Design of FastBuildings library

Introduction

This document describes the design choices for the Modelica library **FastBuildings**.

Goal of the library

FastBuildings is a library for building energy simulation with the following ambitions:

- low order models for fast simulation, upscaling to districts and optimization
- single and multiple zone building simulation
- modular and easily extensible
- can be compiled by JModelica (also to FMUX) and OpenModelica
- compatible with the **IDEAS** library

Structure

The library has the following typological subpackages:

- Zones - Windows
- Buildings
- HVAC
- Users
- Input
- Examples

Conventions

Imports

The following imports are put on the top level, so usable anywhere in the package:

```
import SI = Modelica.SIunits;
import HT = Modelica.Thermal.HeatTransfer;
```

Naming

Class names can be long and descriptive. They should always start with a capital. For instance names, different rules apply.

> ***Important***
>
> **All instance names start with a small letter.**
>
> **Camel case is applied to all instance names.**
>
> **Wherever possible, 3 letter abbreviations are used.**

One of the only exceptions to these conventions are *T* and *Q_flow* order to maintain compatibility with the *Modelica.Thermal.HeatTransfer* package and avoid confusion.

Many models will be built according to an resistance-capacitor (RC) analogy. It's important to be consistent in the naming of all these components. Therefore, it is adopted to name the components as follows:

- resistance: **resXyz[io][1-9]** (eg. resZon, resWali, resWalo, resFlo1)
- capacitor: **capXyz[1-9]** (eg. capZon, capHea)
- thermal resistance (parameter): **rXyz[io][1-9]**, where Xyz[io][1-9] corresponds to the resistor component (eg. rZon, rWali, ...)
- thermal capacity (parameter): **cXyz[1-9]**, where Xyz[1-9] corresponds to the resistor component (eg. cZon, cHea, ...)

Others:

- windows: **win[1-9]** (eg. win1, win2)
- All temperatures start with **T** (TAmb, TWali ...)
- **rHum, vWin** stand for ambient temperature, relative humidity and wind speed
- **iGloHor**, **irr[1-9]**: global horizontal radiation, irradiation on a specific surface (eg. irr1, irr2)
- design conditions: add *Des* (TAmbDes)
- nominal values: add *Nom* (Q_flowNom)
- all heatPorts are named **heaPor[...]**
- sensors, prescribed temperatures and prescribed heat flows: **senXyz, preXyz**, where Xyz is the variable that is measured (eg. senTZon, preTAmb, preQHeaEmb)

Aliases and propagation

All parameters of *resistances, capacitors* etc. have to be progagated to the level of the *zone*. This allows to switch more easily between different zone models in a building. The naming has to be consistent (see above).

For example, avoid this:

```
model NotNice
  Capacitor capZon(c = 1e6) "Thermal capacity of the zone";
end NotNice;
```

but prefer this:

```
model Nice
  parameter SI.HeatCapacity cZon = 1e6 "Thermal capacity of the zone";
  Capacitor capZon(C = cZon);
end Nice;
```

