

Automated Transformation of MATLAB, Simulink and Stateflow Models

Ingo Stürmer¹, Dietrich Travkin²

¹Model Engineering Solutions, Berlin, stuermer@model-engineers.com

²University of Paderborn, travkin@uni-paderborn.de

Abstract

Model-based development and automatic code generation have become an established technology in the controller design process. Numerous modeling guidelines are build to enable or improve code generation, raise code efficiency and to ease model comprehension. Manually checking guideline conformance of complex models as well as manually eliminating guideline violations can be error-prone and laborious. In this paper we present an approach which provides developers with tool support in analysis and improvement of MATLAB, Simulink and Stateflow models. Developers have the possibility to automatically detect and interactively eliminate guideline violations. Furthermore, they get support in creating guideline compliant models, e.g. by instantiating modeling patterns.

1. Introduction

Nowadays, model-based development is common practice within a wide range of automotive embedded software development projects. In model-based development, de facto standard modeling and simulation tools such as MATLAB, Simulink and Stateflow are used for specifying, designing, implementing, and checking the functionality of new controller functions. The quality and efficiency of the software are strongly dependent upon the quality of the model used for code generation. For that purpose best practices are provided by usually adopted modeling guidelines such as the *MathWorks Automotive Advisory Board (MAAB)* guidelines [3].

There are tools available to help the modeler check a model with regard to guideline conformance, such as the Simulink *Model Advisor* [1] or *MINT* [2]. The Model Advisor, for example, assists the developer in reporting violations of block settings, model configurations, or modeling styles that do not comply with such guidelines.

However, for huge controller models, this can add up to a few hundreds, or even a few thousands, violations that must be corrected manually by the modeler. That is a cumbersome, complex, and expensive task.

A recent in-house case study at DaimlerChrysler showed us, that up to 80% of guideline violations can be fixed fully or semi-automatically, i.e. with some user interactions (ref. [6] for details). For that reason we developed the MATLAB, Simulink and Stateflow Analysis and Transformation Environment (MATE). This environment also supports in detecting previously specified guideline violations, but additionally – compared to the Model Advisor or MINT - provides the possibility to automatically apply predefined model transformations, e.g. repair operations, layout improvements, and modeling pattern instantiations. MATE is fully integrated into the MATLAB, Simulink and Stateflow environment.

The rest of the paper is structured as follows: In section 2 we motivate the usage of modeling guidelines and emphasize the necessity of tool support in analysis and repair of models. In section 3 we describe typical model transformations provided by MATE. The specification of model transformations is depicted in section 4. Finally conclusions are drawn in section 5.

2. Modeling Guidelines

An agreement to follow certain modeling guidelines is important to increase the comprehensibility (readability) of the model, facilitate maintenance, ease testing, reuse, and extensibility, and simplify the exchange of models among OEMs and suppliers. This is the purpose of the MAAB guidelines and patterns [3]. Following the modeling conventions stated in those guidelines can support the translation of a model into proper code.

Such publicly available guidelines are often supplemented by in-house sets of modeling guidelines in order to check the models used for production code

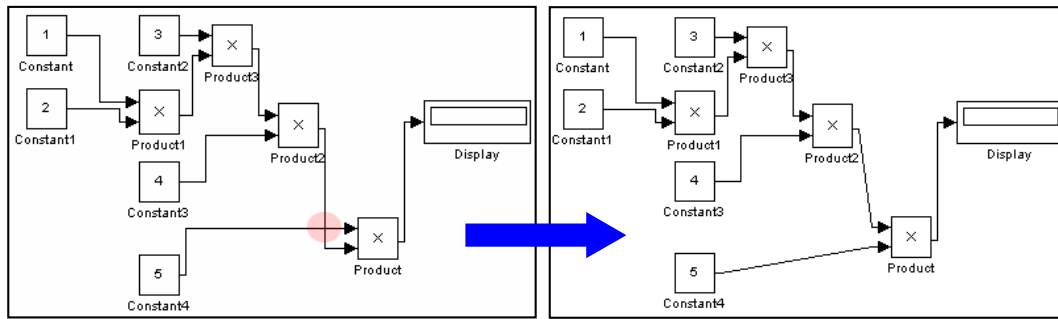


Figure 1: Elimination of Line Intersections by Changing Port Order

generation. For example, these guidelines can improve the efficiency of the generated code. The Model-based development guidelines of DaimlerChrysler consist of more than 200 modeling rules and patterns. These guidelines are published via the e-Guidelines Server [4], a web-based infrastructure for publishing and centrally administering different sets of guidelines.

