

**IEA EBC ANNEX 60 MODELICA LIBRARY –  
AN INTERNATIONAL COLLABORATION TO DEVELOP  
A FREE OPEN-SOURCE MODEL LIBRARY  
FOR BUILDINGS AND COMMUNITY ENERGY SYSTEMS**

Michael Wetter<sup>1</sup>, Marcus Fuchs<sup>2</sup>, Pavel Grozman<sup>3</sup>, Lieve Helsen<sup>4</sup>,  
Filip Jorissen<sup>4</sup>, Moritz Lauster<sup>2</sup>, Dirk Müller<sup>2</sup>, Christoph Nytsch-Geusen<sup>5</sup>,  
Damien Picard<sup>4</sup>, Per Sahlin<sup>3</sup>, Matthis Thorade<sup>5</sup>.

<sup>1</sup>Lawrence Berkeley National Laboratory, Berkeley, CA, USA

<sup>2</sup>RWTH Aachen University, E.ON Energy Research Center, Aachen, Germany

<sup>3</sup>EQUA SE, Sweden

<sup>4</sup>KU Leuven, Department of Mechanical Engineering, Belgium

<sup>5</sup>UdK Berlin, Germany

## ABSTRACT

This paper describes the collaborative development of the Annex 60 Modelica library, a free, open-source library for building and community energy systems. The library is developed within the Annex 60 project that is conducted under the umbrella of the International Energy Agency's Energy in Buildings and Communities Programme (IEA EBC). Our goal is to develop and distribute a well documented, vetted and validated open-source library that serves as the core of future building simulation programs and that can be integrated with existing programs as well. The work brings together experts in Modelica for building energy applications and coordinates the previously fragmented development that led to four libraries that were incompatible, hard to combine and each itself limited in scope. The work resulted in a library that is now used as the core of these four Modelica libraries. The paper describes the agreed upon requirements, scope, current status of implementation, quality control process and structure of the library. The paper also provides illustrative examples.

## INTRODUCTION

Annex 60 is a collaborative project among 38 institutes from 16 countries, conducted between 2012 and 2017 under the umbrella of the International Energy Agency's Energy in Buildings and Communities Programme (IEA EBC). Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on non-proprietary Modelica, Functional Mockup Interface and Building Information Modeling standards.

This paper describes the research and development of an open-source, free library for building and community energy systems that is jointly developed within Annex 60 using the Modelica language (Mattsson and Elmqvist, 1997). In 2012, before the beginning of Annex 60, five groups (EQUA SE Sweden, KU Leuven Belgium, LBNL USA, RWTH Aachen University Germany, and UdK Berlin Germany) developed their own Modelica libraries for building per-

formance simulation. These libraries were difficult to use with each other, in some cases incompatible, and there was a significant duplication of effort. In 2013, the decision has been made by the Annex 60 team to form a group that joins efforts to avoid such a fragmented development. Subsequently, collaborative development started on a free, open-source library which is hosted at <https://github.com/iea-annex60/modelica-annex60> and which is described in this paper. The anticipated outcome is an open-source, freely available, documented, validated and verified core Modelica library that is used by the above five libraries, and hopefully, by other whole building simulation programs. A goal of the library is to provide models that support the design and operation of building and community energy systems as integrated, robust, performance based systems with low energy use and low peak power demand while maintaining thermal comfort. Our goal is to set a foundation for the development of models, and implement a model library, that can serve the building simulation community over the next decades, and that will be adopted by various building simulation programs in addition to the above Modelica libraries.

This effort is also expected to support realizing proposition 6 of the vision spelled out in the position paper of Joe Clarke that was written on behalf of the IBPSA board (Clarke, 2015). Specifically, proposition 6 states:

IBPSA will encourage manufacturers to provide more fundamental descriptions of components and make these available within a standard library.

We believe that the here presented work could serve as the basis of such a standard library. Proposition 6 of the same paper also states IBPSA's support for "... a library of control system template definitions to representative cases." Although adding control templates is currently a lower priority, we believe with the Annex 60 library to have created a framework that allows populating such a control template library.

This paper discusses how the Annex 60 library is being developed and it discusses the current status of development. As of today, we developed a library with a core functionality and merge it automatically into individual Modelica libraries for building performance simulation. The Annex 60 library is now used by `AixLib` from RWTH Aachen (Constantin et al., 2014; Lauster et al., 2014), `Buildings` from LBNL Berkeley, CA (Wetter et al., 2014), `BuildingSystems` from UdK Berlin (Nytsch-Geusen et al., 2013), and `IDEAS` from KU Leuven (Baetens et al., 2012).

