Model verification and debugging of EOO models aided by model reduction techniques
* Work in Progress Paper **

Anton Sodja Borut Zupančič

Fakulteta za elektrotehniko, Univerza v Ljubljani, Slovenia,
{anton.sodja,borut.zupancic}@fe.uni-lj.si

Abstract

Equation-based object-oriented modeling approach significantly reduced effort needed for model implementation by releasing modeler of performing many error-prone tasks. An increasingly more complex models can be built, preferably from components of different model libraries. However, complexity of the models complicate the process of verification – assuring that the model was implemented correctly and behaves as expected – and possible subsequent debugging. A cause of error in a model with over 1000 different equations can be often hard to find by the desk-checking method. This requires the development of new modeling environment tools for model understanding and automated discovery of the fault causes.

The difficulty of designing such tools in EOO modeling environments is linked to the difficulty of mapping simulation form to the model sources. Furthermore, debugging of complex models consisting of over thousand equations by traversing each equation may be very ineffective, especially when the fault has multiple and not very evident causes.

A model reduction methods is proposed and discussed as a method of verification. With model reduction methods it is possible to identify the most important parts of the model which have contributed to the specific model behavior. Because model reduction can be performed on original model representation, the difficulty of mapping simulation form back to model source is avoided.

Keywords verification, debugging of EOO models, verification by model reduction

1. Introduction

Important step in process of modeling and simulation is verification of the model. With verification we assure matching of our conceptual model with the implementation of the model.

Traditionally models were implemented in imperative programming languages and verification of the models could rely heavily to the established software debugging practices and techniques . However, declarative equation-based object-oriented modeling languages, like Modelica, introduced a new abstraction layer in implementation of the model to improve modeling process. Differential algebraic equations by which model is described can be entered directly and further such models can be combined into more complex models according to the rules defined by their interfaces. Such implementations of the models preserve topology of the modeled system and models are thus more evident and clear, but for simulation purposes such model must be preprocessed – translated into the simulation form. Translation is performed automatically.

Since the modeler is no longer acquainted with the simulation form of the model, the paradigm of the model verification and debugging has changed and we can no longer effectively use software debugging tools developed for imperative programming languages. However, verification and debugging of declarative modeling languages is still not solved adequately and it remains a challenging research topic.

2. Overview of verification techniques

A number of verification techniques have been developed, spanning from very informal approaches to formal mathematical proofs of correctness. Whitner and Balci [13] categorized verification methods into six distinct perspectives based on increasing level of mathematical formality: informal, static, dynamic, symbolic, constraint and formal analysis. Effectiveness of verification usually increases as method becomes more formal, but on behalf of increased complexity.

2.1 Informal analysis

Informal analysis techniques, such as desk checking, are most commonly used verification strategies where model is evaluated using the human mind. The evaluations can be made by mentally exercising the model, reviewing the logic behind the equations, algorithms and decisions, and

3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. October, 2010, Oslo, Norway.

examining the effects that the various implementations will have on the overall outcome of the model.

Declarative modeling languages specifically improve efficiency of the informal analysis of the model, since implementation of the model is close to the form of conceptual model with preserved topology of the modeled system.

However, informal analysis is very time consuming and its success depends on the level of knowledge and expertise of the individual, which comes into concern when very complex models are built with an aid of pre-prepared model libraries with implementation of the components that may not be completely known to the user.

2.2 Static analysis

Static analysis is a verification method where only the source code of the model is analyzed without actually performing simulation.

It is the best supported verification method by EOO modeling tools. The most important aspect of it is structural analysis – checking that number of variables match the number of equations in the model (resulting system of equations derived from model is well-constrained) and in case if over- or under-constrained system of equations was detected, a tool (debugger) must be able to report the location of the error consistently with the user's perception of the simulation model and possibly suggest the right error-fixing solution [1]. In Modelica language standard 3.0, restrictions have been introduced into the language in order to force matching of a number of unknowns and equations on every hierarchical level (in each submodel) [8]. This has improved efficiency of the debugging over- and under-constrained models since it is possible to easily determine a submodel with too many or too few equations.

