Views for Efficient Program Understanding of Automotive Software

Jochen Quante

Robert Bosch GmbH
Corporate Sector Research and Advance Engineering Software
P. O. Box 30 02 40, 70442 Stuttgart, Germany

Jochen.Quante@de.bosch.com

Abstract

Automotive Software is often developed using graphical notations, such as ASCET or Matlab Simulink models. This means that also different approaches to program comprehension have to be developed. In this paper, we present the results of a research project that developed and evaluated different views on such models.

1 Introduction

Program comprehension is always a challenge, except for very simple programs. According to Lehman and Belady [5], 40% of the entire software lifecycle costs account for program comprehension. This is not only true for textual programming languages, but also for model-based development. Therefore, program comprehension support is also desirable for model-based languages.

In a first attempt to better support program comprehension, we asked the community to demonstrate their tools on a problem of our domain [1]. Unfortunately, the results were not convincing [6], so we had to come up with our own solutions which are introduced in the following.

2 Views for Program Comprehension

In architecture descriptions, *views* [4] are used to focus on different concerns of the same system. The same concept can be used on implementation level. A given piece of software or model can only express a certain amount of aspects. Almost always, there are other aspects that stay hidden and are therefore hard to comprehend. If we change the software to emphasize these hidden aspects, others will be neglected. Views are a means to solve this problem.

This is also true for model based software development. A model only focuses on some aspects. For example, ASCET[1] focuses on data flow and neglects control flow. This makes it very hard to understand the control flow of an ASCET model, although it is still important. The same is true for data flow in a C function.

In order to identify the most beneficial views for the automotive domain, we first set up a collection of views. Beside obvious views to visualize the *control flow*, we identified domain specific categories like a *variant view* which highlights one variant of a product line software. We found the following four categories of views:

Scope reducing views Those parts of the software that are irrelevant in the current scope are faded out. The user's focus is directed to the relevant information.

- *Variant views* hide inactive code. This is helpful when developing a software product line. For example, the Eclipse CDT[2] greys out code that is inactive due to the current macro definitions.
- *Context aware views to simplify computations* use calibration data to simplify code. For example, if a parameter is set to zero, a following addition can be omitted, or a multiplication can be substituted with zero.
- *Context aware views to simplify control flow* use fixed values (fixed parameters in a function call or fixed through calibration) to fade out code with no impact. For example, if a condition evaluates to `false`, the `true` branch is not relevant.

Structural views show only the structure of a given part of the software. This is particularly useful for navigation and orientation. Considered structures were *composition structure* (directories, files, functions, etc.) and the *call graph*.

Data flow views help to track data flow, in particular in software that is specified in a control flow centric way. But also for data flow oriented ASCET models, such views can be very helpful for tracking data flows through a system – especially when there are additional hidden dependencies.

[1] `http://www.etas.com/en/products/ascet_md_modeling_design.php`

[2] `http://www.eclipse.org/cdt/`

Control flow views help to understand the sequence of execution, in particular in data flow oriented representations such as ASCET. Examples are *control flow graphs*, *Nassi Shneiderman diagrams*, and the *ASCET sequence view* (see below).

The complete list consists of about 20 views. Based on this list and corresponding examples, we questioned function developers, team leaders, and calibration engineers about their potential usefulness. Based on the results of this survey, we prioritized the view candidates and realized the two views with the highest expected benefit with respect to their specialties in the automotive domain. One of them is introduced in the following.

3 Concrete View: Signal Flow

To comprehend the physical interrelationships between values at certain points in the software, the data flow between these points is very important. Therefore, a view that shows these dependencies is of great interest, especially for calibration engineers. They have to set values for engine specific parameters in the software to derive a specific variant from a product line software. Therefore, they have to understand the physical effect chain which is represented in the software. For example, they want to know which of the parameters affect which output variable.

A technique that is intended for calculating the flows through a function is *slicing* [9, 2]. For a given variable at a given program point, slicing can find out which other parts of the program are affected by that variable, or which parts of the program influence that variable. It can also determine the dependencies between two program points, which is called *chopping* [3]. These techniques answer exactly those questions that application engineers have when working with our software models. Therefore, our signal flow view is a realization of slicing and chopping for ASCET.

In Figure 1, the result of this analysis on an ASCET model is presented. This example shows one specialty of ASCET. Because one can have multiple occurrences of variables in one picture, data flow analysis has to derive dependencies that were not present as data connections in the original picture[3]. This means the data flow model may be incomplete prior to data flow analysis.

Recently, a very similar approach has successfully been used for understanding signal flows through a component-based embedded system [10]. Another recent publication [7] shows how slicing can be applied to MATLAB Simulink models. Along with the positive feedback from our engineers, this is another indicator that this technique is well suited for easier understanding of models of embedded software.

[3]This is the standard case for control flow oriented languages, but not for data flow oriented languages.

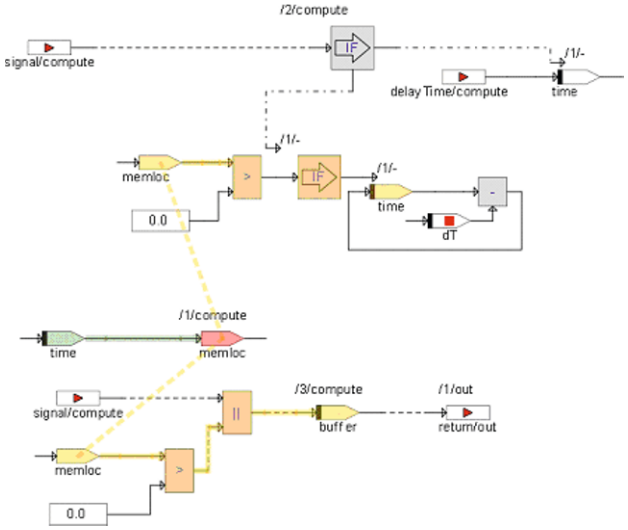

Figure 1: Signal Flow View for ASCET.

4 Outlook

Since it turned out that specific questions require specific views, we are now about to develop a general general analysis framework for C code and ASCET models along with a number of standard views. The idea is that one can easily construct individual new views based on that. Another consequence of this research project is the initiation of the development of a tool that provides interactive navigation and viewing of software documentation. The presented view is a part of that.

References

[1] A. Begel and J. Quante. Industrial program comprehension challenge 2011: Archeology and anthropology of embedded control systems. In *Proc. of 19th ICPC*, pages 227–229, 2011.

[2] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM TOPLAS*, 9(3):319–349, 1987.

[3] D. Jackson and E. J. Rollins. Chopping: a generalization of slicing. Technical report, Carnegie Mellon University, CS-94-169, July 1994.

[4] P. Kruchten. Architectural blueprints – the "4+1" view model of software architecture. *IEEE Software*, 12(6):42–50, 1995.

[5] M. M. Lehman and L. A. Belady. *Program evolution: processes of software change*. Academic Press Professional, Inc., 1985.

[6] J. Quante. When program comprehension met bug fixing. *Softwaretechnik-Trends*, 32(2), May 2012.

[7] R. Reicherdt and S. Glesner. Slicing MATLAB simulink models. In *Proc. of 34th ICSE*, 2012.

[8] A. Thums and J. Quante. Reengineering embedded automotive software. In *Proc. of 28th ICSM*, pages 493–502, 2012.

[9] M. Weiser. Program slicing. In *Proc. of 5th ICSE*, pages 439–449, 1981.

[10] A. R. Yazdanshenas and L. Moonen. Tracking and visualizing information flow in component-based systems. In *Proc. of 20th ICPC*, 2012.