The approach of the library distribution is modeled after Linux, which has a kernel that is used by different distributions (e.g., Ubuntu, RedHat etc.), which then provide installation packages, user support and detailed documentation. In a similar fashion, the Annex 60 library provides reliable base classes for building and HVAC component models. Thus, the individual model libraries of participating institutions can be further developed, each with their specific focus, while compatibility between the libraries is ensured by the use of the common base classes from the Annex 60 library. In addition to the advantages of having a reliable and well-tested common foundation for model development, we expect further benefits of this approach from increased compatibility, exchange and collaboration as opposed to the previously fragmented development of model libraries.

## CONVENTIONS AND NOMENCLATURE

This section defines various technical terms that are used throughout the paper and that may not be clear to readers that are not familiar with the Modelica language.

A *package* is a collection of models, functions, or other packages that are used to hierarchically structure the library. A *class* is a term that includes models, blocks, functions and packages. *Connectors* are instances that define variables and parameters, and are used to connect models or blocks with each other. A *block* is a special case of a model in which all connectors are either inputs or outputs. In a *model*, the causality of a connector need not be specified. The value of a *parameter* cannot change with time. In contrast, the value of an *input signal* or a *variable* may change with time.

## FUNCTIONAL REQUIREMENTS

The library should fulfill several functional requirements that originate from typical use cases in building performance simulation. The use cases include warm water heating and distribution systems, cooling systems, ventilation systems, heat demand calculations of multiple buildings, controls design, co-simulation and export of models for use in other simulators or in building automation systems. The use cases aim at optimal design and operation of thermal energy systems as well as district energy systems and model use dur-

ing operation. Thus, we focus on annual whole building simulation, on single as well as on multiple buildings.

Regarding physical resolution, various effects need to be modeled to ensure suitability for typical use cases. All mass flow rates are pressure driven to allow simulations of duct and piping networks. Optionally, it must be possible to remove the equations for pressure drop from the model. Fluid flow networks must allow adding transport delays. The media models must be compatible with models from `Modelica.Media` to ensure compatibility with other libraries. For air, it must be possible to model water vapor and trace substances such as CO<sub>2</sub> or VOC concentrations as needed for simulation of indoor air quality. For cooling devices, no detailed modeling of the state and distribution of refrigerants is conducted as this would lead to computing time that is too large for annual building simulation.<sup>1</sup>

Regarding dynamic system behavior, models with different idealizations should be implemented. As applicable, it should be possible to approximate the dynamic response of models using a first or higher order response as needed, and optionally disable the model dynamics to conduct a quasi steady-state simulation. For example, the dynamics of sensors can be approximated by a first order response. This can be disabled to obtain a steady-state sensor, or to add a more detailed sensor response to the output signal of the sensor.

To simplify the use and the workflow integration of the library, Python scripts will be provided to extract simulations results, provide typical examples that illustrate the use and allow easy integration of typical boundary conditions such as TMY3 weather data. It must be possible to export as a Functional Mockup Unit (FMU) thermofluid components for water, air and glycol with different concentrations. A simulator must not have to recompile the FMU when the medium concentration changes. All models will exclusively use SI units.

## MATHEMATICAL REQUIREMENTS

Component models from this library are typically assembled to form system models that lead to systems of ordinary differential equations, which are coupled to algebraic systems of linear, nonlinear and discrete equations. To ensure that these systems of equations can be solved efficiently, they need to satisfy certain mathematical properties. In this section, we describe the main requirements that have been followed during the development of the library. For a more detailed list of requirements, see <https://github.com/iea-annex60/modelica-annex60/wiki/Style-Guide>.

For equations that describe the physics, we require the

<sup>1</sup>For such applications, the AC library from Modelon or the TIL library from TLK-Thermo GmbH may be used.

following mathematical properties:

Equations must be differentiable and have a continuous first derivative that is bounded on compact sets, i.e., they need to be once continuously differentiable on compact sets. This is not only needed for numerical efficiency, but also to establish existence of a unique solution to the system of differential equations (Coddington and Levinson, 1955). For example, instead of using  $y(x) = \text{sign}(x) \sqrt{|x|}$ , this equation needs to be approximated by a differentiable function that has a finite derivative near zero because  $\lim_{x \rightarrow 0} y'(x) = \lim_{x \rightarrow 0} 1/(2\sqrt{|x|}) = \infty$ . Otherwise, Newton-Raphson solvers may fail if  $x$  is close to zero as the Newton step length is proportional to the inverse of the derivative of the residual function.

