# An Approach to Reverse Engineering of *C Programs* to *Simulink Models* with Conformance Testing

Indranil Saha
Computer Science Department
University of California, Los Angeles
CA 90095, USA
indranil@cs.ucla.edu

Kuntal Chakraborty
HTS[*]Research
151/1, Doraisanipalya, Bannerghatta Road,
Bangalore 560 076, India
kuntal.chakraborty@honeywell.com

Suman Roy
SETLABS[†] Infosys Technologies Ltd.
44 Electronics City, Hosur Road,
Bangalore 560 100, India
suman_roy@infosys.com

B VishnuVardhan Reddy
Indian Statistical Institute
203, B.T. Road
Kolkata 700108, India
vishnuhcu@gmail.com

Venkatappaiah Kurapati
Indian Statistical Institute
203, B.T. Road
Kolkata 700108, India
venkatappaiah@gmail.com

Vishesh Sharma
Dept. of CSE
Indian Institute of Technology
New Delhi 110 016, India
vishesh_iitd@yahoo.co.in

## ABSTRACT

To reuse legacy C code effectively in Model Based Development process, it is highly desirable that the code be converted to Simulink model, a de facto standard in many industrial application domains, such as avionics and automotive control. In this paper we present the design methodology of a tool that translates a C code to equivalent Simulink model with proper correctness assurance.

## Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Reverse engineering*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering*

## General Terms

Design, Experimentation, Verification

## Keywords

C, Simulink, Reverse Engineering, Conformance Testing

## 1. INTRODUCTION

Recently Avionics and Automotive industries are leaning more and more towards Model Based Development (MBD) of safety-critical software. In embedded system community, MBD implies development of control models and simulations of those models to ensure their correctness using tools such as Matlab and SCADE. The auto code generators associated with these tools can generate software codes from the model, thus enabling control engineers to generate embedded software automatically without knowing the target language syntax. Simulink is a high-level model design tool which is very popular in many industrial application domains. It is considered as a *de facto* standard in avionics and automotive domains. Avionics and Automotive industries are already in possession of large legacy code bases developed in C programming language, which were not developed following MBD processes. Realizing the benefit of adhering to MBD in their software development life-cycle, these industries are now interested in embracing MBD approaches in their new projects; thereby having models for all of their old legacy C codes for the ease of reuse and maintenance.

In this paper, we describe an approach to build a tool which will be useful to translate legacy C code to an equivalent Simulink model containing only Simulink library blocks. We also provide a mechanism to ensure the correctness of such a translation. Zsombori of DaimlerChrysler has built a translator to convert a C code to a Simulink model [6] by using JavaCC (Java Compiler Compiler). The shortcoming of the work is that the tool does not follow any industrial guideline while generating a Simulink model. Organization of the generated Simulink model in terms of localization and positioning has been the major issue in this work, whereas in our work we strictly follow MAAB guidelines [5] for organization of the output Simulink model. Moreover, in Zsombori's work, handling of function calls and arrays have been identified as open problems. In our work, we provide complete solutions for both of them. For large legacy code, handling function calls is of prime importance. We show how function level model generation can be done by properly resolving the issues on local and global variables, and calling a model from another model.

## 2. DESCRIPTION OF THE TOOL

Figure 1 depicts the overall architecture of the tool. The tool is called C2Sim. This tool has mainly two components: a C to Simulink translator and a test suite generator. The translator takes a C function and generates a Matlab script file. This script file contains appropriately generated model construction commands which generate appropriate components in the target Simulink model. The target Simulink

[*]Honeywell Technology Solutions
[†]Software Engineering and Technology Labs

*ISEC'09,* February 23–26, 2009, Pune, India.
ACM 978-1-60558-426-3/09/02.

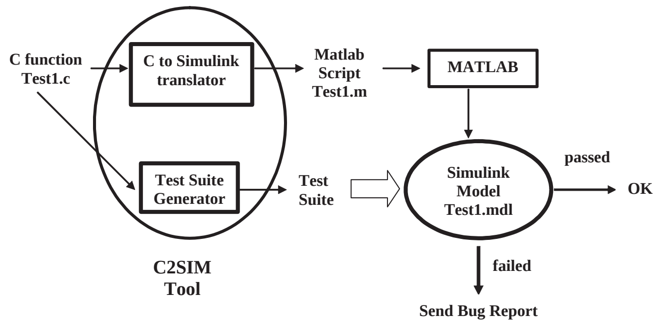

**Figure 1: Architecture of C2Sim Tool**

model can be generated by running the script in Matlab. We use a methodology called Conformance testing to validate a translated model with respect to the C fucntion from which it has been generated. In figure 1, the test suite generator generates test cases from the C program with proper coverage criteria. These test cases are applied to the generated Simulink model to ensure its correctness. We use *Blast*, a software model checker [4], for generating test cases from a C function.

