Industrial Evaluation of an Efficient Equation Model Debugger in OpenModelica

Åke Kinnander
Siemens Turbo Machinery AB
Finspång, Sweden
ake.kinnander@outlook.com

Martin Sjölund
Dept. Computer and Information Science
Linköping University,
Linköping, Sweden
martin.sjolund@liu.se

Adrian Pop
Dept. Computer and Information Science
Linköping University,
Linköping, Sweden
adrian.pop@liu.se

***Abstract*— The ease of use and the high abstraction level of equation-based object-oriented languages (EOO) such as Modelica has the drawback that programming and modeling errors are often hard to find. To address this problem, we have earlier developed an advanced equation model debugger in the OpenModelica tool. The aim of this paper is to perform an independent investigation and evaluation of the OpenModelica equation model debugger on industrial models. The results turned out to be mainly positive, the debugger located several kinds of errors such as division by zero, chattering, etc. It remains to further evaluate the debugger on larger industrial models.**

Keywords—debugger, equation, Modelica, industrial, fault, model

I. INTRODUCTION

The development of today's complex products requires integrated environments and equation-based object-oriented declarative (EOO) languages such as Modelica [4] [5] for modeling and simulation.

The increased ease of use, the high abstraction, and the expressivity of such languages are very attractive properties. However, the drawback of this high-level approach is that understanding the causes of unexpected behavior and numerical errors of simulation model is very difficult, in particular for users who are not experts in simulation methods.

Therefore we have recently developed an advanced integrated equation model debugger [2], for the Modelica language, as part of the OpenModelica tool. This is quite different from debuggers of conventional algorithmic programming language debuggers [7] [6] [8]. We have earlier developed a debugger [1] for the algorithmic subset of Modelica and a debugger [9] that analyzes the causes of over-constrained or under-constrained systems of equations. The new debugger is also based on the recent development of the advanced bootstrapped OpenModelica compiler [3].

The integrated equation model debugger has been evaluated by the designers and performs well on both small and big models. However, an independent evaluation of the debugger by industrial users on industrial problems was still missing. Such an evaluation is the main topic of this paper.

The rest of the paper is structured as follows: first the errors to be investigated and models to be evaluated are briefly presented. Section II introduces the debugger tests in more detail. Section III presents debugging of errors in the logarithmic temperature calculation whereas Section IV presents debugging of errors due to bad initial values. Finally, Section V presents conclusions.

A. *Errors to be Investigated*

In order to investigate different types of errors that could be expected to occur, a small and simple evaporator model is used. This has been fetched from a larger model used for transient analysis of combined power plants by Siemens Industrial Turbomachinery AB in Finspång, Sweden. The following errors are to be investigated:

1. Division by zero
2. Errors in the average logarithmic temperature difference used for heat transfer calculation:
    a. Inlet temperature difference =0
    b. Inlet temperature difference=outlet temperature difference.
3. Boiling in the evaporator that causes halt of simulation progress by much too small time steps (stiffness)
4. Various test of bad initial values, with variation of pressure, temperatures, flows and masses in the different parts of the process.

The model selected is a simplified model of an error free model, hence the above test will be deliberately inserted and the debugging tool will only be examined by its outputs, while a sharper application for a real model development where errors are unknown and the debugger support for identifying them will be more apparent, will be carried out later. The reason for this is the limited time for testing available, and that a sharp application will only provide stochastic errors and could thereby not be planned in time.

B. *Models for the Evaluation*

The following model shown in Figure 1 will be used for the investigation.

It consists of an evaporator model that has flue gases as heating source and water as coolant, producing dry steam

to the steam sink. The steam production is decided by the heat from the flue gases, the enthalpy (temperature and pressure) of the water source (`FWpump`), and the steam extraction to the steam sink (`SteamSink`) that in turn is tracking the evaporators drum pressure with a negative bias of 0.1 to 1 bar. The model has 1110 equations.

