Modelica – A Strongly Typed System Specification Language for Safe Engineering Practices

Peter Fritzson, Vadim Engelson, Andreas Idebrant[†], Peter Aronsson,
Håkan Lundvall, Peter Bunus, Kaj Nyström
Department of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden
Email: petfr@ida.liu.se

[†]MathCore Engineering AB, Teknikringen 1B, SE-583 30 Linköping, Sweden
www.mathcore.com

1 Abstract

Recent years have witnessed a significant growth of interest in modeling and simulation of engineering application systems. A key factor in this growth has been the development of efficient equation-based simulation languages, with Modelica as one of the prime examples. Such languages have been designed to allow automatic generation of efficient simulation code from declarative specifications. A major objective is to facilitate reuse and exchange of models, model libraries, and simulation specifications.

The Modelica language and its associated support technologies have achieved considerable success through the development of domain libraries in a number of technical areas. By using domain-libraries complex simulation models can be built by aggregating and combining sub-models and components from various physical domains.

The concept of *safe engineering practices* has been one of the most important guidelines when designing Modelica. This made it natural to make Modelica a statically strongly typed language, which allows the compiler to check the consistency of a design before it is executed, in contrast to dynamically typed languages such as Matlab.

The ability of static checking has also influenced the design of conditional equations and the ongoing the design of variant handling features in Modelica. Moreover, the language allows support for standardized physical units, thus enabling tools for unit checking of relationships and connections between interfaces. A third possible level of checking is through design rules within application-specific libraries, which can be enforced via assert statements. These properties taken together gives a good foundation for safe engineering practices, even though more work is needed to further increase the safety quality level.

2 Background on the Modelica Language

In the fall 1996, work started towards standardization and unification of multi-domain (multi-physics) modeling based on object oriented mathematical modeling techniques by defining a model description language Modelica for modeling dynamic behavior of engineering systems, intended to become a de facto standard. In November 2003, version 2.1 of the Modelica language was released, which was presented together with a large number of industrial applications during the 3rd International Modelica Conference arranged by PELAB, Linköping University, Sweden. Modelica is superior to current technology mainly for the following reasons:

- *Acausal modeling*. Modeling is based on equations instead of assignment statements as in traditional input/output block abstractions. Direct use of equations significantly increases re-usability of model components, since components adapt to the data flow context in connection structure in which they are used.
- *Object-oriented physical modeling of multiple domains*. This technique makes it possible to create model components that correspond to physical objects in the real world, in contrast to established techniques that require conversion to signal blocks. For application engineers, such ”physical” components are particularly easy to combine into simulation models using a graphical editor. The object-oriented methodology is employed to support hierarchical structuring, reuse, and evolution of large and complex models.
- The Modelica approach to multi-physics simulation enables *real-time simulation* with short deadlines not possible using loosely coupled approaches to multi-physics through connection of present simulation applications for different application domains.

The following figure shows hierarchical component-based modeling using the Modelica technology, with hierarchical decomposition of a design, and strongly typed checkable connections between system components.

