Software Architecture and Implementation of Modelica Buildings Library Coupling for Spawn of EnergyPlus

Michael Wetter[1] Kyle Benne[2] Baptiste Ravache[1]

[1]Lawrence Berkeley National Laboratory, Berkeley, CA
[2]National Renewable Energy Laboratory, Golden, CO

Abstract

Spawn of EnergyPlus is a next-generation energy simulation engine that targets control design and implementation workflows. Spawn reuses the weather, lighting, loads, and envelope modules from EnergyPlus through a precompiled library and couples them with HVAC and control models implemented in Modelica. Thus, for Spawn, the EnergyPlus HVAC models are removed. Spawn has been designed to perform coupled simulation with any number of EnergyPlus models, supporting simulation of a single building or multiple buildings as part of a district energy system.

This paper describes how the Modelica objects are implemented and synchronized to allow the modular specification at the Modelica-level that uses a Functional Mock-up Unit (FMU) that contains the EnergyPlus model. A key feature of our implementation is that multiple instances of Modelica models call C functions, which jointly build a data structure that defines parameters, inputs and outputs of the EnergyPlus model. This data structure is used during the initialization to generate an FMU that contains a fully configured EnergyPlus model. This FMU is then accessed by all Modelica models to exchange with EnergyPlus values for parameters, inputs and outputs during the simulation. This setup allows the Modelica models to be instantiated in a modular, object-oriented manner, as is typical for Modelica, yet they jointly construct and use an FMU that contains EnergyPlus.

Compared to an HVAC and envelope simulation that uses a native Modelica building model of comparable level of detail, the Modelica-EnergyPlus model translates about 35% faster and simulates about 50% faster.

Keywords: Modelica Buildings Library, Spawn of EnergyPlus, Modelica External Object, FMI

1 Introduction

Modelica has been shown to be well suited to support research, development and design of building and district energy systems, including their control logic (Wetter and Treeck 2017; Wetter, Treeck, et al. 2019). These applications typically require coupled simulations of the energy system and the building envelope. Coupled simulation of building envelopes and energy systems has proven challenging for various reasons. Building envelope models such as the ones in the Modelica Buildings (Wetter, Zuo, Thierry S. Nouidui, et al. 2014), BuildingSystems (Nytsch-Geusen et al. 2013) and IDEAS (Jorissen, Reynders, et al. 2018) libraries add a significant amount of code and a correspondingly large number of continuous-time state variables. These result in long translation times as Modelica tools do not yet satisfactory exploit repeated structures to keep translation time reasonably short. For simulation, the envelope model introduces a large number of continuous time states. These present a problem for the implicit ordinary differential equation solvers that are typically used on these stiff problems as these solvers scale superlinearly in the number of states. At the same time, the use of explicit solvers requires careful model tuning, which is not practical for most users (Jorissen, Wetter, and Helsen 2015). Finally, porting envelope models to Modelica would require considerable resources for porting algorithms including for shading calculations, 3D heat transfer, and coupled heat and moisture transfer through building fabrics, many of which may be better implemented in traditional imperative code. Tools for converting 3D data models for the building envelope would also need to be adapted to support input for Modelica. While future efforts by different building simulation developers may proceed along these lines, more advances are needed in Modelica translators, and multi-rate solvers for systems of stiff ordinary differential equation need to be accessible from Modelica tools in order to make use of such models practical for simulation of large buildings.

The US Department of Energy (DOE) has sponsored the development of EnergyPlus, a whole building energy simulation program (Crawley et al. 2001), since 1996. EnergyPlus is built on fundamental assumptions that makes it poorly suited to modeling building control sequences as they are implemented in physical controllers. DOE has also sponsored the development of the Modelica Buildings Library which is well suited to model HVAC and controls, but suffers from scalability to large building models for the above mentioned reasons. Spawn of EnergyPlus (or just Spawn) is the latest whole-building energy simulation program sponsored by DOE. Developed by the National Labs and industry, Spawn reuses the EnergyPlus envelope model and couples it to Modelica HVAC and control models from the Modelica Buildings Library (Wetter, Benne, et al. 2020), thereby combining the strengths of the two

