Model-based control with FMI and a C++ runtime for Modelica

Rüdiger Franke[1] Marcus Walther[2] Niklas Worschech[3] Willi Braun[4] Bernhard Bachmann[4]

[1]ABB, ruediger.franke@de.abb.com, [2] TU Dresden, marcus.walther@tu-dresden.de,
[3] Bosch Rexroth, niklas.worschech@boschrexroth.de,
[4] FH Bielefeld, {willi.braun, bernhard.bachmann}@fh-bielefeld.de

Abstract

Modelica describes physical systems on a high level, using model objects, multi-dimensional arrays and other data structures as well as graphical representations. Modelica models are translated to differential-algebraic equation systems and compiled to executable code prior to their execution in numerical solvers. The translation gives a lot of possibilities for code optimization. This is particularly important for model-based control applications.

This paper investigates the exploitation of C++ for Modelica code optimization. C++ supports advanced programming concepts and at the same time aims to "leave no room for a lower-level language … (except for assembly code in rare cases)" (B. Stroustrup: The C++ Programming Language, 2014). The features exploited here include polymorphism, templates, built-in exception handling and object destructors.

The ideas have been implemented in the OpenModelica C++ runtime. The paper describes its enhancement with new array features and with an FMI 2.0 interface. FMI serves as interface between modeling tools and control applications. In particular the new FMI 2.0 meets requirements of numerical optimization solvers in model-based control.

A publically available application example demonstrates the achievements. CPU times obtained with the OpenModelica C++ runtime are significantly faster than CPU times obtained with the C runtime or with Dymola.

Keywords: Modelica, OpenModelica, FMI, C++, model-based control, MPC, MHE, SQP, HQP.

Application example

The findings are demonstrated on a DrumBoiler example that bases on the Modelica Standard Library. Exploiting FMI 2.0, the same model is translated and exported as different FMUs using different Modelica tools, tool options and compiler flags.

Table: Obtained CPU times using different Modelica tools and tool options for FMI export

Modelica Tool for FMU export	CPU time with gcc flag		
	-O0	-O2	-Ofast
OpenModelica 1.9.3	16.6 s	15.5 s	**13.5 s**
OpenModelica 1.9.3 +cseCall	6.0 s	5.5 s	**5.2 s**
Dymola 2015FD01	3.4 s	1.7 s	**1.3 s**
OpenModelica 1.9.3 +simCodeTaget=Cpp	5.6 s	1.9 s	**1.0 s**
OpenModelica 1.9.3 +simCodeTaget=Cpp +cseCall	2.7 s	1.0 s	**0.6 s**

The widely used optimization solver HQP provides FMI import. It solves the same startup optimization program for each FMU. Amazing runtime differences of more than a factor of 25 are observed for the example, depending on how the FMU was generated.

The paper describes the optimization approach giving that much flexibility and investigates why the C++ runtime of OpenModelica is superior.

The described optimization technology serves as basis for model-based control in many industrial applications.

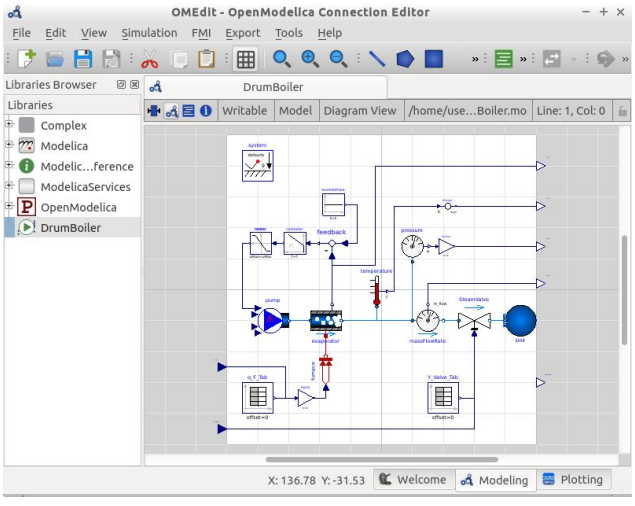

Figure: DrumBoiler model in OMEdit