Furthermore, equations where the first derivative with respect to another variable is zero must be avoided. For example, let  $x, y \in \mathbb{R}$  and  $x = f(y)$ . An equation such as  $y = 0$  for  $x < 0$  and  $y = x^2$  is not allowed. The reason is that if a simulator tries to solve  $0 = f(x)$ , then any value of  $x \leq 0$  is a solution, which can cause instability in the solver. An example of such a situation is a valve. If it were to have no leakage flow, then any value of the pressure drop would cause zero mass flow rate, which may lead to ill-conditioned equations for some flow networks. More formally, the conditions for the Implicit Function Theorem (Polak, 1997) need to be satisfied as this guarantees existence and differentiability of an inverse of the function.

Clearly, models for controls require discrete variables that can only take on certain values, such as for switching equipment on or off. This certainly is allowed, but must be implemented using a hysteresis to avoid chattering. For example, an equation such as  $y = 0$  if  $T > 20^\circ\text{C}$  and  $y = 1$  otherwise is not allowed as this can lead to chattering in continuous time solvers.

## PROCESS FOR QUALITY CONTROL

The Annex 60 library is used as a foundation on which different libraries for end users are developed. It therefore needs to provide reliable models and results. Thus, a strict process for quality control was implemented from the start of the library development. This process includes open source code development in a version control system on GitHub, automated regression testing of the entire library, and a review process before authorizing any code changes. This process is supervised by a core development team consisting of the authors of this paper but requirements are documented in a project Wiki and open to contributions from the community, given that such contributions meet the set quality standards.

Using git for version control of the source code and documentation of the library allows for keeping a record of all development stages and changes of the

library. Stable models are kept in the so-called master branch. Additions and code changes are restricted to dedicated branches for feature development that are documented in a corresponding issue tracker. In order to prevent any unintended effects of code changes to existing models, the translation statistics and reference results for each model are automatically created, stored, and managed by using the Python package BuildingsPy. This package includes functions for unit testing. The implemented process for quality control requires to provide a scripted test for every model, so that these tests can be run automatically for the whole library. When introducing a model and its test, the simulation results are saved by the unit testing routine. The unit testing is run before accepting any changes to the master branch. Differences in the translation statistics and simulation results between reference results and the tested model are automatically plotted and need to be accepted or rejected manually, thus guarding against introducing unwanted effects on any of the existing models.

Once code changes have been documented and evaluated by unit testing, they need to be reviewed by a second developer. Thus, the author of the changes issues a pull request to suggest moving the new code into the master branch. Only after all comments from the reviewer have been addressed by the original author of the changes is the new code merged to the master branch. At the time of writing, this process has been applied to over 200 issues and 800 changes to the code base. Finally, the underlying version control system can serve as a safety net if changes need to be reverted despite of the described quality control process. As each step of code change as well as the responsible author is documented at all times, such corrections would require little effort.

## REQUIREMENTS FOR ADDING CLASSES

The Annex 60 library aims at following a coherent set of conventions and requirements to ease maintenance and further development. For a detailed list of requirements, see <https://github.com/iea-annex60/modelica-annex60/wiki/Style-Guide>.

The library follows and extends the conventions of the Modelica Standard library. In addition, the names of models, blocks and packages must start with an uppercase and be a combination of adjectives and nouns using camel-case to combine multiple words, whereas instances of these models, blocks and packages start with a lower case letter. Instance names are usually a character, such as T for temperature and p for pressure, or a combination of the first three characters of a word, such as TSet for temperature setpoint or higPreSetPoi for high pressure set point.

New components of fluid flow systems must extend the partial classes defined in `Annex60.Fluid.Fluid.Interfaces`. If the new class is partial or

if it is not intended for the end-user, it must be added to a `BaseClasses` package. Each new model must be used in at least one model contained in the `Examples` or `Validation` package, which illustrates or validates its behavior and is included in the unit tests.

Each class needs to be documented. Every variable and parameter must have a documentation string. The `documentation` section of the model provides a more comprehensive documentation describing the main equations, model assumptions and limitations, typical use, options, validation, implementation and references. Some of these sections are optional as they are not applicable to all models. Each class also contains a list of revisions made by the developers to keep track of the changes and their rationale.

