role.

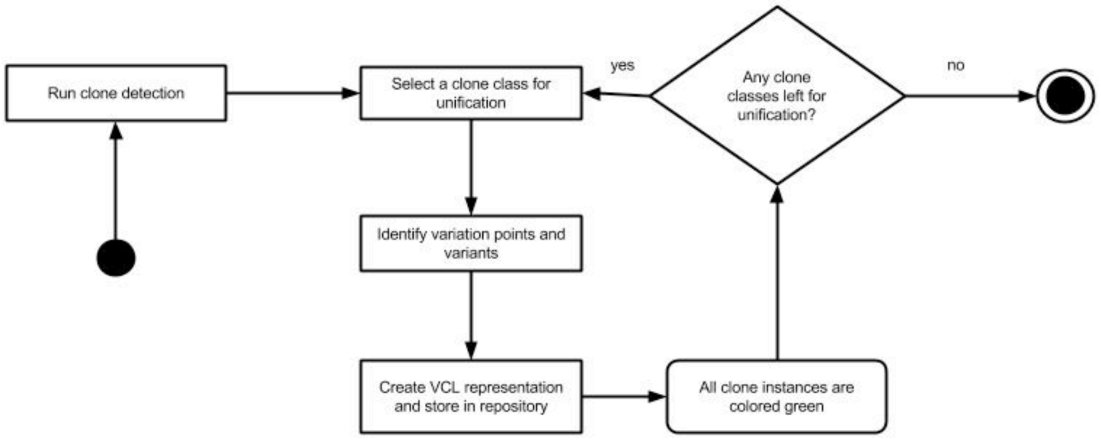
Run clone detection
Select a clone class for unification
yes
Any clone classes left for unification?
no
Identify variation points and variants
Create VCL representation and store in repository
All clone instances are colored green

Figure 1. Flow of activities for the Model Manager

When a developer (Figure 2) needs to reuse a subsystem from another part of the model or from another model, there could be two possibilities. Either this subsystem is an existing clone that has a VCL representation in the repository or it is the first cloning of a subsystem. In the former case, the developer would generate the configured clone from the VCL representation of the managed clone in the repository for the new use, while marking this new copy accordingly. For the latter case, the developer will clone the existing subsystem and reuse in the new place. In this case also, the two copies will be marked with another unique color indicating the presence of a clone that has not yet been stored in the repository. This marking will be a hint for the model manager to create VCL representation of this new emerging clone with suitable variation points and variants.

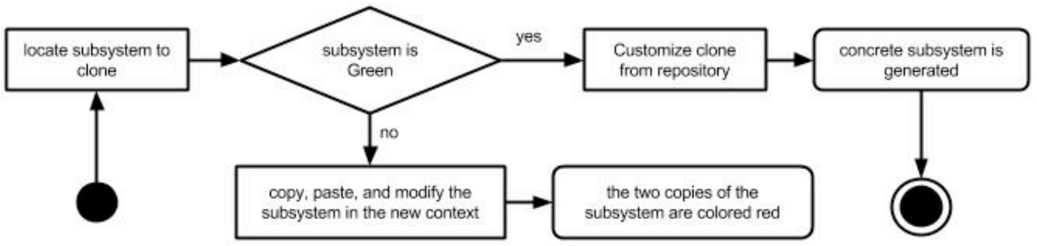
locate subsystem to clone
subsystem is Green
yes
Customize clone from repository
concrete subsystem is generated
no
copy, paste, and modify the subsystem in the new context
the two copies of the subsystem are colored red

Figure 2. Flow of activities for Model Developer

When the developer selects a clone from the repository for reuse, she will be presented with a list of variation points for the selected clone, and a list of previously existing variants to choose from for each variation point. The developer role can only configure the new clone instance selecting from these predefined variants for a variation point and is not allowed to create new variants or new variation points. For every variation point, there will be a default variant. When the developer is done with the configuration, a concrete subsystem is generated based on the developer's selection of variants and the developer can now use this block where required.

If the developer modifies a generated copy of a clone in ways other than those captured by the VCL representation, the block is marked with a third unique color. The model manager will later analyze these clones for the extent of changes made to them. If the changes are few, this instance could be merged with the generic version in the repository by defining new variation points or new variants for the existing variation points. Due to the flexibility provided by VCL, there are almost no restrictions on the variants that could be provided at a variation point. However, if the new changes have made this copy significantly different from the original clone, the model manager can also choose to remove this particular instance from the clone class completely.