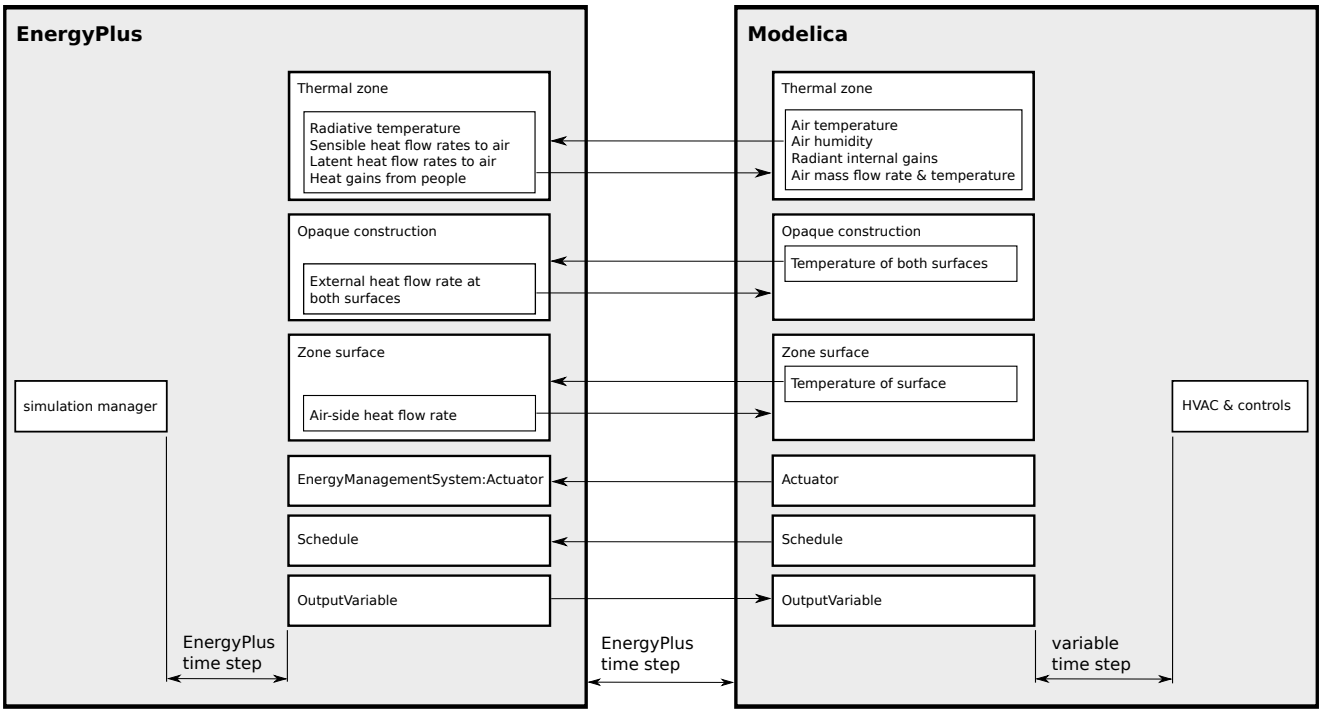


Figure 1. Partitioning of the envelope, room and HVAC model.

software approaches and implementations. Spawn is not an imminent replacement for EnergyPlus. Rather, it is intended to provide several capabilities that significantly advance beyond EnergyPlus and the Modelica Buildings Library. These include modeling of novel HVAC and district energy systems, scalable simulation of large buildings, simulation of control sequences represented in ways that also allow their implementation on building automation systems through a digitized control delivery process, and intrinsic support of multi-physics simulation and co-simulation with third party models.

This paper describes the additions to the Modelica Buildings Library that enables coupling Modelica models to the EnergyPlus envelope model in a way that automatically sets up the coupled simulation. While this implementation is specific for the coupling of building envelope models, a similar mechanism could be used to couple other models for building or district energy systems, e.g., aquifer thermal energy storage in which individual boreholes are connected to the same subsurface model.

The paper is structured as follows: Section 2 describes the variables that need to be exchanged between Modelica and EnergyPlus and states the requirements for coupled simulations. Section 3 describes the implementation. The key contribution is the mechanism that allows a deterministic synchronization of the execution of multiple instances of Modelica models. This synchronized execution is necessary for the software to collect all data required to generate one FMU for the whole building, before any Modelica model requests parameter values or output values from this FMU. Section 4 shows examples of the implementation, and Section 5 provides concluding remarks.

2 Requirements for Modelica Implementation

Figure 1 shows the variables that we require to be exchanged during the simulation between EnergyPlus and Modelica. The coupling variables connect Modelica thermal zone models, which implement the room air heat, mass and pressure balance, with the EnergyPlus thermal zone models that compute the convective heat gains from building fabrics and from internal loads. To support radiant systems, such as a radiant floors, coupling variables connect surface temperatures and heat flow rates between Modelica and EnergyPlus. Coupling variables are also used to read the values of EnergyPlus output variables for use in Modelica-implemented controllers, and to override EnergyPlus schedules and EnergyPlus Energy Management System actuators (Ellis, Torcellini, and Crawley 2007). The latter can be used to send signals to EnergyPlus to control non-HVAC elements such as an active facade, lighting, or other equipment that contributes to heat gains in the room and its surfaces.