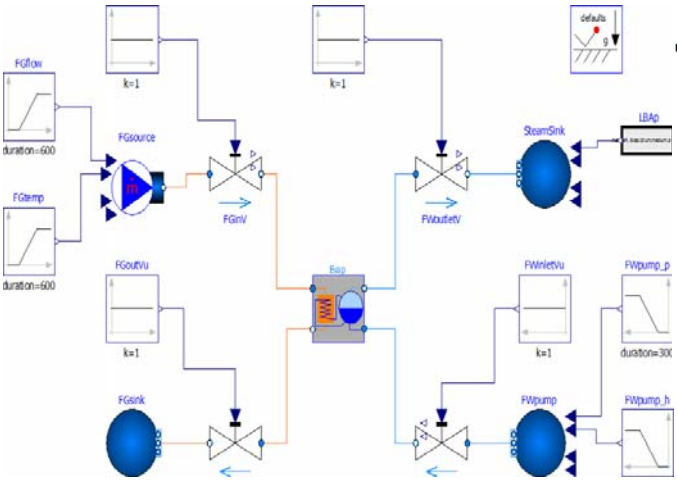

Figure 1. Evaporator test model

The evaporator model is designed according to Figure 2.

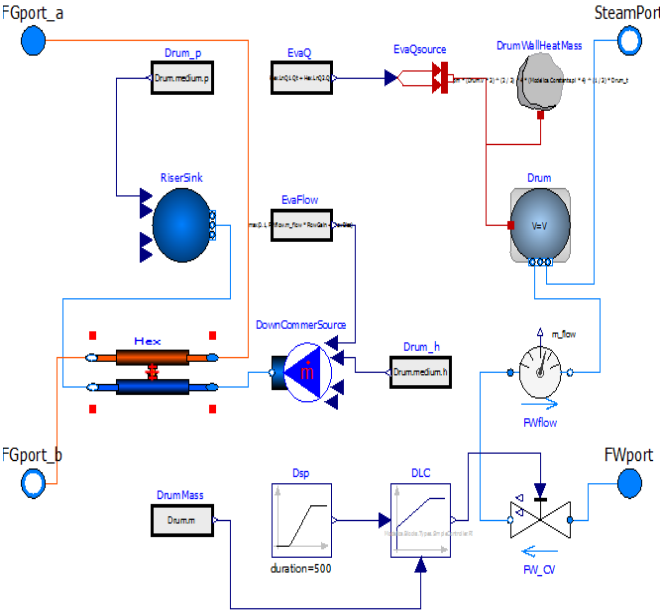

Figure 2. Evaporator model

The evaporator model has a drum model (`Drum`), a heat exchanger (`Hex`) and a level controller (`DLC`) that controls level by a control valve (`FW_CV`) . The level controller is no actual water level controller in length units (m), instead it controls the amount of water (kg) in the drum. The thermal capacity of the drum metals is represented of a heat capacity model (`DrumWallHeatMass`) but the insulation towards surroundings are assumed to be perfect, i.e. no heat losses to the outside of the drum. This model has 809 equations.

The drum model is the volume model from the Fluid library manipulated to only let steam exit, i.e., always perfect separation. The heat exchanger is according to Figure 3.