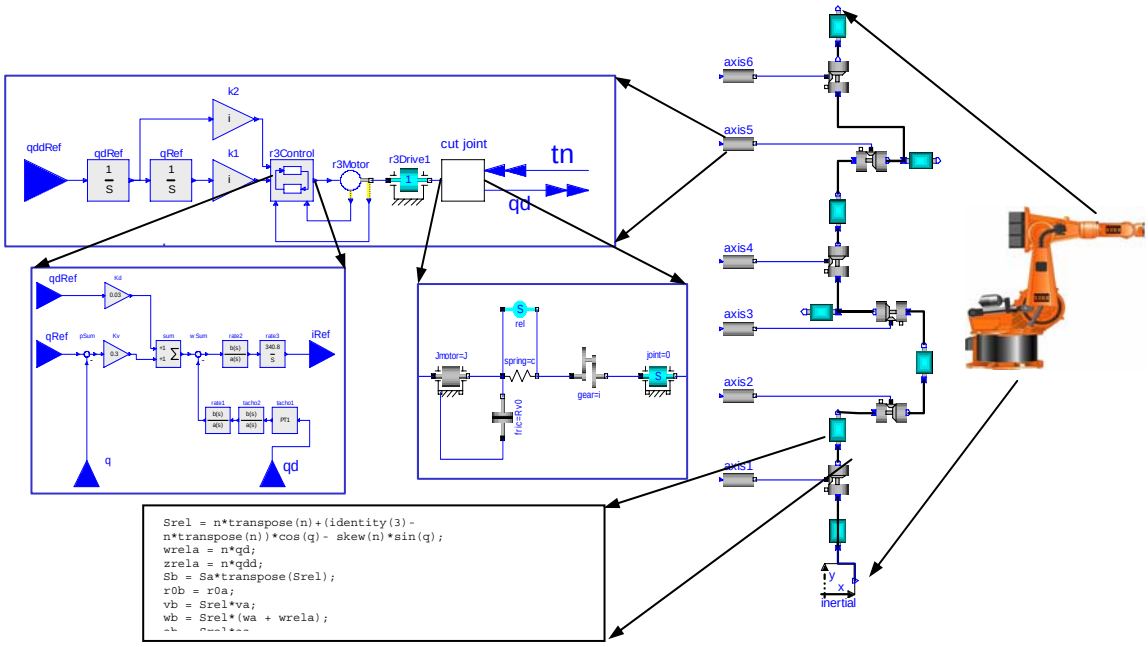

Figure 1. Hierarchical model of an industry robot, including components such as motors, bearings, control software, etc. At the lowest (class) level, equations are typically found.

3 Safe Engineering Practices and Checkable Models

What do we mean by the term safe engineering practices? When constructing a system such as a car, a train, or a nuclear power plant, we would like the design to be safe in the sense that it should exhibit unexpected behavior that can cause accidents. One way to increase the safety of a design is to have well specified components, interfaces, and system architecture, and to be able to verify the properties of the design against formal requirements. We have two groups of verification techniques:

- *Formal verification* techniques allows the consistency of the design to be checked before it is used. Here we include static checking of type constraints for strongly typed languages, unit checking of relationships in physical systems models, and checking domain-specific design guidelines by analyzing the application model code, as in the model checking approach mentioned briefly below. If an inconsistency is found, static model debugging [2] can be used for finding the probable causes of the problem.
- *Dynamic verification* techniques, i.e. a kind of systematic testing, will systematically test the design by executing it for many combinations of design parameters that hopefully well represents the design space of the implemented system.

A third technique is to make multiple independent and redundant designs, let them execute in parallel, and use a majority vote mechanism if there are differences.

In practice, both formal verification and dynamic verification techniques are typically employed to ensure maximum safety for critical systems.

4 Graphic Model Configuration

Another aspect of safe engineering practices is reusing well-tested simulation model components through an easy-to-use graphical user interface, as depicted in Figure 2, where the tool checks that connected ports are type compatible.

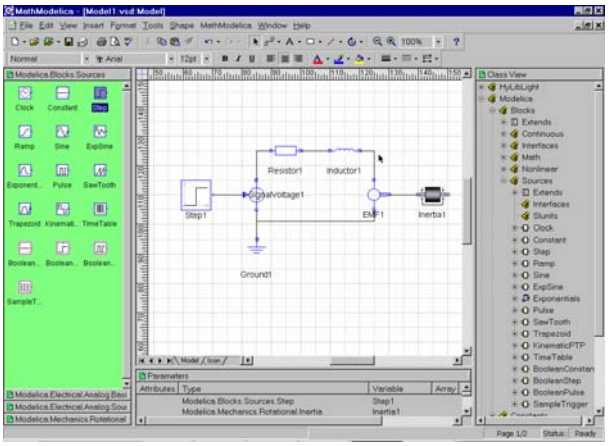

Figure 2. The MathModelica graphic model editor showing a simple electro-mechanical DC-motor model.

The MathModelica graphic model editor allows picking components from the library windows to the left, dragging these components icons into the drawing area in the middle, and connecting these by lines that represent communication or attachment between the components.

5 Wheel Loader Application

A somewhat larger application is a wheel loader, where we would like to model the lifting mechanism together with a control system, for investigation of the dynamics to ensure safe operations. The 3D design of the wheel loader mechanism was created using the SolidWorks CAD system, Figure 3 left, automatically translated to Modelica code using a CAD to Modelica translator [2], and connected to a controller modeled in Modelica.

A special library of wheel loader components was created, with model components in Modelica. The mapping from icons and connections to Modelica code is standardized and part of the Modelica language design. Connected components from this library, forming a wheel-loader mechanism, is shown in Figure 3 right.

