ExternalMedia: A Library for Easy Re-Use of External Fluid Property Code in Modelica

Francesco Casella[1] Christoph Richter[2]
[1]Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
[2]Institut für Thermodynamik, TU Braunschweig, Germany
casella@elet.polimi.it ch.richter@tu-bs.de

Abstract

The modeling of thermo-physical fluid properties is of great importance when modeling thermo-fluid systems. The Modelica Standard Library provides a number of medium models that can be used in component models but are not sufficient in many applications. This paper presents a new interface library with a Modelica front-end that allows for an easy inclusion of external fluid property code in Modelica using the standard interfaces provided in the Modelica.Media library. The new library was developed as an open-source project and is available for free from the Modelica website including an interface to the FluidProp software developed and maintained at TU Delft. The new library can easily be extended to other external fluid property code.

Keywords: external fluid property code; Modelica.Media; thermo-fluid systems

1 Introduction

Modelica is finding more and more applications in the field of thermo-fluid system modeling due to the many advantages of the object-oriented equation-based approach. A fundamental problem in this field is the availability of good Modelica models for the computation of fluid properties. The Modelica.Media library was included in the Modelica Standard Library in version 2.2. It currently provides several ready-to-use models for ideal gases, mixtures, water/steam, moist air, table-based incompressible fluids, and generic linear fluid models which can be used in a wide range of applications. The library and some applications are described in [1] and [2]. However, there exists a large class of engineering systems such as refrigeration systems, heat pumps, or organic Rankine cycles that require accurate models of application-specific two-phase fluids which are currently not provided in the Modelica Standard Library.

One possibility to overcome this limitation is to write the required medium models in Modelica, possibly by conforming to the Modelica.Media interfaces for greater compatibility. The advantage of this approach is that self-contained Modelica models are obtained that can be optimized for efficiency. The major drawbacks are that writing such code requires a sizable investment in terms of time and effort, and that the developed code can only be re-used in a Modelica context.

The other possibility is to take advantage of existing fluid property code developed for general-purpose applications and to interface that code to Modelica. This approach offers a couple of unique advantages compared to a Modelica-internal solution:

- Many existing fluid property codes are well-tested and used in a number of commercial applications and products.
- Many existing fluid property codes provide fast and robust solvers for the inverse iteration of fluid properties.
- External fluid property codes can be used in a number of different software tools such as simulators, office programs, and post-processing tools.

Some existing publications such as [3] show that interfacing external fluid property code from Modelica is a feasible alternative to Modelica-internal solutions. This solution becomes extremely interesting if the effort of developing the interface for any given external fluid property code is kept to a minimum. The ExternalMedia library was developed with this objective in mind. The current implementation considers two-phase, single-substance fluids since this combination already covers many interesting applications that cannot be developed using existing Modelica.Media models. Fluid mixtures might be supported in the future.

The goals of the ExternalMedia library can be summarized as follows:

- The new medium models shall be 100% compatible to the Modelica.Media interface.
- The new interface library that handles all external fluid property codes should work with all available Modelica tools and C/C++ compilers.
- The effort to interface new external fluid property codes should be kept as little as possible.
- The new approach shall be numerically efficient to be comparable with current Modelica-internal solutions.

The new library including all source code will be released on the Modelica website and will be made available under the Modelica license.

2 Architecture of the Library

The new fluid property library consists of three main parts: A Modelica front-end called ExternalMedia, an interface layer written in C, and an object-oriented interface library called ExternalMediaLib written in C++ that handles a number of external fluid property codes.

The Modelica front-end of the new library is the ExternalMedia library, whose class structure is illustrated in Figure 1. This Modelica library contains a package named ExternalTwoPhaseMedium that extends from PartialTwoPhaseMedium defined in Modelica.Media.Interfaces. The ExternalTwoPhaseMedium package is generic. The actual external fluid property code used is specified by setting the values of suitable string constants in the medium package. The libraryName specifies the name of the external fluid property code to be used whereas the substanceName defines the name of the substance from this external fluid property code. The mediumName defined in the PartialMedium package in the Modelica.Media library is also passed to the interface library but is not used for the specification of the fluid. The new external medium model can be used in any component model that uses a medium package extending from PartialTwoPhaseMedium.

A set of functions in the ExternalTwoPhaseMedium package corresponds one-to-one to C-functions defined in the C interface layer. These functions are called according to the external function mechanism as defined in the Modelica language specification. The interface layer functions manage a collection of C++ objects that define the interface to the external fluid property codes. A class diagram of this part of the new library is shown in Figure 2.