The robustness of the models is also a key requirement, and the guidance explained in the section “mathematical requirements” needs to be followed. Fluid flow models must be stable near zero mass flow rate even under the assumption that flow rates or heat input are approximate solutions obtained using an iterative solver. Fluid flow models must be well-behaved if the mass flow reverses direction.<sup>2</sup> Furthermore, fluid flow models use the `Media` models, which have physical constraints such as the valid temperature range or relative humidity bounds outside which they are not valid. These bounds need to be taken into account by the models in order to avoid non-physical situations or convergence problems. Parameters and variables should whenever possible use units from `Modelica.SIunits` and they should declare bounds for minimal and maximal values. Default values for parameters, which can be adapted by the end-user, should be declared using the `start` attribute so that the user gets a warning if no other value has been provided.

Each model must be validated using either measurement data, cross validation with other simulators or with analytical solutions. Validation models need to be added to the `Validation` package and be included in the unit tests.

Finally, new models should be in line with the scope of the library as described in the functional requirements.

## DESIGN DECISIONS

This section describes the main design decisions for the Annex 60 library.

### **Media**

In `Modelica`, component models typically call functions to obtain thermodynamic properties such as the specific enthalpy.

We experimented with two implementations. One approach called `MediaFunctions` is using functions for the thermodynamic properties of a medium, with

---

<sup>2</sup>By well-behaved, we do not mean, for example, that a performance-curve based model of a direct expansion cooling coil computes the right physical results if there is a slight backward flow, but rather that the model is robust. For example, it suffices to add no energy to the air if there is slight backward flow.

an enumeration as a function argument that declares the medium type such as air or water. The other approach called `MediaPackages` is using a separate `Modelica` package for each medium type, as is done in `Modelica.Media`.

The main differences between these two implementations are as follows:

1. For `MediaFunctions`, models of HVAC equipment require parameters for the medium type and for the default values of pressure, temperature, mass concentration and trace substances. The medium type needs to be propagated to the functions that compute the thermodynamic properties.
2. For `MediaPackages`, there is one package for each media type, as in `Modelica.Media`. Models of HVAC equipment contain a replaceable parameter for the medium package that needs to be set to medium type. Prior to the model translation, the medium type such as water, air or glycol must be declared, but its concentration can be changed after translation.

Based on these implementations, we decided to organize media in packages as is done in `Modelica.Media`. The benefits are:

1. Full compatibility with `Modelica.Media`.
2. Default values for the medium, such as the default pressure and mass concentration, can be propagated through its declaration because packages can contain constants.
3. `Modelica` translators can verify that connected fluid ports contain the same medium, and refuse to translate the model if this is not satisfied.

The advantage of using `MediaFunctions` would be that only two sets of FMUs need to be generated, one for the single species medium water, and one for the dual-species media such as air and glycol. However, the cost of separately compiling FMUs for air and for glycol is small as this can be fully automated and be done as part of a simulation engine development.

While the implementation of `Annex60.Media` is compatible with `Modelica.Media`, we generally use simpler implementations. For example, in `Annex60.Media.Air`, water is only present in vapor form even if the water vapor pressure is above the saturation pressure. This allows to compute temperature from specific enthalpy and mass concentration without requiring iterations, thereby leading to more efficient models.

### **Fluid connectors**

We decided to use the same fluid connectors as defined in `Modelica.Fluid`. These connectors declare the medium package (used to assert that only models with the same medium are connected), the mass flow rate as the conserved quantity, the absolute pressure as the potential variable, and the following variables that are

carried by the mass flow: the specific enthalpy, optionally the mass fraction such as to declare the mass concentration for water vapor in air, and optionally trace substances. Trace substances are concentrations that can be neglected for the thermodynamic calculations such as CO<sub>2</sub> or VOC concentrations.

