Thermodynamic Property and Fluid Modeling with Modern Programming Language Constructs

Martin Otter[1] Hilding Elmqvist[2] Dirk Zimmer[1] Christopher Laughman[3]

[1]DLR - Institute of System Dynamics and Control, Germany
{martin.otter, dirk.zimmer}@dlr.de
[2]Mogram AB, Magle Lilla Kyrkogata 24, 223 51 Lund, Sweden, Hilding.Elmqvist@Mogram.net
[3]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, laughman@merl.com

Abstract

Modelica is used extensively to model thermo-fluid pipe networks. Experience shows that Modelica models in this domain have limitations due to missing functional expressiveness of the Modelica language. In this paper, a *prototype* is described that demonstrates how thermodynamic property and thermo-fluid pipe component modeling could be considerably enhanced via modern language constructs. This prototype is based on the Modia modelling and simulation prototype and relies on features of the Julia programming language. It utilizes some key ideas of Modelica.Media, and part of Modelica.Media was semi-automatically translated to Julia.

Thermodynamic property models (abbreviated as Media models below) require a great deal of flexibility with regards to the choice of thermodynamic and dynamic states to achieve robust and fast simulations. These medium models need functions to describe thermodynamic relationships with different inputs and differential equations to describe dynamic behavior. When such medium models using the Modelica language were first introduced, the only mechanism available that satisfied these requirements was that of a replaceable Modelica package. Special constructs for functions were also added to enable media modeling. This use of packages was not part of the initial language design, however, as they were primarily intended for the organization of model components. As a result, compilers typically handle packages completely at compile time. This fact has several significant implications, such as the restriction from changing the medium during simulation or the level of detail of the medium model during simulation.

This paper investigates alternative media and fluid modelling architectures available in the modern programming language Julia. Mechanisms of interest instead of replaceable packages include member functions, function references and multiple dispatch of functions[1]. The resulting architecture provides more dynamic flexibility and uses common language constructs so that it is easier to understand and maintain.

The design of the fluid library prototype for Modia is based on a new approach by (Zimmer et al. 2018)[2]. This approach is currently used in aircraft industry and enables the robust modeling of fluid streams and avoids the creation of large non-linear equation systems that are still a major source of problems for conventional fluid libraries in Modelica. For example the following part of a model:

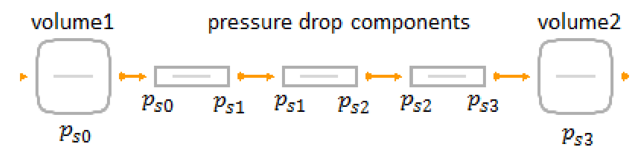

Figure 1. Three pressure drop components connected between two volumes ($\boldsymbol{p_{si}}$ is the static pressure at the indicated location).

would result in nonlinear-equation systems with Modelica.Fluid, since nonlinear pressure drop components are present without volumes in between. With this new approach used within the Modia fluid library, only one linear equation system with constant coefficients appears and can therefore more robustly solved.

In the Modia fluid library prototype, a Medium is an instance of a Julia data structure and the reference to the instance is treated as Modia variable that is propagated through connections.

In the Modelica.Fluid package there are many options that can be set on component level or globally. In the Modia fluid library prototype the complexity of the code and of the options is drastically reduced, by only providing the dynamic momentum balance, only describing pressure drop components as function of mass flow rate and having only one discretization scheme for a pipe. Still the simulation is potentially more robust as when defined with Modelica.Fluid, because no nonlinear algebraic equations occur if pressure drop components are connected together without a volume in between.

[1] Multiple dispatch in Julia means that method selection is based on the types of all non-optional function arguments (if possible at compile-time, otherwise at run-time).

[2] https://elib.dlr.de/120701/