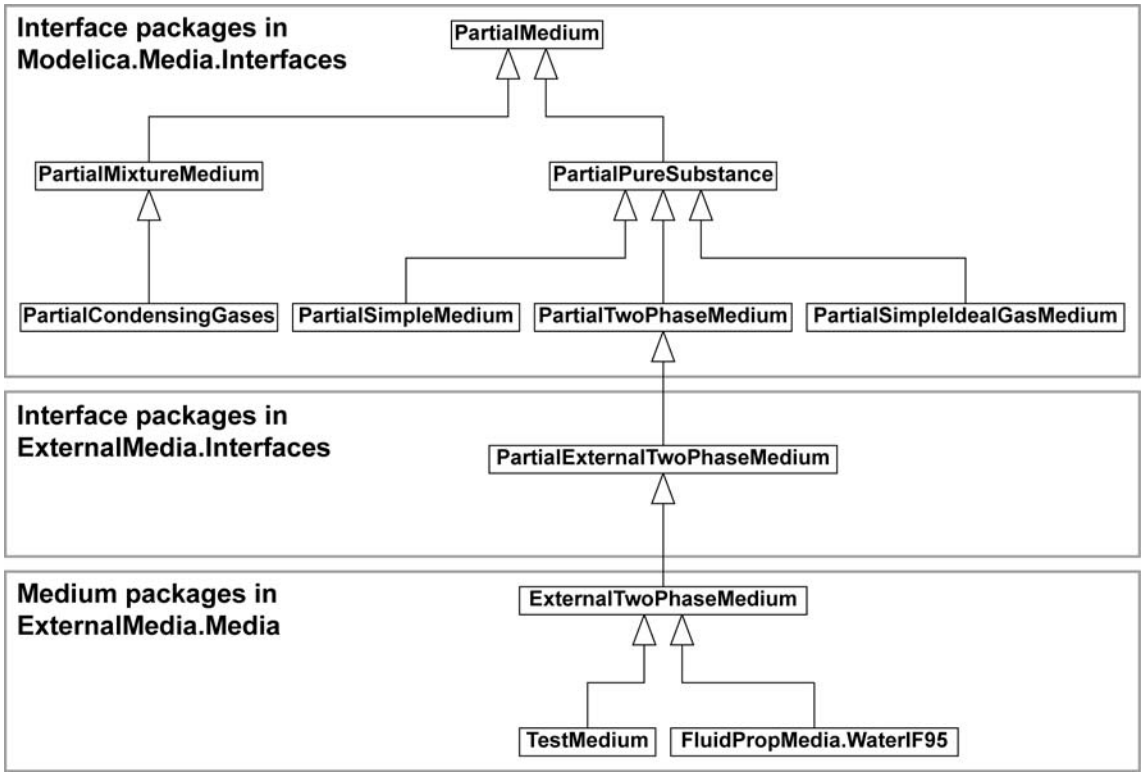

Figure 1: UML class diagram of packages in Modelica.Media and ExternalMedia library.