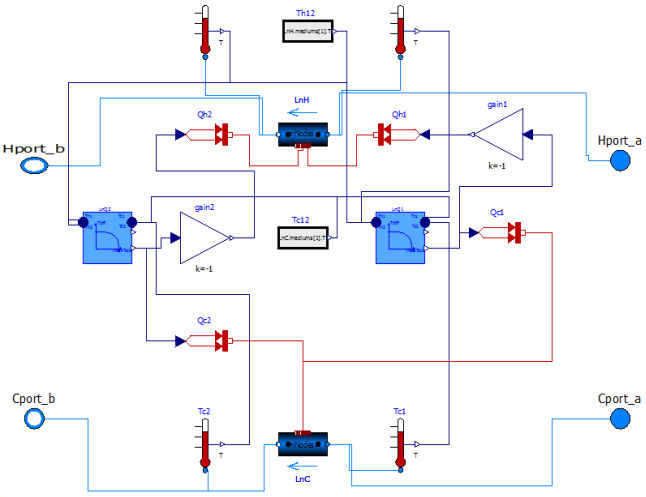

Figure 3. Heat exchanger

The heat exchanger has a pipe model (`LnH`) that corresponds to the flue gas channel and a pipe model (`LnC`) that corresponds to the pipe bundle carrying the water to be heated. The heat exchanger has parallel flows. In this simplified model the overall heat transfer coefficient (J/m^2/K) is a constant, i.e. it takes not configuration or medium properties into account. The driving temperature difference is calculated for respectively inlet and outlet sections of the pipe bundles (i.e. they configured with 2 nodes each) by the models `LnQ1` and `LnQ2`. The temperatures are measured besides inlets and outlets for both pipe models also in the middle (`Tc12` and `Th12`). This model has 580 equations.

Finally, the heat calculation models `LnQ1` and `LnQ2` are according to Figure 4. All other models are from the standard Modelica library.

The model has a low pass filter with an input connected in the text layer where the logarithmic temperature difference from the connectors `Th` and `Tc` which both are connecting both inlet and outlet temperatures respectively medium side, is calculated. The output of the model is the calculated heat transfer (W). The overall heat transfer coefficient `ktot` is calculated from parameters representing the heat transfer coefficient from flue gas to metal, `k_outer`, and from metal to water, `k_inner`. In the present full size boiler model those parameters are replaced with connectors that provides more
accurate values, based on medium and flow properties calculated in separate models. The heat transfer area is a ramp with selectable duration and height values, to be adopted to what's needed from initializing aspect (duration of suitable size respectively to the real heat transfer area).

This model contributes with 39 equations of the total of 1110.