The huge number of modeling rules necessitates models to be checked for guideline compliancy by means of automated tools, since pure manual model reviews are laborious and prone to error. The same holds for the manual elimination of guideline violations. Furthermore, few reviewers would be capable of keeping all the modeling rules to be checked in mind.

Several tools like the Simulink Model Advisor [1] or MINT [2] offer frameworks for executing M-scripts, which can be used to check guideline conformance. MATE in contrary provides the possibility to *graphically* specify guideline violation patterns, which can be automatically detected in a model (see [6] for more details). This allows to specify model analysis and repair functions on a higher level of abstraction, compared to implementation-specific M-scripts used by the Model Advisor or MINT (cf. also [7]).

In a recent in-house case study at DaimlerChrysler, we used the MINT framework in order to check a typical but huge controller model from the automotive domain. The emphasis of this case study was to (a) evaluate the maturity of the model with regard to production code generation and (b) to check the model concerning guideline compliancy. It turned out that the model contained more than 2,000 guideline violations. After closer inspection, we estimated that up to 45 percent (900) of these 2,000 rule violations could have been fixed automatically, if tools such as MATE had been used [6]. Furthermore, we estimated that approximately 40 percent could have

been repaired with user feedback (interactive repair operations). Only 8 percent actually required a manual correction. A final 4 percent remained undefined, which meant that the modeler had to determine whether the reported violations really infringed upon a modeling guideline.

To sum up: up to 80% of guideline violations can be fixed automatically or with interactive repair functions. For that reason, MATE shifts the focus from pure model analysis to high-level model repair functions, i.e. model transformation.

3. Model Transformations

MATE provides different kinds of operations that modify a MATLAB, Simulink and Stateflow model:

After detecting guideline violations during a model analysis phase, all applicable, pre-defined *repair operations* are suggested to eliminate the violations reported. For example, product blocks with more than two operands can automatically be transformed into a cascade of product blocks with two operands [7]. If the Simulink model is to be used for fixed-point code generation, this modification is necessary to determine scaling informations for intermediate results. Since most guideline violations can be fixed fully automatically or at least with few user interactions, such repair operations can reduce the effort of model refactoring/repairing enormously.

While a developer is modeling a system he/she has the opportunity to apply “*beautifying*” *model transformations* via a context menu. For example, it is possible to align Simulink blocks horizontally or vertically to each other or to avoid line intersections by changing port order (see Figure 1, crossing signal lines of *Product* block). Such model transformations improve the arrangement of the elements in a model and thus ease model comprehension without significantly delaying the modeling process.

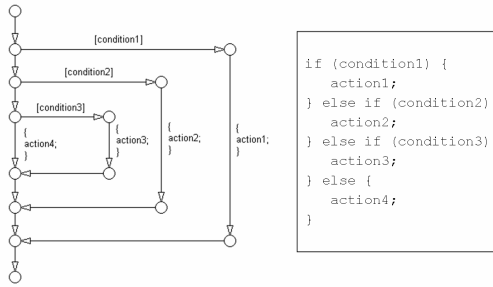


Figure 2: If-then-else modeling pattern (left) and corresponding pseudo-code (right)