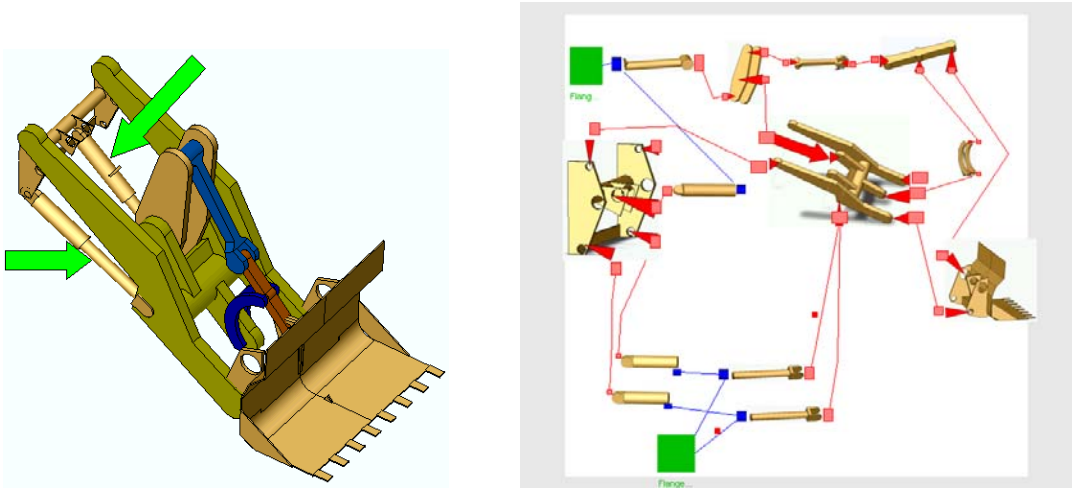

Figure 3. Left: 3D view of wheel loader mechanism created in a 3D mechanical CAD system. Right: Connected components from the wheel-loader library, forming a model of a wheel-loader mechanism.

The obtained simulation and visualization of the wheel loader mechanism can be run and controlled in real-time, e.g. to let an operator trainee interact with the wheel loader.

6 Flight Dynamics Library with Applications to Fighter Aircrafts

The Aircraft library is a versatile Modelica library for modeling and simulating aircraft dynamics, e.g. to test different kinds of controllers and other components in an aircraft. It has been developed in cooperation between MathCore Engineering AB and FOI – the Swedish Research Institute. The project was financed by FMV (Swedish Defence Material Administration) and conducted by FOI. The library is a property of FOI.

The library is structured into a number of sublibraries which contain models for describing the aerodynamics, atmosphere aerodynamic impact on aircraft, bodies, engine models, coordinate transformation models, etc. The library is easy to use, and automatically computes up-to-date center-of-mass and moment of inertia depending on the mass and position of components included in the application model. External components written in C can be included, e.g. external controller models.

The Aircraft library has successfully been used to model the flight dynamics of a generic aircraft, whose visual appearance has superficial similarities with the Swedish JAS Gripen aircraft, using the MathModelica tool for modeling, simulation, and 3D visualization. Below is an aircraft icon in the library.

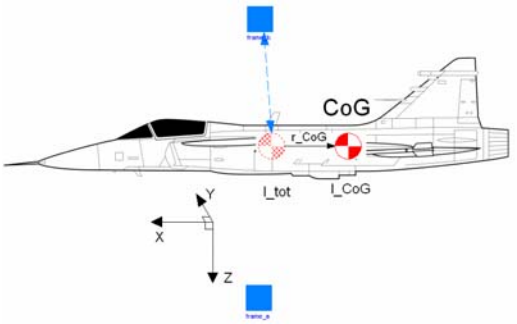

Figure 4. Aircraft with variable center of gravity and mass moment of inertia.

The `Aircraft` library currently consists of the following subpackages: `Aerodynamics` (see Figure 5 below), `Atmosphere` – for aerodynamic impact on aircraft, `Bodies` – aircraft bodies without aerodynamic influence, `Examples` – Example models, `Functions`, `Icons`, `Interfaces` – connector interfaces, `Propulsion` – engine models, `Test` – Test models to verify behavior, `Transformations` – transformation models for frames, `Types`, and finally `Utilities`.

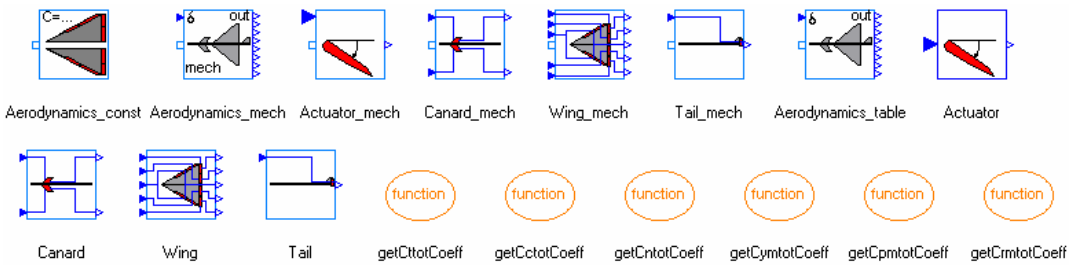

Figure 5. Aerodynamics subpackage of the AirCraft library.

7 Formal Verification

Formal program verification is a technique that aims at proving that programs meet certain specifications, i.e. that the actual program behavior fulfils certain specified properties. Model checking is a specific approach to verification of temporal properties of reactive and concurrent systems. Formal verification is usually carried out by using model checking algorithms to demonstrate the satisfiability of certain properties formalized as logical formulae over the model of the system.