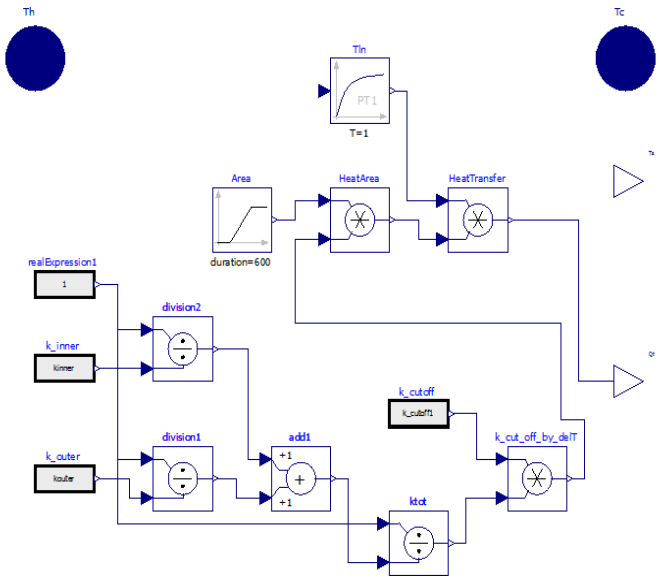

Figure 4. Heat transfer calculation model

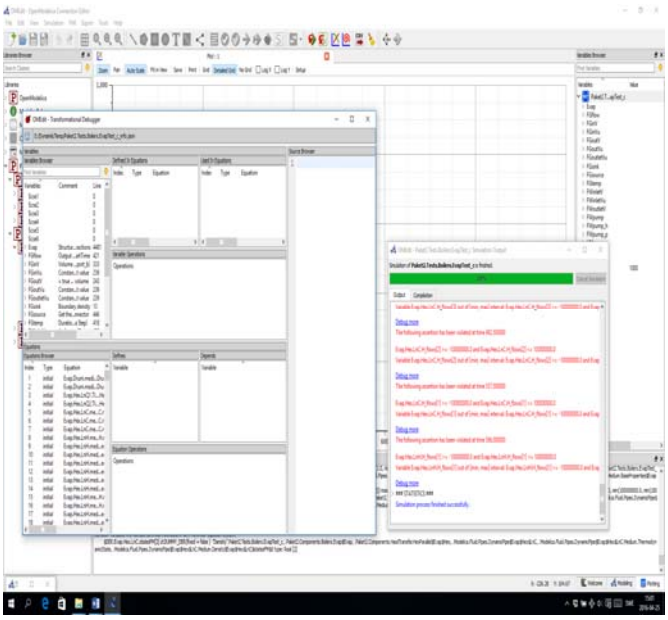

Figure 5. Information from OpenModelica with debugger activated at successful simulation

II. Debugger Tests

A. *Activation of debugger*

The debugger is activated by setting the flag « Launch transformational debugger ». After a successful simulation the output windows are containing following information (Figure 5).

The simulation output window contains assertion violation messages that are false, because the enthalpy flow H (W) has too narrow range in the Standard Modelica library. It ought to be at least 10 times as big.

This violation has no influence on the simulation result (might there be an unnecessary delay?). The window shows with a green bar that 100 % of simulation is done and the blue text that it has been successful. The debugger transformational debugger window shows all variables in the variable browsers window and all equations in the equation browser window, as found in the simulation code. All other frames in the debugger are empty.

B. *Division by zero by parameter setting*

The test is done by setting parameter k_inner to zero. The simulation output window displays following messages (Figure 6)

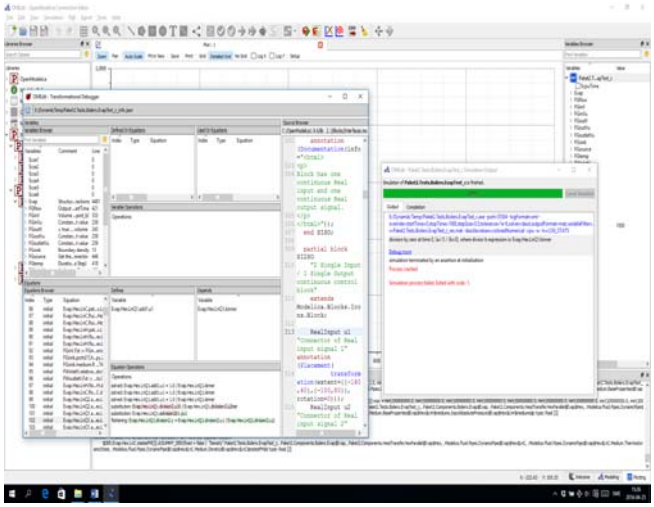

Figure 6. Division by zero by parameter setting

The simulation output window gives the required information that simulation crashed at initialization due to an assertion that avoids division by zero and this is caused by k_inner=0.The debugger window looks as before but after clicking debug more in the simulation output window it looks as in (Figure 6).

The equation browser marks initial equation with index `102:Evap.Hex.LnQ1.add1.u1:=DIVISION(1.0,Evap.Hex.LnQ1.kinner)`, which is the same information as in simulation output window. The frame denoted Defines give the variable that becomes undefined by the zero division. The frame denoted `Depends` give the variable name `Evap.Hex.LnQ1.k_inner`.
For this error the debugger gives all information necessary.

C. *Division by zero by time function*

The `k_inner` variable is replaced by a time function that ramps it down to zero in 100 seconds. This results in a never ending simulation .

The solver manages to pass the 100 s time point where `k_inner` is zero and a division by zero occurs. No plots are available but the ramp proceeds to negative values for `k_inner`. The solver has skipped the exact 100 s time point, but then continued into other problems, due to the negative `k_inner` value. On the passage it has however produced two messages about zero division at time 100 when they occurred. In the case of the user being unaware of the division problem, the large amount of output in the simulation output window hides those messages.