Guidelines like the modeling guidelines of DaimlerChrysler [4] or the MAAB guidelines [3] usually contain best practice modeling patterns for recurring modeling tasks, e.g. patterns for if-then-else or switch-case constructs in Stateflow diagrams. Figure 2 (left) illustrates a typical pattern to be used when modeling control structures such as the one specified by the pseudo-code on the right-hand side of Figure 2. The *instantiation of modeling patterns* can be very time-consuming, e.g. when the nesting deepness of the control structure increases. MATE speeds up this process and helps developers in complying with such guidelines by providing the possibility to instantiate modeling patterns by a few clicks. For example, an if-then-else pattern complying to the one from Figure 2 can automatically be instantiated using a

pre-defined transformation operation. Therefore, the developer only has to select the desired operation via a context menu and to set the number of desired decision branches in a pop-up dialog. He/she no longer has to create and align junctions and transitions manually to reach the required structure and layout. Only the conditions and actions have to be specified manually. The highlighted portion of Figure 3 illustrates the instantiated if-then-else pattern.

4. Specifying Model Transformations

MATE is a framework which exemplarily provides some model transformation operations but primarily offers the possibility to specify and apply new ones.

There are two ways to specify a model transformation operation: implementing Java code or graphically define graph transformation rules. In both cases the Simulink and Stateflow model is represented by a so called *abstract syntax graph*, which the transformations are applied to.

The abstract syntax graph is an object structure. Each object represents an element in a Simulink or Stateflow diagram (concrete syntax). The links between objects represent element relations, e.g. containment. Figure 6 illustrates a part of an abstract syntax graph representing the two product blocks and a connected constant block shown in the Simu-

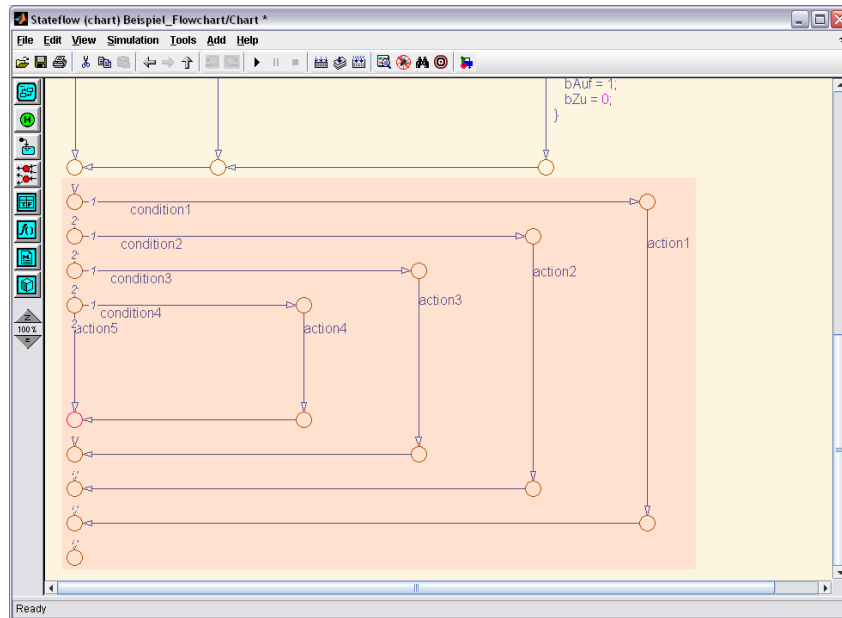


Figure 3: Instantiated if-then-else modeling pattern (highlighted with colored background)

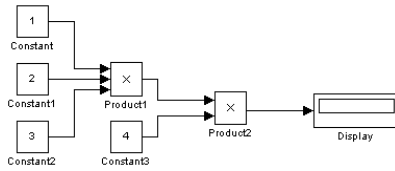


Figure 4: Concrete Syntax Example

link diagram from Figure 4. Each product block in the Simulink diagram is represented by an object of type *Product*, each signal line by a *Line* object, etc. Attributes *name*, *type*, *value* contain additional information about the elements.

A transformation in MATE modifies the abstract syntax graph, i.e. it creates or removes objects, adds, removes or re-connects links, or changes attribute values.

Figure 7 exemplarily illustrates a graphical specification of a model transformation. The transformation is specified using so called story diagrams, a kind of UML activity diagrams with nested collaboration diagrams. Given a *Product* object, i.e. a product block, as a starting point, this transformation tries to find the specified object structure in the abstract syntax graph, then removes all the elements marked with «destroy» and adds all the elements marked with «create». In this case the transformation replaces product blocks with more than two operands by a cascade of product blocks with only two operands.