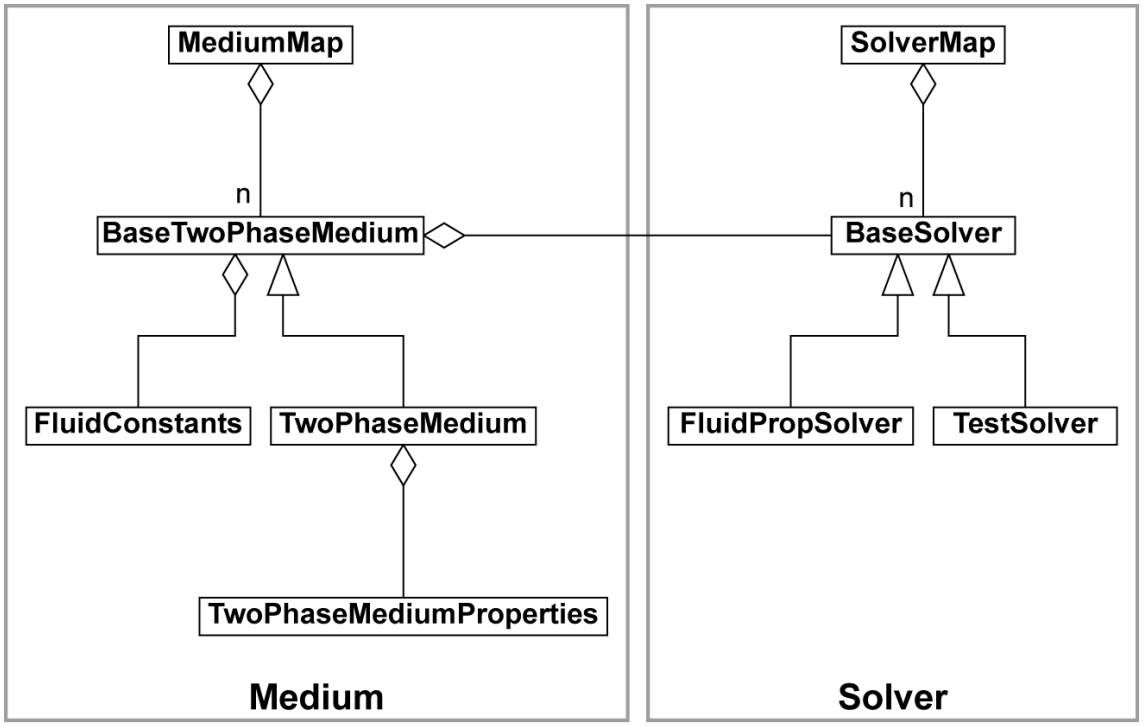

Figure 2: UML class diagram of C++ objects in the ExternalMediaLib library.

The first fundamental object is the Solver object that encapsulates the external fluid property code. In order to manage several different solvers at the same time, the interface layer defines the map SolverMap which is a collection of Solver objects indexed by the strings defined in the ExternalTwoPhaseMedium package. Each time an external function is called, these strings are passed as arguments. This allows for an instantiation of the corresponding solver when the function is called the first time and for the interface layer to point to the correct solver in any subsequent function call.

The second fundamental object is the TwoPhaseMedium object which corresponds with a point in a thermodynamic phase diagram such as a pressure-enthalpy diagram or a point on the saturation curve for saturation properties. Each TwoPhaseMedium object contains a pointer to the corresponding Solver object and a record of type TwoPhaseMediumProperties which is used as a cache record containing all possible thermodynamic properties including transport properties. All instances of these objects are stored in the map MediumMap which is indexed by an integer called uniqueID.

In order to understand how the library works, consider the following code snippet:

```
import SI = Modelica.SIunits;

package Toluene
  extends ExternalTwoPhaseMedium(
    mediumName="Toluene",
    libraryName="REFPROP",
    substanceName="Toluene");
end Toluene;

model Example
  Toluene.ThermodynamicState state;
  SI.Density d;
  SI.SpecificEnthalpy h;
equation
  state = Toluene.setState_pT(1e5,300);
  d = Toluene.density(state);
  h = Toluene.specificEnthalpy(state);
end Example
```

The setState_pT() function of the medium package calls the corresponding C function of the interface layer, passing the values of pressure and temperature as well as the three medium identification strings. If those strings are not already present in the SolverMap, an instance of the corresponding solver (in this case REFPROP [4]) is added to the SolverMap. Subsequently, an instance of TwoPhaseMedium is added to the MediumMap and the setState_pT() function of the Solver is called to compute all fluid properties. The computed fluid properties are stored in the TwoPhaseMediumProperties object that acts as a cache record. Finally, a unique identification number is returned to identify the TwoPhaseMedium object in the MediumMap. This number is stored in the ThermodynamicState record together with the values of pressure, temperature, density, specific enthalpy, and

specific entropy. Note, that the setState_pT() function in Modelica is an impure function since it returns a different uniqueID each time it is called.

When the density() function is called, the corresponding interface layer function is called with the uniqueID stored in the ThermodynamicState record. This allows for retrieving the already computed value for the density from the TwoPhaseMedium object in the MediumMap. The same thing happens when the specificEnthalpy() function is.

At the next simulation step, when the setState_pT() function is called again, a new TwoPhaseMedium object is allocated in the MediumMap and a new unique identification number is returned. In order to avoid running out of memory, the MediumMap is used as a circular buffer with a predefined maximum number of TwoPhaseMedium objects. The size of the buffer must be large enough to accommodate all setState_XX() function calls during a single simulation step.

The most straightforward implementation of the Solver objects computes all possible fluid properties at once when a setState_XX() function is called. This often is a reasonable option since most of the CPU time is often spent on the inverse iteration while the additional cost of computing all fluid properties is small. However, it is always possible to decide that the setState_XX() functions of the Solver only compute and store some of the properties and that the additional computations are triggered when the functions to retrieve these additional properties are called. This is very often a good idea for the transport properties. This mechanism allows for avoiding unnecessarily repeated computations in a flexible way that is 100% compatible with the existing structure of the Modelica.Media package.

Note, that the call of the setState_XX() functions will usually be performed before the other function calls because of the BLT partitioning of equations performed by the Modelica compiler. If this does not happen (e.g., due to the presence of implicit equations), the property functions could be called before the unique identification number has been set, thus with a default uniqueID=0. In this case it is still possible for the interface layer to select the correct solver by using the medium identification strings and to compute the required property using the values of pressure, temperature, etc. stored in the ThermodynamicState record. The uniqueID argument is thus introduced for efficiency reasons, i.e. to avoid unnecessarily repeated computations, but is not required for the correctness of the results.