For a ramped denominator passing zero, the debugger is

not optimal in case the solver manages to pass the critical point and that consecutive errors then hides the information from the user. A solution would be the option to let the user decide if division by zero should be accepted or not, i.e. the solver should then interrupt and save when any denominator having a passage of zero.

D. *Division by zero due to mismatch in parameter settings*

By deselection of the heat transfer use in the `LnC` pipe in the Hex model (Figure 3) the test model still checks OK, but now with only 1106 equations instead of 1110. Simulation gives the following simulation output window and transformational debugger window (Figure 7).

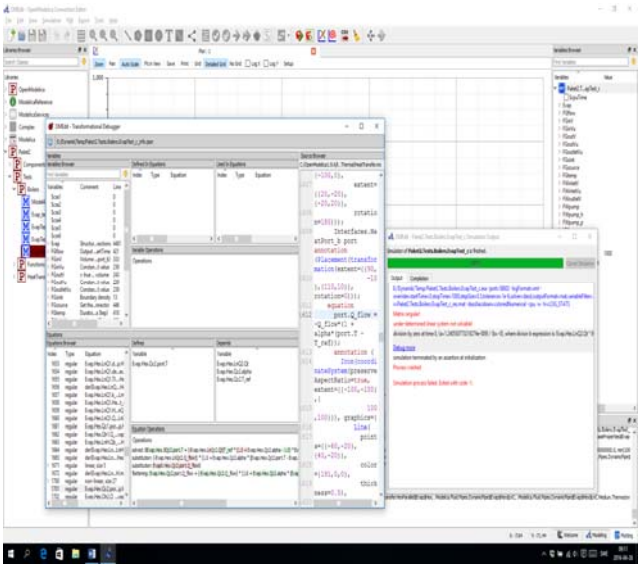

Figure 7. Outputs after no use of heat transfer specified but corresponding heat transfer connectors anyway connected

The simulation crashes at initialization due to division by zero where the denominator is involving the variables `Qt` and `alpha`. A check of the model reveals that `alpha` is the heat transfer coefficient to surroundings and is deliberately zero. That is, this model is impossible to run although it checks OK. The debugger points to the equation numbered 1258. Marking that equation points to a source code line 1612 where the Modelica standard library (MSL) component `PrescribedHeatFlow` (PHF) equals the heat ports heat flow (`Q_Flow`) to the connector input `Q_Flow` (the prescribed flow). This equation contains also a dependence on alpha, but with alpha=0 the equation should be OK.

`port.Q_Flow=-Q_Flow`. The `PHF` model calculates its internal temperature in order to be able to have the prescribed heat flow by this equation (index 1258). This temperature should be the same as the connected pipes temperature as there is no thermal resistance in the connection itself so why is it calculated in this way? One could not be sure that any user would understand that this equation is unexpected and caused by a wrong parameter setting in the pipe model.

It would have been better if there were a consistency check between connectors and their use in the program. One way would be not to show the connector unless it is activated, or give an error message if it is anyway connected. Could the variables browser shed some light over the problem? The variables browser is presented in (Figure 8) after the port temperature has been clicked.