We also explored using temperature instead of specific enthalpy in the connector. Modelica's concept of flow and stream variables allows connecting multiple fluid ports. It then automatically generates the conservation equation  $\chi_{mix} = \sum_i \dot{m}_i^+ \chi_i / \sum_i \dot{m}_i^+$ , where  $\chi$  is the conserved quantity,  $\dot{m}_i^+ = \max(0, \dot{m}_i)$  is the mass flow rate if it is positive and the subscripts mix stands for the mixture and  $i$  for the connected ports. If  $\chi = h$  is the specific enthalpy, then  $h_{mix}$  is always correct. If  $\chi = T$  were temperature, then  $T_{mix}$  is wrong if the specific heat capacity  $c_p$  depends on temperature. We therefore selected the specific enthalpy to be in the fluid connector, as is also used in Modelica.Fluid.<sup>3</sup>

### Package structure

In Modelica, classes can be collected and organized hierarchically into packages. This section describes how the Annex 60 library organizes classes into the following main packages:

`BoundaryConditions` contains models for weather data reader, solar radiation and sky temperature.

`Controls` contains models for continuous time and discrete time controllers and for set point scheduling. `Fluid` is the main package that contains fluid flow components such as heat exchangers, pumps, valves, air dampers and boilers. We considered introducing subpackages `Fluid.{Air, Water, Glycol}` but have not done so yet as this would lead to duplication of code and documentation. Consequently, users always need to assign the media.

`Utilities` contains the major packages `Psychrometrics`, which implement blocks and functions for psychrometric properties, and `Math`, which provides blocks and functions that are once continuously differentiable approximations to functions such as  $\min : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ ,  $\text{abs} : \mathbb{R} \rightarrow \mathbb{R}$ , the Heaviside step function or cubic spline interpolation. The functions in the `Math` package are used to satisfy the earlier discussed differentiability requirements.

Major packages contain user guides, and all packages contain an `Examples` or a `Validation` package that demonstrates how to use the models, and that are used for validation and regression testing.

## MAIN CLASSES OF THE LIBRARY

The Annex 60 library provides models for simulating HVAC systems in buildings, containing both hydronic

<sup>3</sup>See also <https://github.com/iea-annex60/modelica-annex60/issues/28> for a discussion.

and air flow models. These systems combine different types of components such as energy and mass transfer models, media models for air and water, control models and supporting utilities. These types are grouped in separate packages. This section gives the highlights of these packages.

### Fluid component models

A typical HVAC system contains components such as valves, pumps, dampers and heat exchangers. Essentially these are all components that transfer mass and/or energy. These models are therefore grouped in the `Fluid` package. The most important parts of this package are now discussed.

#### Conservation of energy and mass

Since all of these models need to conserve mass and energy, the `Fluid` package heavily relies on the `ConservationEquation` model that implements these conservation laws. The model is implemented in a generic way such that it can be used for different types of media: compressible and incompressible and media with or without moisture or trace substances. The energy and mass conservation laws can be configured to be a steady state or dynamic balance. The `ConservationEquation` model is instantiated by the `MixingVolume` model, which is used by most equipment models.

#### Flow networks

Since the physics of fans and pumps is similar, they are implemented using the same models in the `Movers` subpackage. This package contains four mover models, each using a different control signal.

1. `FlowControlled_m_flow` directly sets the mass flow rate, independent of the head pressure that results from the duct or pipe network simulation.
2. `FlowControlled_dp`, directly sets a pressure head, independent of the mass flow rate that results from the duct or pipe network simulation.
3. `SpeedControlled_Nrpm` sets the speed. Mass flow rate and pressures are calculated from similarity laws, from a user-provided pump curve and the duct or pipe network simulation.
4. `SpeedControlled_y` is identical to `SpeedControlled_Nrpm`, except that the input signal is normalized by the nominal speed.

Various types of flow resistance are implemented. For example, `FixedResistanceDpM` implements a pressure curve according to  $K_v = \dot{m}_{nom} / \sqrt{\Delta p_{nom}} = \dot{m} / \sqrt{\Delta p}$ , where  $\dot{m}_{nom}$  is the nominal mass flow rate,  $\Delta p_{nom}$  is the nominal pressure drop, and  $\dot{m}$  and  $\Delta p$  are the actual mass flow rate and pressure drop. In a neighborhood around zero, this function is regularized. In valves and dampers, the  $K_v$  value is a function of the actuator input  $y$ . The library implements models in the `Actuators` subpackage where the relation between  $K_v$  and  $y$  is expressed using a linear, quick

opening or an equal percentage characteristic. Custom opening characteristics using a table can also be used, as well as a pressure independent characteristic. Various dampers, two-way and three-way valves of these types are implemented.