Modeling with components specified in modeling and simulation languages is sometimes difficult because many semantic properties that should be obeyed during the design are not formalized in the modeling language. There exist rules that users of the components should follow in order to create semantically, mathematically, and physically correct models.

The model checking approach has however proven successful for models based on finite-state automata and is based on state space inspection. In our current research we aim at a general automated model checking framework for hybrid models specified in a non-trivial subset of Modelica, see [10].

8 Connections to HLA and GRID Technology

The HLA – High Level Architecture is a framework standard originated by the American Department of Defense [9] for supporting reusability and interoperability of simulation models and model components, especially on distributed heterogeneous computing platforms, often with real-time operation.

In comparison, the Modelica standard also supports reusability and interoperability, but to a greater extent, with easier-to-use graphic component composition, and also in a much stricter way by automatically ensuring tightly-coupled interoperability even for hard real-time deadlines; safer interconnectivity through a uniform strongly typed language; a higher degree of reusability through acausal equation-based classes that fit into any data-flow context; and a higher level of specifications since equations can be directly specified instead of manual coding in languages like C++ or similar.

One issue emphasized by HLA is operation on distributed platforms. Such mechanisms are currently being developed for Modelica in the context of the GRIDModelica project, to

support efficient model configuration and simulation on multiprocessors or distributed systems of processors connected into a GRID, which also is relevant for HLA.

9 Conclusions

Modelica is a strongly typed equation-based modeling language, with a high potential for supporting safe engineering practises through properties such as being declarative, strongly typed, amenable to formal processing such as verification, providing architectural support for complex system modeling, etc.

The language has proven itself in complex industrial applications such as wheel-loader models, aircraft dynamics models, industry robots, etc. A generalization of the language and its run-time mechanisms called GRIDModelica, is showing promising results in providing interoperability and interfacing of model components also in a distributed parallel multi-processor computational HLA-like context.

10 References

[1] Peter Bunus. *Debugging Techniques for Equation-Based Languages*, Ph.D. Thesis No. 873, June 2, 2004, Dept. of Computer and Information Science, Linköping University, Sweden..

[2] Vadim Engelson, Peter Bunus, Lucian Popescu, and Peter Fritzson. Mechanical CAD with Multibody Dynamic Analysis Based on Modelica Simulation. In *Proceedings of the 44th Scandinavian Conference on Simulation and Modeling* (SIMS'2003), available at www.scan-sims.org. Västerås, Sweden. September 18-19, 2003.

[3] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. 940 pp. Wiley-IEEE Press, Dec 2003. (Book web page: www.mathcore.com/DrModelica)

[4] Andreas Idebrant, Peter Fritzson, Martin Hagström. AirCraft – A Modelica Library for Aircraft Dynamics Simulation. In *Proceedings of the 5th EuroSim Congress on Modeling and Simulation*, Paris, Sept 6-10, 2004.

[5] IEEE P1526.1. M&S HLA – Federate I/F Spec, Draft. July 1998.

[6] Håkan Lundvall, Peter Bunus and Peter Fritzson. Automatic Generation of Model Checkable Code from Modelica. In *Proceedings of the 45th Scandinavian Conference on Simulation and Modeling* (SIMS'2003), soon available at www.scan-sims.org. Copenhagen, Denmark. September 23-24, 2004.

[7] MathCore Engineering AB. *MathModelica User's Guide*. www.mathcore.com. 2003.

[8] MathCore Engineering AB. *MathModelica CAD Users Guide*. Internal draft. www.mathcore.com. 2003.

[9] Modelica Association. www.modelica.org.

[10] Kaj Nyström, Peter Aronsson, and Peter Fritzson. GridModelica – A Modeling and Simulation Framework for the GRID. In *Proceedings of the 45th Scandinavian Conference on Simulation and Modeling* (SIMS'2003), soon available at www.scan-sims.org. Copenhagen, Denmark. September 23-24, 2004.

[11] PELAB. Modelica Research web page, Linköping University, www.ida.liu.se/labs/pelab/modelica.

[12] Pnueli, M. Siegel, and O. Shtrichman. The Code Validation Tool (CVT)- Automatic Verification of a compilation process. In *Software Tools for Technology Transfer*, Volume 2, 1999.

[13] Michael Tiller. *Introduction to Physical Modeling with Modelica*. 360 pp. Kluwer, Dec 2001.

[14] L. Zuck, A. Pnueli, Y.Fang, B. Goldberg, and Y.Hu. Translation and Run-Time Validation for Optimized Code. In Proc. of *Workshop on Run-Time Verification* (RV), July 2002.