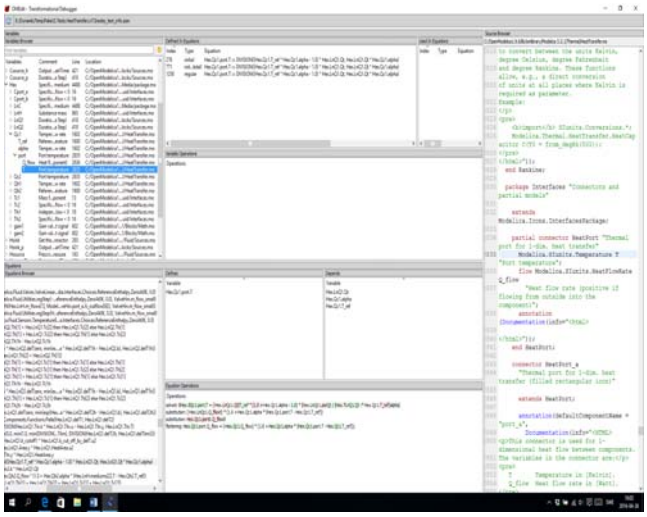

Figure 8. Information from variables browser

It gives the same information added with the initial equations. A question is why is not the debugger pointing to the initial equations indexes 276 and 771, as the simulation output window tells that the error appeared at initialization.

III. Errors in the Logarithmic Temperature Difference Calculation.

Errors in the logarithmic temperature difference calculation should be treated the same way as the division by zero. Interesting is however, if the solver also for this types of errors manage to pass the critical point as their time duration could be expected to be very short. Basically this investigation is more an investigation of the solver and not the debugger but the debugger will be activated and therefore also this investigation is a part of the paper.

A. *Temperature differences passing zero*

This test is achieved by removing the numerical fences that prevents zero crossing. Unfortunately, it turns out that there are no crossings that passes `delT=0` for the case simulated, and the test needs further work to be carried out, and therefore postponed and not published in this paper. This is unexpected and errors might have occurred when moving the model from another tool to OpenModelica.

B. *Temperature differences at outlet and inlet passing equal values*

This is happening without any numerical problems, i.e., the solver skips the critical time where they are equal or

happens to avoid it without any actions.

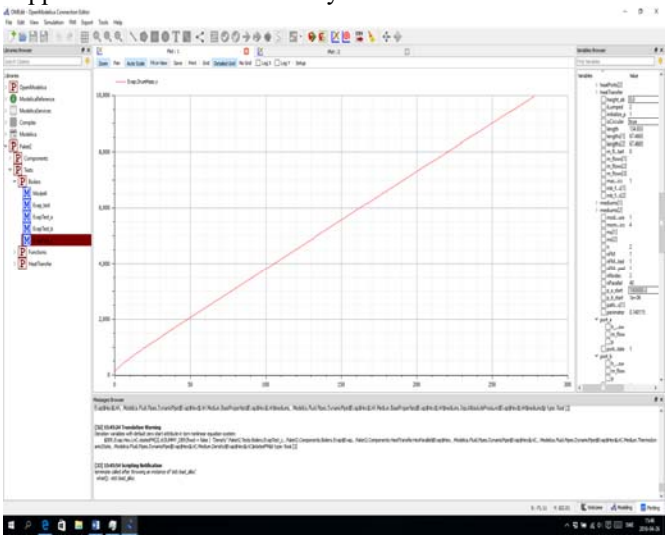

Figure 9. Plot of drum mass at blocked drum outlet

IV. BAD INITIAL VALUES OR BAD SIMULATION BOUNDARIES

The debugger support, if any, is to be investigated for this type of problems where simulation runs into numerical problems.

A. *Too high backpressure*

By increasing the back pressure from the steam pipes to exceed drum pressure, and thus preventing steam flow out of the drum. The simulation terminates at 277.7 s. The result file is written, i.e. it is possible to plot. The plotting reveals that the simulation crash is probably due to that the drum gets filled with water. The transformational debugger window points at the drum. The simulation output window recommends to log non-linear systems (NLS). Doing this gives a not responding OpenModelica. Restart gives a runtime error. A restart and simulation again without LOG_NLS activated gives the same result. The plotting of the Drum parameter mass shows that drum gets filled as it has no outlet (Figure 9).

The debugger test failed here on an OpenModelica problem with handling LOG_NLS. However, at this error the result file was generated and provides a tool for debugging. On the other hand, LOG_NLS is not a part of the debugger. The debugger information for this type of failure is not sufficient to remedy the problem directly, although it points at the drum as a probable cause. Eventually the recommended logging of NLS could have given the direct cause of crash. From the OpenModelica user point of view, the plotting after crash is very valuable, and it reveals that the drum gets filled from the LBA system (no check valves in this test model), which calls for corrective actions regardless of the what caused the actual solver crash.