### Energy transfer

The library provides the model `HeaterCooler_T`, which is a heater or cooler that maintains an outlet temperature, subject to optional capacity limits. The outlet temperature is obtained from an input signal. The model `HeaterCooler_u` is a variant of this model that takes as an input a control signal that is proportional to the heat to be added to or removed from the fluid. A heat exchanger with constant effectiveness can be used to transfer energy between different fluid streams. A similar model exists for a mass exchanger that transfers moisture. The model `RadiatorEN442_2` implements a radiator model based on the EN 442-2 norm.

### Other models

Various models are available for integrating fluid components in larger models. Models from the `Sensors` package can be used for integrating control components and for performing analysis. The `Sources` subpackage implements components for enforcing boundary conditions on fluid ports. The user can configure the model to obtain the mass flow rate or pressure from a parameter or from an input signal. In a similar way, the leaving temperature, enthalpy, species or trace substance concentrations can be defined.

### Media

The library contains media implementations for water and air. `Water` is considered to be incompressible with constant thermal properties. `Air` contains moisture, has a pressure-dependent density and constant thermal properties. More detailed implementations are available in the package `Media.Specialized`.

### Control models

The `Controls` package contains basic control components such as PID controllers and blocks for set point resets. Our intention is to expand this package to provide control blocks and template control sequences that are commonly used in building control systems.

### Utilities

The `Utilities` package contains models that simplify the consistent implementation of other components. The `Psychrometrics` subpackage, for example, contains functions and models for relating the vapor fraction and partial pressure to the humidity and wet bulb temperature. The `Math` subpackage contains commonly needed once continuously differentiable approximations to mathematical functions.

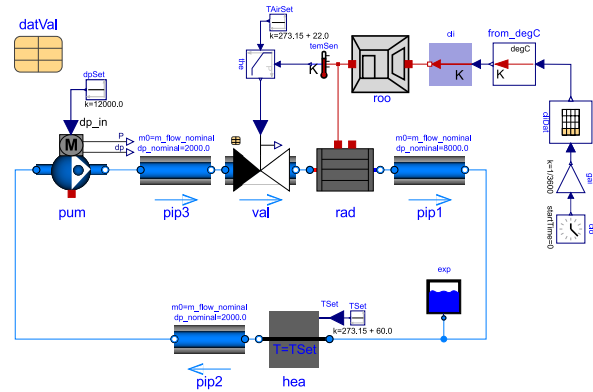


Figure 1: Schematic model view of a hydronic heating system implemented using the Annex 60 library.

## ILLUSTRATIVE EXAMPLES

This section describes two example models that illustrate the library. Both are currently being developed for numerical benchmarking.

### Hydronic heating

This system is used as a hydronic benchmark. It is kept simple to allow comparison with other simulators.

Figure 1 shows the schematic model view. The main models are a room (`roo`), a radiator (`rad`), three pipes (`pip1` to `pip3`), a valve (`val`), an ideal boiler (`hea`), an expansion vessel (`exp`), a pump (`pum`), a thermostat controller (`the`) and climate data (`cli`) for the weather data.

In this model, the pump generates a constant pressure, and the boiler produces a constant outlet temperature of 60°C. Depending on the room temperature, a thermostat, implemented using a P-controller, outputs the valve control signal to track 22°C. The valve resistance causes the flow to the radiator to vary. The outside temperature is read from a data file. The nominal value of the mass flow in the hydraulic loop is  $\dot{m}_{nom} = 0.01$  kg/s. The room model is approximated by a first-order model, and hence cannot distinguish between radiative and convective heat transfer. Thus, the radiator model transfers all heat by convection. All pipe models calculate the pressure losses, but they are adiabatic. Figure 2 shows results of a yearly simulation with a typical weather file for North Europe with cold winters and mild summers.

The computing time for an annual simulation is only a few seconds, as most components are steady state (e.g. pump or valve) or low-order (e.g. building).

### Multizone air exchange

We also implemented a model for a scalable air flow benchmark that can be used to test media implementations for air as well as the effects of increasing the problem size on the numerical solution. The basic set up of the air flow benchmark is modeled after the approach used in the Buildings library. As described in Wetter (2006), this implementation models the air flow