If the BaseProperties model defined in the Modelica.Media library is used to compute the medium properties, the circular buffer for the MediumMap can be avoided. A unique identification number is instead stored in each instance of the BaseProperties model. This number is set once and for all during the initialization phase. The setState_XX() functions are then called within the BaseProperties model by explicitly supplying the uniqueID. The same TwoPhaseMedium object in the MediumMap is thus used for all computations in the corresponding BaseProperties object. In order for the MediumMap to distinguish between these static unique identification numbers and the transient unique identification numbers discussed in the previous paragraphs, the former are given positive values while the latter ones are given negative numbers.

3 Implementing new Medium Models

Implementing the interface to a new external fluid property code is a straightforward task requiring a limited amount of time.

First of all a new Solver must be defined, extending from the BaseSolver. This new Solver has to implement all abstract setState_XX() functions defined in the base class. These functions will actually call the external fluid property code and store the retrieved properties in the TwoPhaseMediumProperties cache.

Then, a few lines of code must be added to the getSolver() function of the SolverMap object in order to recognize the new identification strings of the additional external fluid property code. All remaining functionality is already provided by the library framework.

4 Current Status and Future Development

The framework of the new library for the support of external two-phase single-substance medium models is complete. Two Solvers are already implemented. The first Solver, TestSolver, is a dummy fluid model roughly corresponding to cold water which can be used to troubleshoot the C/C++ and Modelica compiler setup without worrying about the actual external code. It can also be used as a starting point for new user-defined fluid property codes.

The second available Solver is an interface to the FluidProp software [5] developed and maintained at TU Delft which provides a common interface to several external fluid property codes including StanMix, TPSI, and the whole REFPROP database. FluidProp can be downloaded for free even though the REF-

PROP module requires purchasing a license from NIST. Since FluidProp is based on the proprietary COM architecture by Microsoft, the corresponding solver can only be compiled under MS Windows using a MS Visual Studio compiler, even though an extension based on open-source architectures is envisioned for the near future.

The library framework is fully compliant with standard Modelica (2.2 and 3.0) and with standard ANSI C/C++. New Solvers can thus be implemented and used within any Modelica tool, using any C/C++ compiler.

The library, including all source code, will be released under the Modelica License and will be made available on the Modelica website. The C/C++ source code is fully documented, using the Doxygen tool. Future development might include the development of new general-purpose Solvers as well as the development of an external media interface for fluid mixtures.

Furthermore, the object-based fluid property library TILFluids developed at TU Braunschweig and presented in [6] uses the code of the presented external fluid property library and provides a different Modelica interface. TILFluids also provides interfaces to other software tools such as MS Excel or MATLAB/Simulink that might be included in a future release of the ExternalMedia library.

5 Conclusions

This paper presents a new fluid property library for two-phase single-substance fluids that allows for an easy inclusion of external fluid property code in Modelica, using the standard interfaces for two-phase media defined in the Modelica.Media library. Any model designed to use models derived from these standard interfaces can therefore be used without any modification. The new library is freely available under the Modelica license, and can easily be extended by including other external fluid property codes. Further development might extend the interface to single-phase pure substances and mixture media, as well as two-phase mixture media. Interested users are welcome to use the new library in their applications and are invited to contact the authors for contributions to the project.

References

[1] H. Elmqvist, H. Tummescheit, and M. Otter. *Object-Oriented Modeling of Thermo-Fluid Systems*. In Proc. of 3rd International Modelica Conference, pages 269-286, Linköping, November 2003.

[2] F. Casella, M. Otter, K. Prölß, C. Richter, and H. Tummescheit. *The Modelica Fluid and Media library for modeling of incompressible and compressible thermofluid pipe networks*. In Proc. of 5th International Modelica Conference, pages 631-640, Vienna, September 2006.

[3] H. Tummescheit and J. Eborn. *Chemical Reaction Modeling with ThermoFluid/MF and MultiFlash*. In Proc. of 2nd International Modelica Conference, pages 31-39, Oberpfaffenhofen, March 2002.

[4] E. W. Lemmon, M. Huber, and M. McLinden. *NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP, Version 8.0. National Institute of Standards and Technology, Standard Reference Data Program*, Gaithersburg, 2007.

[5] *FluidProp: A software for the calculation of thermophysical properties of fluids*. http://fluidprop.tudelft.nl/

[6] C. Richter. *Proposal of New Object-Oriented Model Libraries for Thermodynamic Systems*. Doktorarbeit, TU Braunschweig, to be published in 2008.