4 Validation

The most important aspect of the proposed mechanism is to effectively handle the wide range of variability that could possibly be present in the clones. To validate the feasibility of VCL for capturing all kinds of variations, we created numerous Simulink subsystems forming clone pairs. Each clone pair captures only one form of variation. Using VCL commands, we created generic solutions for each of these clone pairs. Overall, we captured all possible forms of variations listed by Stephan et. al [10]. These include:

- different layout (color, position, size, other attributes) of elements
- different ordering of elements (blocks, lines, ports, branches)
- different names of elements (blocks, lines)
- different values of elements (blocks)
- added or deleted block
- changed block type

A concrete example of a clone pair's generic subsystem annotated with VCL, and its configuration to regenerate the original subsystems, is shown in Figure 3.

```
System {
          Name          "Subsystem"
          Location      ?@subsystem_location?
          …
          Block {
                    BlockType  Sum
                    …
                    Position   ?@sum_position?
          }
          Block {
                    BlockType  Outport
                    …
                    Position   ?@outport_position?
          }
          Line {
                    …
          #break line1_points
                    DstBlock   "Add"
                    DstPort    2
          }
          Line {
                    SrcBlock   "Add"
                    SrcPort    1
          #break line2_points
                    DstBlock   "Out1"
                    DstPort    1
          }
}
```

(a)

```
#set subsystem_location = "[596, 16, 1412, 554]"

#set sum_position = "[240, 80, 270, 120]"

#set outport_position = "[355, 93, 385, 107]"

#output "Layout2a_Position.mdl"

#adapt "Layout2a_Position"

#set subsystem_location = "[3, 15, 819, 553]"

#set sum_position = "[245, 135, 275, 175]"

#set outport_position = "[360, 103, 390, 117]"

#output "Layout2b_Position.mdl"

#adapt: "Layout2b_Position"

          #insert line1_points
          Points      [47, 0; 0, 55]
          #endinsert

          #insert line2_points
          Points      [37, 0; 0, -45]
          #endinsert

#endadapt
```

(b)

Figure 3. (a) Annotating the subsystem with VCL. (b) Configuring the subsystems

5 Conclusions and Future Work

In this paper we have proposed a clone management framework for managing Simulink model clones. The benefits of using VCL as the variability technique includes separating the variability concern from the functionality concern. The variability mechanism has been validated by converting a number of clone pairs with a varied set of differences into generic representations of VCL.

We are working on the development of a prototype tool based on this proposal. In addition to the clone detection tool, we also need a clone-matching tool whereby we can search for other copies of a known clone in the newly developed parts of a model. In future we can give visual rendering to VCL frames like *pure:variants VAR_Multiport Switch* and *VAR_Const* [9]. In this manner, cloned subsystems will have extra property pages at configuration time to concretize the block/subsystem before using it. Further, empirical evaluation of the approach is considered as future work.

Acknowledgements

The first author was partially funded by Erasmus Mundus IDEAS project. The second author is supported by the Dutch Automotive Innovation Programme (HTAS) project (HTASI10002).

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 Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), 2012 pp. 295 - 304, 2012, Trento, Italy.

[2] Bassett, P., Framing Software Reuse: Lessons From the Real World, Yourdon Press, Prentice Hall, 384 pages, 1996

[3] Baxter, I. D., Yahin, A., Moura, L., Sant’Anna, M., and Bier, L., “Clone detection using abstract syntax trees”. In Proceedings of the International Conference on Software Maintenance (ICSM), Washington, DC, USA, 1998. IEEE Computer Society.

[4] Cordy, J.R., “Submodel Pattern Extraction for Simulink Models”, In Proceedings of the Software Product Line Conference, SPLC, 2013, Tokyo, Japan.

[5] Duala-Ekoko, E., and Robillard, M. P., “Clone region descriptors: Representing and tracking duplication in source code”. In ACM Transactions on Software Engineer- ing and Methodology, 20(1), pages 1–31. ACM, June 2010.

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

[7] Deissenboeck,F., Hummel, B., Jurgens, E., Pfaehler, M., Schatz, B., “Model clone detection in practice”. In Proceeding of the 4th International Workshop on Software Clones, IWSC '10, pp. 57-64, 2010.

[8] Haber, A., Kolassa, C., Manhart, P., Nazari, P. M. S., Rumpe, B., Schaefer, I., “First-Class Variability Modeling in Matlab/Simulink”, In Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems (VaMoS’13), pp. 11-18, 2013.

[9] Leitner, A., Ebner, W., and Kreiner, C., “Mechanisms to Handle Structural Variability in MATLAB/Simulink Models”, J. Favaro and M. Morisio (Eds.): ICSR 2013, LNCS 7925, pp. 17–31, 2013.

[10] Stephan, M., Alalfi, M. H., Stevenson, A., and Cordy, J. R., “Using Mutation Analysis for a Model-Clone Detector Comparison Framework”, In Proceedings International Conference on Software Engineering (ICSE 2013), New Ideas and Emerging Results (NIER) Track, San Francisco, CA, USA.

[11] Schulze, M., Weiland, J., Beucho, D., “Automotive Model-Driven Development and the Challenge of Variability”, SPLC '12, September 02 - 07 2012, Salvador, Brazil

[12] Toomim, M., Begel, A., and Graham, S. L., “Managing duplicated code with linked editing”. In Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC ’04), pages 173–180, Washington, DC, USA, 2004. IEEE Computer Society.

[13] VCL. Variant Configuration Language. http://vcl.comp.nus.edu.sg. [19 November 2013]