V. CONCLUSION

The basic property of the debugger is at numerical problems and violation of assertions like numerical ranges that a certain variable is expected never to exceed. Then the debugger points out the equation causing the problem. In this paper only small (1000-equation size) models have been tested, that shows that the debugger works. The debugger has also been evaluated [2] on 11116-equation size models such as V6Engine. To really evaluate the benefits of the debugger, but also its functionality, it should be applied to larger industrial models.

The following conclusions were made from the test with the transformational debugger for equation models:

- The debugger works well to find zero denominators that are parameters.
- The debugger does not come into play automatically if the zero denominator is only a momentarily value, as the solver managed work around such time points in so far tested simulations. However, it catches the problem in the simulation output window, and gives a message that by clicking opens the transformational debugger window which displays the concerned equation. However, there is a risk that this is unnoticed as the solver continues and could generate a lot of consequential or other messages that could hide the zero denominator messages. It would be preferable if the simulation output window could aggregate messages of the same type into one, expandable, line, thereby giving a better overview of the all the types of messages the simulation has generated.
- A zero denominator caused by structural model errors, like connection to not used connectors (this should not pass the model checking) the debugger points to the causing equation. One could not ask more of the transformational debugger, but the OpenModelica model check or model building could be made to prevent such mistakes.
- At numerical problems causing long execution times the debugger points to the equations that has problems, but to understand the exact problem, plots of variables could be necessary – hence the result file should always be generated, regardless if the simulation is interrupted by solver or manually. This is not the case in the tested version for all the tests.

VI. ACKNOWLEDGMENTS

This work is partially supported by the ITEA 2 MODRIO project via the Swedish Government (Vinnova) and by Siemens Turbo Machinery AB.

REFERENCES

[1] Adrian Pop and Peter Fritzson (2005). A Portable Debugger for Algorithmic Modelica Code. In *Proceedings of the 4th International Modelica Conference*, Hamburg, Germany.

[2] Adrian Pop, Martin Sjölund, Adeel Asghar, Peter Fritzson, Francesco Casella. Integrated Debugging of Modelica Models. *Modeling, Identication, and Control*, Vol 35, No 2, pp. 93-107, DOI: http://dx.doi.org/10.4173/mic.2014.2.3, ISSN 1890-1328, Aug 2014.

[3] Martin Sjölund, Peter Fritzson and Adrian Pop. Bootstrapping a Compiler for an Equation-Based Object-Oriented Language. DOI: 10.4173/mic.2014.1.1. *Modeling, Identification and Control*, Vol 35, No 1, pp 1-19, 2014.

[4] Modelica Association. *Modelica Language Specification version 3.3 revision 1*. 2014. URL www.modelica.org.

[5] Peter Fritzson. Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach. 1250 pages. ISBN 9781-118-859124, Wiley IEEE Press, 2014.

[6] Nicholas Nethercote and Julian Seward. Valgrind: a Framework for Heavyweight Dynamic Binary Instrumentation. In Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation. PLDI '07. San Diego, California, USA, 2007, pp. 89-100. doi: 10.1145/1250734.1250746

[7] Richard Stallman, Roland Pesch, Stan Shebs, et al. (2011). *Debugging with GD*B. Free Software Foundation. [online] Available at: < http://unix.lsa.umich.edu/HPC201/refs/gdb.pdf> [Accessed 30 October 2011].

[8] Andreas Zeller. Why Programs Fail, Second Edition: A Guide to Systematic Debugging. ISBN: 978-0123745156, 2009

[9] Peter Bunus. Debugging Techniques for Equation-Based Languages. PhD Thesis. Department of Computer and Information Science, Linköping University, 2004.