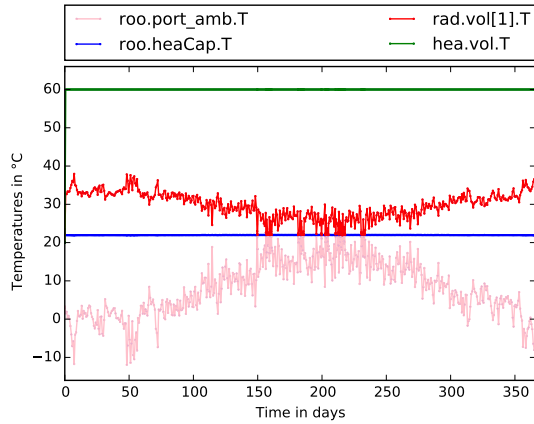


Figure 2: Annual plot of the outside, room, radiator and heater temperature.

through small orifices, large apertures as well as stack effects driven by pressure differences. The original model showed good agreement with a reference simulation of air flows in CONTAM. In that validation, a three-room model with two floors was used. Two rooms on the ground floor were modeled with an open door between them. Furthermore, small orifices in one of these rooms made a connection to the outside environment, and one orifice each connected these rooms to one single room in the upper floor. This way, the interplay of large apertures and small orifices as well as pressure differences between different heights and between inside and outside was incorporated.

The air flow benchmark in the Annex 60 library extends this test approach to be easily scalable to increasing problem sizes in order to investigate the behavior of different models as the underlying system of equations grows. Therefore, the different aspects tested in the three-room model were incorporated into similar room modules that can be automatically connected in any chosen number to form a building context for air flow tests. These modules include a single room, a hallway element, a representation of the outside environment, and a staircase element. Any number of rooms and hallway elements can be combined with one staircase element to form a scalable floor model. The staircase element serves as a connection of several such floors stacked on top of each other to form a building. Based on outside pressure conditions and indoor air temperatures for each room, the model calculates pressure differences and the resulting air flows between different rooms.

Figure 3 shows a concept drawing of the air flow benchmark. In this example, three room modules, three hallway elements, and a staircase module at the right are used for each floor. The floors are connected by the two staircase elements. The black arrows illustrate how air can flow through the staircase elements, the open doors, the hallway and its orifices, which represent leakage air flows to the outside environment.

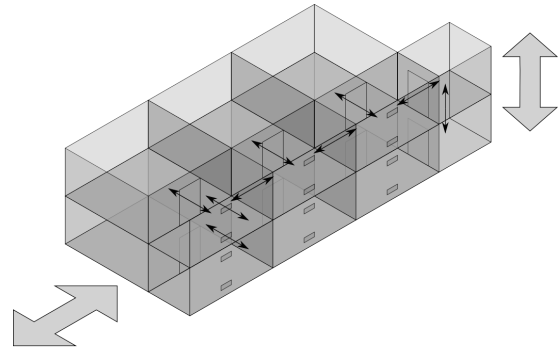


Figure 3: Concept of the scalable air flow benchmark.

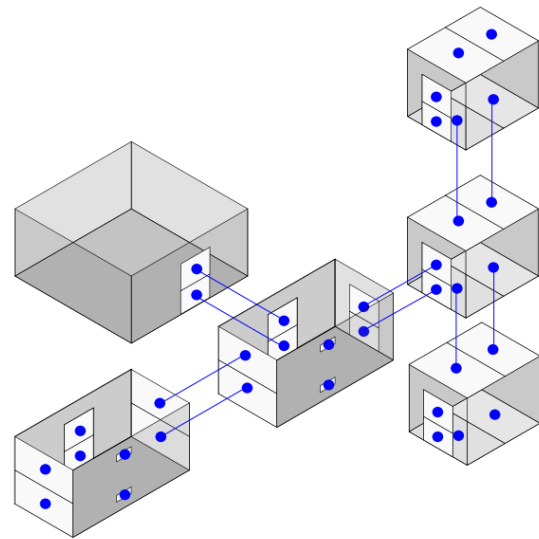


Figure 4: Fluid connections for the air flow benchmark.

The larger grey arrows show the dimensions in which the setup can be extended by adding more hallway elements and room modules, or by increasing the number of floors stacked on top of each other.

Regarding the model implementation, each module consists of one representative air volume. This volume is calculated from the dimensions of the rooms and is assumed to be located at the medium height of the floor. In order to allow for bi-directional flow through large apertures, each of these are divided into two halves of the cross section with a fluid connector each. The distribution of fluid connectors and the connections between different modules is illustrated in Figure 4. For each model, internal connections link all the fluid connectors to the air volume of the room. This setup will serve as a basis for the benchmark of different media and pressure-driven flow computations.