This may seem inefficient, but it allows for instance to specify a thermal resistance for an exterior wall at zone level, and split this resistance according to a specific rule over the different *resistance* components in the wall itself. Example:

```
model Example_SplittedResistance_Fix
  extends FastBuildings.Zones.BaseClasses.Partials.Partial_SZ;

  FastBuildings.Zones.BaseClasses.Capacitor capZon(C = cZon) ;
  FastBuildings.Zones.BaseClasses.Capacitor capWal(C = cWal) ;
  FastBuildings.Zones.BaseClasses.Resistance resWale(R = 0.5 * rWal) ;
  FastBuildings.Zones.BaseClasses.Resistance resWali(R = 0.5 * rWal) ;

  parameter SI.HeatCapacity cZon "Thermal capacity of the zone" ;
  parameter SI.HeatCapacity cWal "Thermal capacity of the zone" ;
  parameter SI.ThermalResistance rWal "Total thermal resistance of the walls, in K/W" ;

equation
  ...

end Example_SplittedResistance_Fix;


model Example_SplittedResistance_Free
  extends FastBuildings.Zones.BaseClasses.Partials.Partial_SZ_CRE;

  FastBuildings.Zones.BaseClasses.Capacitor capZon(C = cZon) ;
  FastBuildings.Zones.BaseClasses.Capacitor capWal(C = cWal) ;
  FastBuildings.Zones.BaseClasses.Resistance resWale(R = (1-posCapWal) * rWal) ;
  FastBuildings.Zones.BaseClasses.Resistance resWali(R = posCapWal * rWal) ;

  parameter SI.HeatCapacity cZon "Thermal capacity of the zone" ;
  parameter SI.HeatCapacity cWal "Thermal capacity of the zone" ;
  parameter SI.ThermalResistance rWal "Total thermal resistance of the walls, in K/W" ;
  parameter Real posCapWal = 0.5 (min=0, max=1) "Position of the capacity in the wall: 0=inside, 1=outside";

equation
  ...

end Example_SplittedResistance_Free;
```

Issues, questions, doubts

Concept: having all inputs through the sim seems nice and clean (sim = Simulation Input or Info Manager). However, there are some limitations, both from the language and from the tools. An overview of issues that shaped the design:

- not possible to redeclare outer components (Modelica specs?)
- this is not problematic: if the inner is a subtype of the outer it works. So we have to choose the right inner component at the top level of every simulation, and it has to extend from a PartialSim. Every model should have the following declaration: `outer PartialSim sim`.
- In order to make graphical connections from the sim to other components, eg in a zone model, this PartialSim should have the necessary connectors (RealOutputs). This means that all possible inputs have to be defined in the Partial.
- to keep the models lean, I tried to declare those connectors as conditional realoutputs. The extended sim models can specify which connectors are available by specifying the booleans.
- the difficulty is that conditional components can only be referenced in connect equations. So specifying a value for a conditional RealOutput cannot be done in an equation. A RealExpression or similar has to be used, and it's output connected to the conditional RealOutput. This seems inefficient and error-prone.
- moreover, it seems that JModelica has issues with the booleans. The models compile fine, but the CasadiModel cannot be instantiated. The following error message is thrown:

```
RuntimeError                              Traceback (most recent call last)
<ipython-input-84-790aa345e46f> in <module>()
----> 1 c=CasadiModel(fmux)

/home/roel/soft/JModelica_sdk_5464/Python/pyjmi/casadi_interface.py in __init__(self, name, path, verbose)
    104
```

```
    105         # Load CasADi interface
--> 106         self._load_xml_to_casadi(self._tempxml, verbose)
    107
    108         self._ode_conversion = False

/home/roel/soft/JModelica_sdk_5464/Python/pyjmi/casadi_interface.py in _load_xml_to_casadi(self, xml, verbose)
    818         options["sort_equations"] = False
    819         options["eliminate_dependent"] = False
--> 820         self.ocp.parseFMI(xml, options)
    821         casadi.updateDependent(self.ocp)
    822

/home/roel/soft/JModelica_sdk_5464/Python/casadi/casadi.py in parseFMI(self, *args)
  32401
  32402         """
> 32403         return _casadi.SymbolicOCP_parseFMI(self, *args)
  32404
  32405     def addVariable(self, *args):

RuntimeError: SymbolicOCP::readExpr: Unknown node: BooleanLiteral
```

So either add all potential outputs to the Partial_sim?

Maybe with conditional realoutputs? And the subcomponents can use the same booleans to specify inputs? They can be connected and will be removed automatically when the boolean is on false.

Naming: booXXX for booleans?

How to specify the Q_flow of heaPorEmb?

conventions: powEle, qHeaCoo ?

Overleg met Ruben

Array van irr maken voor de instralingen

protected pre componentnes

pow voor elektrisch verbruik

windows onder zones

mapje Buildings.SZ en Buildings.MZ

Input: geen SIM voor elk model

Geen underscores!!

Aparte interfaces package