OMJulia: An OpenModelica API for Julia-Modelica Interaction

Bernt Lie[1], Arunkumar Palanisamy[2], Alachew Mengist[2], Lena Buffoni[2], Martin Sjölund[2], Adeel Asghar[2], Adrian Pop[2], Peter Fritzson[2]

[1]University of South-Eastern Norway, Porsgrunn, Norway, Bernt.Lie@usn.no;
[2]Linköping University, Linköping, Sweden, Peter.Fritzson@liu.se

Abstract

Modelica is an object oriented, acausal equation-based language for describing complex, hybrid dynamic models. From a control systems point of view, the support of models with inputs and outputs is of particular importance. About ten Modelica implementations exist, of which most are commercial and two are open source; the implementations have varying levels of tool functionality. Many Modelica implementations have limited support for model analysis. It is therefore of interest to integrate Modelica tools with a powerful scripting and programming language, such as Julia.

Julia is a modern and free language for scientific computing. Julia has very good support for plotting, linear algebra, random numbers/statistics, automatic differentiation, optimization, machine learning, differential equations, signal processing, data frames, graph algorithms, file handling/databases, etc. Although support for control tools is lacking compared to MATLAB, control packages are more developed and simpler to install than, e.g., in Python. In summary, integration of Modelica with Julia facilitates many needed analysis possibilities and can speed up the development of effient simulation models.

A number of design choices for interaction between Julia and Modelica tools are discussed. The simplest approach is to interact via text strings of command code. A more convenient approach is to use an API in Julia (the script tool) which hides the interaction code details. For simulation, both of these approaches lead to interaction between Julia and compiled Modelica code, with some resulting run-time overhead in the call. A third approach could be to translate Modelica code into Julia code instead of C code. The produced Julia code can then be included in a Julia session, and can take advantage of Julia tools with no run-time overhead for the simulation. A fourth possibility is to utilize meta programming capabilities of Julia and extend Julia with the possibilities of Modelica (as in the Modia project). Some advantages and disadvantages of the approaches are discussed.

In this paper, the second approach is taken, and Julia package OMJulia is introduced with an API for interaction between OpenModelica and Julia. Some discussion of the reasoning behind the OMJulia design is given. The API is based on a new class *ModelicaSystem* within package OMJulia, with systematic methods which operate on instantiated models. OMJulia supports handling of FMU and Modelica models, setting and getting model values, as well as some model operations such as simulate and linearize. Results are available in Julia for further analysis.

OMJulia is a further development of a previous OMPython package; a key advantage of Julia over Python is that Julia has better support for control engineering packages. OMJulia represents a first effort to interface a relatively complete Modelica tool to Julia, giving access to an open source set-up for modeling and analysis, including control synthesis, easily installable from a unified package manager.

In addition to documenting OMJulia with some basic examples, slightly more advanced examples are included to illustrate the possibilities of implementing dynamic models in OpenModelica and carry out control systems analysis in Julia. Because Python has poor design for control systems analysis, Modelica-Julia interaction is more intersting for control applications. The examples illustrate use of Julia for linearization of Modelica models, control analysis, controller synthesis, and comparison of the control design.

Although not shown in the paper, the Modelica-Julia integration makes it straightforward to do state estimation (random number generators, linear algebra), test out optimal control and model predictive control (control systems package, optimization code), develop surrogate models (machine learning), carry out structural analysis (graph theory algorithms), etc. Some of the methods take advantage of OpenModelica's algorithm for translating DAE models to state space models.

The Julia API/OMJulia makes it possible to utilize a mature Modelica implementation (OpenModelica) out of the box, and add tools that are not part of Modelica. The approaches for tighter integration of Modelica with Julia (Modelica-to-Julia translation, Modia) are, of course, also interesting — tighter integration promises better performance compared to the OMJulia approach. These tighter integration approaches are currently limited in scope, but are interesting developments for the near future. *Keywords*: Modelica, FMI, FMU, OpenModelica, Julia, Julia API, OMJulia