Object-oriented modeling where variables are also objects with many additional properties besides value enables also other kind of static analysis, for example, unit checking [6].

The advantage of static analysis in EOO modeling is that costly translation and simulation of the model is avoided, but it can not fully verify that intentions of the modeler are being met or determine the model's behavior.

2.3 Dynamic analysis

Verification by dynamic analysis is accomplished by evaluating model's simulation results. In general this a complex task since it is not easy to determine what to test and how to test it. Furthermore, it can be complicated to interpret the analysis' results.

The EOO modeling introduce additional difficulties into verification by dynamic analysis due to optimization performed during translation of the model when a lot of information about the original model's structure is lost. Mapping the simulation form back to original model is thus a very challenging task.

Currently, no modeling tool supporting Modelica provides even the simplest run-time debugging capabilities that could be used for inspecting of failed tests. However, there were some prototypes of automated debuggers for EOO languages developed [1, 9]. They are based on the principles of debugging of the programming languages. The debugging strategy is interactive and based on the user's choice of the variables which trajectories are wrong and according to these variables the simulation form of the model is sliced so that only parts (equations and algorithms) having influence on selected-variables' calculation are shown to the user. A dependency graph is built where nodes represent equations that contributed to the result connected by directed edges labeled by variables or parameters which are inputs or outputs from or to the equation represented by the node. Then the user is able to classify a variable as variable with wrong value or classify an equation as correct (which results in rebuilding the dependency graph) and modify some of equations and variables' values interactively. Each equation in dependency graph is also mapped to model source.

Although debugging methods [1, 9] provide significant advantage over no debugging tools at all, they have some serious deficiencies. The user is basically still operating on the level of model's simulation form, even if mapping to model code positions is provided. Also scalability, when dealing with large models with many equations, is problematic while considering each equation one by one is very impractical.

A special topic of dynamic analysis is resolving numerical problems that might arise during simulation. However, this is very advanced topic, while user must posses a good knowledge of the simulation form to which model was translated and properties of the numerical solver.

2.4 Symbolic analysis

Verification by symbolic analysis addresses some drawbacks of the dynamic analysis, namely the inability to verify all possible test cases. Verification by symbolic analysis seeks to determine the behavior of the model during simulation by providing symbolic inputs to the model.

2.5 Constraint analysis

Constraint analysis verifies on the basis of comparison between model assumptions and actual conditions arising during simulation of the model. The model assumptions are made already during the development of the conceptual model. In Modelica they can be checked by means of assert statement provided by language standard [7] and are extensively used in Modelica Standard Library, for example, in *Modelica.Media* it is in that way assured that the medium equations are never used outside the valid temperature range.

The disadvantage of constraint analysis is reliance on formal model specifications which is needed to effectively state and place the assertions. Creating a formal specification is a difficult task.

2.6 Formal analysis

Formal analysis is based on formal mathematical proof of correctness. It usually can not be applied even on the most simple models and basically it has no practical value so far.

3. Problematics of verifying EOO models

Modeling is a process of extraction, organization and representation of the knowledge about physical system [2].

The contribution that equation-based object-oriented modeling approach brought to the process of modeling is implementation of the model close to the conceptual model, i.e., model is stated in acausal form and topology of the system is preserved. In contrast to traditional procedures where model was implemented in imperative languages, automatic translation of the model relives the user of tedious and error-prone task of manual manipulation of the model's equations.

EOO modeling approach also enabled truly reusable models and consequently many model libraries have been developed. Rich selection of already prepared components allows building relatively complicated and "error-free" models with a little effort. However, although once common errors due to implementation in imperative languages are no longer an issue, there can be identified three types of faults emerging in EOO models which can be exposed in simulation results [1]:

- when parameter values for the model simulation are incorrect
- when the equations that specify the model behavior are incorrect (violate physical laws)
- when submodels are used inappropriately (assumptions about the system are violated)

These faults can be found by either failed assert statement or by inspection of the simulation results (failed test).