The input of the translator is an ANSI C file containing one or more interdependent functions. The translator generates separate models for these C functions. We consider the C grammar specification available at [1, 2]. The translator successfully translates a C function with the following components: input parameters, return statement, assignment statements with arithmetic and logical operations, function calls, If and Switch selection statements, For, While, and Do-While iteration statements, Arrays and Structures. At present the translator cannot translate a C function containing pointer or recursive function. Simulink has a rich set of model elements which cover various areas of signal processing. We have closely examined the C grammar, and selected the following Simulink blocks which are mainly used for translation: (i) **Sources:** In1, Constant, (ii) **Sinks:** Out1, (iii) **Signal Routing:** Mux, Demux, Bus Creator, Bus Selector, Switch, Index vector, From, Goto, (iv) **Math Operations:** Add, Subtract, Gain, Product, Divide, Abs, Unary Minus, Math Function, Trigonometric Function, (v) **Logic and Bit Operations:** Logical Operator, Relational Operator, Compare to Constant (vi) **Ports and Subsystems:** Model, If - If Action Sub-system, Switch Case - Switch Case Action Subsystem, For Iterator Subsystem, While Iterator Subsystem, (vii) **Discrete:** Unit Delay.

To generate Simulink model for a C function we use the model construction commands that can be used for wide range of operations for working on Simulink models. For example, the command *add_block* adds a block in a Simulink model, the command *add_line* connects two blocks by a line. Our approach to create a Simulink model is to generate a Matlab script containing necessary model construction commands with correct parameters and in proper order. In this translation process our aim is to generate a Matlab script file for a C function using ANSI-C grammar through syntax-directed translation. We capture C grammar in a Yaccfile. For the production rules of the C grammar we identify the appropriate commands with appropriate parameters to generate desired components in the Simulink model and capture them as action items to write the corresponding information for the desired blocks in the targeted file with ".m" extension during the parsing process. The translation process depends on the tokens that the Yacc parser collects from the Lexical Analyzer.

A C program is a collection of interdependent functions. The translator takes each of these functions one by one and converts them to an equivalent Simulink model. So we have separate models for each function. If a function *func1* calls another function *func2*, then the model of *func2* is incorporated in the model of *func1* appropriately with Model block in Port and Subsystems library in Simulink. If a function calls another function, the model (with .mdl extension) of the callee function can be generated successfully from the .m file generated by the translator, if the model of the called function is already available. However, a legacy code may have several functions and it is difficult to know which function is calling what other functions during translation. To deal with this situation, we generate a function call graph for the entire legacy code by some function call generator (for example, CodeViz [3]). Then breath-first search is carried out on the generated function call graph, and the functions are stored in a list. While generating simulink models from the ".m" files, the list is traversed in the reverse direction. So, the functions corresponding to the leaf nodes in the function call graph are considered first. As these functions do not call any other function, models for them are successfully generated. Then the next upper level is considered. In this way, models for all the functions are successfully generated.

## 3. CONCLUSION

Generating Simulink model from legacy C code is of prime importance to avionics and automotive industries. This importance arises from lots of advantages that can be gained using MBD process in the design, development and maintenance of embedded software. The avionics and automotive industries are willing to convert all their legacy C codes to executable models, and then leverage MBD process for further work. The approach we discussed in this paper will be of great help towards this goal. We have implemented the translator covering a large subset of C programming language. We have also completely automated the test suite generation process. The translator and test suite generator tools have been tested on a number of C programs with encouraging results. In near future, we plan to carry out a few large scale industrial case studies on legacy C code coming from automotive domain.

## 4. REFERENCES

[1] J. Degener. ANSI C Grammar, Lex Specification. 1995. Available at http://www.lysator.liu.se/c/ANSI-C-grammar-l.html.

[2] J. Degener. ANSI C Yacc Grammar. 1995. Available at http://www.lysator.liu.se/c/ANSI-C-grammar-y.html.

[3] M. Gorman. CodeViz: A CallGraph Visualiser. http://http://www.csn.ul.ie/ mel/projects/codeviz/.

[4] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software Verification with Blast. In *10th International Workshop on Model Checking of Software (SPIN)*, volume 2648 (LNCS), pages 235–239. Springer-Verlag, 2003.

[5] S. Lehman. Controller Style Guidelines for Production Intent Development Using MATLAB, Simulink, and Stateflow. 2001.

[6] V. Zsombori. Transformation of C Code to Matlab/Simulink Models, 2005. http://www.doc.gold.ac.uk/m̃as01vz/dc_shanghai/project_transformation/summary.pdf.