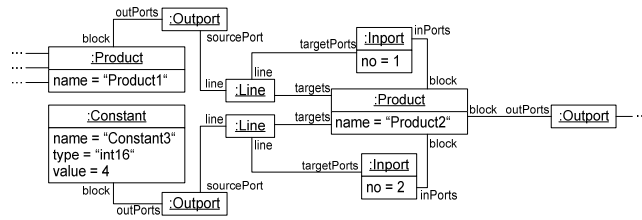


Figure 6: Abstract Syntax Graph Example

5. Conclusion

The adoption of modeling guidelines for the design of automtotive controller models is important. The huge number of modeling rules and patterns that must be kept in mind by the developer requires tool-support in order to check controller models with regard to guideline compliancy. However, the also huge number of violated guidelines makes the model rework cumbersome and prone to error. MATE supports the developer in analyzing MATLAB, Simulink and Stateflow models and automatically or interactively transforming such models into guideline-compliant ones. Apart from that, MATE provides high-level editor functions, which help the developer to design guideline-compliant models by generating pre-defined and configurable modeling patterns or using model beautifier operations.

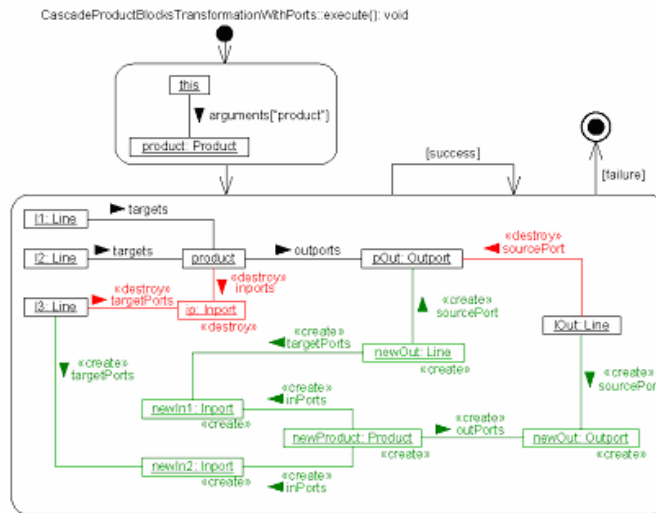


Figure 7: Specification of a Model Transformation

References

- [1] The MathWorks:
<http://www.mathworks.com/products>, 2006.
- [2] Ricardo, Inc., MINT
<http://www.ricardo.com/mint>, 2006.
- [3] Mathworks Automotive Advisory Board (MAAB): *MAAB Controller Style Guidelines for Production Intent*. Release V2.0, The Mathworks, Inc., 2007.
- [4] Model Engineering Solutions:
<http://www.e-guidelines.de>, <http://www.model-engineers.com/products.htm>, 2007.
- [5] Ingo Stürmer, Mirko Conrad, Ines Fey, I., Heiko Dörr: Experiences with Model and Autocode Reviews in Model-based Software Development. *Proceedings of Third International ICSE Workshop on Software Engineering for Automotive Systems (SEAS'06)*, pp. 45-51, 2006.
- [6] Ingo Stürmer, Ingo Kreuz, Wilhelm Schäfer, Andy Schürr: The MATE Approach: Enhanced Simulink[®] and Stateflow[®] Model Transformation, *Proc. of MathWorks Automotive Conference (MAC)*, 2007.
- [7] Ingo Stürmer, Heiko Dörr, Holger Giese, Udo Kelter, Andy Schürr, Albert Zündorf: Das MATE Projekt—visuelle Spezifikation von MATLAB/Simulink/Stateflow Analysen und Transformationen, *Proc. of Dagstuhl Workshop „Modell-basierte Entwicklung eingebetteter Systeme III“*, MBEES 2007, pp. 83-93, 2007.