When a fault is found in a model, the cause or several of them (if the fault have multiple causes) have to be found and corrected in the model implementation. For example, if certain quantity in the model increased unbounded, although stable system with bounded input signal have been simulated, it is a plausible assumption that some equation is wrong or some parameter has been assigned an unphysical value. A strategy of traversing the equations related to this quantity will certainly lead to the solution of the problem, although in a large-size model this procedure is somehow laborious. A debugger based on interactive dependency graph proposed by [9] may be very useful in such case.

However, a case when, for example, model's response exposes unexpected initial undershoot is much more complicated to resolve. Initial undershoot may be a property of the system (a nonminimal phase), only property of the model (introduced by some modeling assumptions and simplifications) or an error in the implementation. A simple traversal of the equations related to the trajectory exposing initial undershoot may not provide a proper insight, especially if a large-size model built from complicated components is under consideration (e.g., component *DynamicPipe* from the library *Modelica.Fluid* can consist of over 100 equations distributed to 10 models from which original component is extended).

Verification of complex EOO models should be supported by a tool that directs the modeler towards the important parts of the model regarding a certain behavior of the model (exposed by trajectories obtained by simulation).

3.1 Model reduction techniques

For some modeling purposes, most notably structural and control design, large size models that include detailed dynamics are undesired, since determining the major design parameters and their relationship to the system performance is difficult [5].

Model reduction techniques represent also an important aspect of the systems analysis, when a low-dimensional model can provide qualitative understanding of the phenomena under consideration [4]. An important property of the model reduction methods when used in system analysis or for structural design is that it generates a *proper model*, i.e. reduced model with the minimum complexity required to meet the performance specifications and possessing physically meaningful parameters and states [5].

In attempt to reduce the complexity of the model (and number of its parameters), a number of mixed numerical-symbolic model reduction techniques have been developed and successfully applied [11, 12, 4, 5].

Model order reduction techniques consists of running a series of simulations, ranking the individual coordinates or elements by the appropriate metric and removing those that fall below a certain threshold [3].

The most straightforward metrics for reduction of the proper models is related to energy or power. Method of Rosenberg and Zhou [10] removes bonds with low power from a bond graph model, Luca [5] introduces activity – the time integral of the absolute value – and Ye and Youcef-Youmi [14] reduces bond graph models by eliminating bonds with smallest associated energy in comparison to its neighbors. Chang *et al.* [3] eliminate system's states with low associated energy in the model comprised of Lagrangian subsystems with force interconnections based on Lyapunov stability. Metrics related to energy or power requires modeling formalism with clearly defined components' energy and power respectively. Method of Sommer *at al.* [12] consists of term substitution and deletion (as well as on some other simplification) in the equations of the DAE system derived from the model. The term are ranked according to their influence on the output error which is defined as a difference of original and reduced model's output. The resulting simplified system of equations can be interpreted again in the form of component equations and can be mapped to a reduced model scheme.

For the simulations that are performed to obtain the error estimates, excitation of the model must be selected in such way that a valid model is obtained in a desired frequency range.

3.2 Verification aided by model reduction techniques

Verification by dynamic analysis as well as subsequent debugging in equation-based object-oriented approach could be improved substantially by using model reduction techniques.

When a behavior of the model obtained by simulation is not as expected or even erroneous, an explanation is sought. It is sensible to first look at that components or (terms of) equations and parameters of the model that have the most influence on the dominant system dynamics and trajectory of the model's variable of interest respectively.

In a large-size models made up of components from various model libraries which implementation is not precisely known to the user, it is not apparent which components or equations of the model have the greatest impact on the response of the model, i.e. on selected variable's trajectory (or part of it). Determination of the most influential components can be automatized by using ranking algorithm known from the model-reduction methods. The user could focus only on the few components contributing most significantly to the simulation results and potentially extend his/her search to lower ranked components. Because the ranking is affected by selected time-window, components can be separated also according to the impact they have during different time of simulation. For example, in the steady-state, dynamic terms (those that include time derivatives) are totally irrelevant, while at the beginning of the transient, terms of equations describing the fast dynamics of the system are those which are worth of the most attention.