### MERGING WITH OTHER LIBRARIES

The Annex60 library is used as the core of the following Modelica libraries: AixLib from RWTH Aachen, which already uses the water-based models and works on integrating the air-based models, Buildings



from LBNL Berkeley, CA, BuildingSystems from UdK Berlin, and IDEAS from KU Leuven.

To automatically merge the Annex60 library with these libraries, developers of these libraries use the BuildingsPy Python package by executing the following commands:

```
1 import buildingspy.development.merger as m
2 src="/home/joe/modelica-annex60/Annex60"
3 des="/home/joe/modelica-buildings/Buildings"
4 mer=m.Annex60(src, des)
5 mer.merge()
```

These commands merge the Annex60 library from the directory `src` into the library specified by the variable `des` and update all hyperlinks, references to package names and file names that contain the Annex60 string. Therefore, users will only see the respective library and do not have to combine models from different libraries.

## CONCLUSIONS

Through the IEA EBC Annex 60 project, multiple institutes started a collaborative development of a free, open-source Modelica library for building and district energy systems. This work harmonized the previously fragmented and duplicative development of libraries with the goal to collectively develop a library that will serve the building simulation community. This required from the developers of previous libraries, each having a considerable code base, to mutually agree upon a common process for the development and quality control that needs to allow for rapid experimentation as is often done in University settings as well as ensuring robust and stable development, which is more important to commercial software companies such as EQUA and National Labs such as LBNL. It also required the developers to agree upon various design decisions and conventions for coding, documentation and validation, to jointly work on implementation and vetting of a core of a library, to refactor their existing libraries, and to open-source previously proprietary code. The team is now working on benchmarking and expansion of the scope.

To our knowledge, this is the first international collaboration for a library with free, open-source models for buildings and district energy systems that are built using an open-standard modeling language.

Our goal is that this will initiate a larger open-source development that can support IBPSA's vision of providing a standard library with fundamental model descriptions that will be supported by manufacturers and integrated in various building performance simulators.

## ACKNOWLEDGEMENT

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

We gratefully acknowledge the financial support by BMWi (German Federal Ministry of Economic Affairs and Energy), promotional reference 03ET1177A, of

the Agency for Innovation by Science and Technology in Flanders (IWT) in the frame of the PhD Fellowship of F. Jorissen and IWT and WTCB in the frame of the IWT-VIS Traject SMART GEOTHERM.

This work emerged from the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and BIM standards.

## REFERENCES

- Baetens, R., De Coninck, R., Van Roy, J., Verbruggen, B., Driesen, J., Helsen, L., and Saelens, D. 2012. Assessing electrical bottlenecks at feeder level for residential net zero-energy buildings by integrated system simulation. *Applied Energy*, 96:74–83.
- Clarke, J. 2015. A vision for building performance simulation: a position paper prepared on behalf of the IBPSA Board. *Journal of Building Performance Simulation*, 8(2):39–43.
- Coddington, E. A. and Levinson, N. 1955. *Theory of ordinary differential equations*. McGraw-Hill Book Company, Inc., New York-Toronto-London.
- Constantin, A., Streblow, R., and Müller, D. 2014. The modelica housemodels library: Presentation and evaluation of a room model with the ASHRAE Standard 140. In Association, M., editor, *Proceedings of the 10th International Modelica Conference*, pages 293–299.
- Lauster, M., Teichmann, J., Fuchs, M., Streblow, R., and Mueller, D. 2014. Low order thermal network models for dynamic simulations of buildings on city district scale. *Building and Environment*, 73:223–231.
- Mattsson, S. E. and Elmqvist, H. 1997. Modelica – An international effort to design the next generation modeling language. In Boullart, L., Loccufier, M., and Mattsson, S. E., editors, *7th IFAC Symposium on Computer Aided Control Systems Design*, Gent, Belgium.
- Nytsch-Geusen, C., Huber, J., Ljubijankic, M., and Rädler, J. 2013. Modelica BuildingSystems eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme. *Bauphysik*, 35(1):21–29.
- Polak, E. 1997. *Optimization, Algorithms and Consistent Approximations*, volume 124 of *Applied Mathematical Sciences*. Springer Verlag.
- Wetter, M. 2006. Multizone airflow model in modelica. In Association, M., editor, *Proceedings of the 5th International Modelica Conference*, pages 431–440.
- Wetter, M., Zuo, W., Nouidui, T. S., and Pang, X. 2014. Modelica Buildings library. *Journal of Building Performance Simulation*, 7(4):253–270.