Furthermore, on behalf of the user, proper reduced model could be generated and the user could further experiment on the reduced model which can be much more easily understand. Because all the parameters and states of the reduced model are physically meaningful, the changes of the reduced model could be easily merged with original model.

An important advantage of the model reduction of EOO models is also that user never needs to consider the translated form of the model, while model reduction is performed on the original representation of the model.

However, in general modeling language such as Modelica, models can be implemented in various ways, for example, entirely in textual form as a set of equations or by connecting basic components from the Standard Library in the graphical interface. That implies using different model reduction strategies and possibly also different metrics.

4. Conclusions

Use of model reduction techniques for verification and debugging have been proposed. Methods originated from model reduction techniques can be used to rank the components (equation terms) of the model with greatest impact on the model behavior (selected trajectory) and proper reduced models can be generated on behalf of the user. Reduced models are easily to comprehend by the user and while proper reduced models have physical meaningful parameters, changes done to the reduced model can be easily merged to the original model.

The advantage of debugging aided by model reduction methods over traditional software debugging methods is that since the user does not need to consider the simulation form of the model anymore, the difficult mapping of translated equations to model source is not needed.

However, most model reduction methods requires specific modeling formalism (e.g., bond graphs) and can be restricted to specific physical domains. The usefulness of verification and debugging tools based on model reduction techniques is therefore limited in a general EOO modeling languages such as Modelica.

References

[1] Peter Bunus. *Debugging Techniques for Equation-Based Languages*. PhD thesis, Linköping University, 2004.

[2] F. E. Cellier and E. Kofman. *Continous System Simulation*. Springer Science+Business Media, New York, 2006.

[3] Samuel Y. Chang, Christopher R. Carlson Carlson, and J. Christian Gerdes. A lyapunov function approach to energy based model reduction. In *Proceedings of the ASME Dynamic Systems and Control Division – 2001 IMECE*, pages 363–370, New York, USA, 2001.

[4] Sanjay Lall, Petr Krysl, et al. Structure-preserving model reduction for mechanical systems. *Physica D*, 284:304–318, 2003.

[5] Loucas Sotiri Louca. *An Energy-based Model Reduction Methodology for Automated Modeling*. PhD thesis, University of Michigan, 1998.

[6] S. E. Mattsson and H. Elmqvist. Unit checking and quantity conservation. In *Proceedings of the 6th Modelica Conference*, pages 13–20, Bielefeld, Germany, 2008.

[7] Modelica Association. *Modelica Specification, version 3.1*, 2009. `http://www.modelica.org/documents/ModelicaSpec31.pdf`.

[8] H. Olsson et al. Balanced models in modelica 3.0 for increased model quality. In *Proceedings of the 6th Modelica Conference*, pages 21–33, Bielefeld, Germany, 2008.

[9] A. Pop and P. Fritzson. A portable debugger for algorithmic modelica code. In *Proceedings of the 4th Internationl Modelica Conference*, pages 435–443, Hamburg, Germany, 2005.

[10] R. Rosenberg and T. Zhou. Power-based model insight. In *Proceedings of the ASME WAM Symposium on Automated Modeling for Design*, pages 1–67, New York, USA, 1988.

[11] P. Schwarz et al. A tool-box approach to computer-aided generation of reduced-order models. In *Proceedings EUROSIM 2007*, Ljubljana, Slovenia, 2007.

[12] Ralf Sommer, Thomas Halfmann, and Jochen Broz. Automated behavioral modeling and analytical model-order reduction by application of symbolic circuit analysis for multiphysical systems. *Simulation Modelling Practice and Theory*, 16:1024–1039, 2008.

[13] R. B. Whitner and O. Balci. Guidelines for selecting and using simulation model verification techniques. Technical report, Department of Computer Science, Virgina Polytechnic Institute and State University, Blacksburg, Virginia, 1989. Technical Report TR-89-17.

[14] Y. Ye and K. Youcef-Youmi. Model reduction in the physical domain. In *Proceedings of the American Control Conference*, pages 4486–4490, San Diego, CA, USA, 1999.