To maximize usability and to enable drag-and-drop use in a graphical Modelica editor by non-experts in Modelica, the coupling needs to satisfy the following requirements.

1. To be able to graphically author and inspect models in a graphical modeling environment, each model (thermal zone, zone surface etc. as shown in Figure 1) should be its own instance, rather than being part of an array of models. This also ensures that translation and simulation diagnostics can be readily understood, which would not be the case if models

DOI 10.3384/ecp21181325

were referred to by an index of an array. Furthermore, if arrays of models were used, then wiring the connections would be impractical, in particular for large building models.

2. To enable simulation of multiple buildings, it should be possible to programmatically collect instances of models that belong to the same building in a hierarchical manner. It must also be possible to set common parameters centrally for all models that belong to the same building.
3. The coupled simulation between Modelica and EnergyPlus should be set up automatically for the user.
4. To allow large synchronization time steps, coupling should be done through slowly varying variables.

These requirements are addressed as follows. The first requirement is addressed by having individual Modelica classes (e.g., a **`model`** or **`block`**) for each object that communicates with EnergyPlus.

The second requirement is addressed by using **`inner`** /**`outer`** declarations of a building **`model`** that is used to set common parameters in the Modelica objects that communicate with the EnergyPlus building model. An example of common parameters is the name of the **`outer`** building declaration, which is used to determine which instances of thermal zone models belong to the same building.

The third requirement is addressed by adding a C layer to the Modelica Buildings Library, and a facility to the EnergyPlus program, that exports EnergyPlus as an FMU. This FMU is such that the required inputs and outputs, as specified by the Modelica instances, are exposed through its interface. This C layer invokes a command that generates the FMU, it loads the FMU, and it exchanges data with the FMU.

The second and third requirements leads to the situation that only after the Modelica model is partially initialized, the configuration of the FMU and hence the content of its `modelDescription.xml` file is known. Thus, the FMU needs to be generated during the Modelica initialization, and loaded before Modelica instances read parameter values from the FMU. This situation is a key reason for implementing our custom code for managing the FMU. This code is called using Modelica external C functions that are synchronized through the here explained mechanism.

The fourth requirement is addressed by modeling in Modelica the fast transients of the room air heat, mass and pressure balance, and coupling to EnergyPlus via the slower varying surface temperatures. This partitioning also has the advantage that the room air temperature, humidity and pressure, which are all connected to the HVAC system, are all natively implemented in Modelica. This allows using the same differential equation solver for these variables and the HVAC system.

We selected FMI for Model Exchange, version 2.0, rather than Co-Simulation because we allow certain signals to have direct feed-through. For example we allow setting a window blind and receiving the updated room daylight illuminance level at the same time step. This is only allowed in Model Exchange. Note, however, because EnergyPlus integrates its continuous time states using its own solvers, the FMU exposes no derivative. For Modelica, it looks like a discrete time model.

3 Implementation

3.1 Modelica Classes

For the coupling, we implemented the following Modelica classes:

`Building` Model that declares a building to which EnergyPlus objects belong to.

`ThermalZone` Model to connect to an EnergyPlus thermal zone.

`ZoneSurface` Model to exchange heat with an inside-facing surface of a thermal zone.

`OpaqueConstruction` Model to exchange heat with both surfaces of an opaque construction. The construction is modeled in Modelica. Heat is exchanged with the room-facing front surface and the back-side facing surface of an EnergyPlus construction.

`Actuator` Block to write to an EnergyPlus actuator.

`OutputVariable` Block to read an EnergyPlus output variable.

`Schedule` Block to write to an EnergyPlus schedule.

These Modelica classes allow communication between Modelica and EnergyPlus objects for thermal zones; thermal zone surfaces, either for the inside-facing surface only, or also its back-side facing surface (that may be located in an adjacent zone, or be the outside, or the ground temperature); Energy Management System (EMS) actuators; schedules; and output variables. To associate the Modelica classes to a building, the `Building` model is instantiated using the **`inner`** component prefix, and the other six classes use an instance of the `Building` model with the **`outer`** prefix. Through this mechanism, every instance that is in the instance tree below the `Building` instance will be associated with that particular building, and multiple buildings can be modeled in one Modelica model. All classes, other than `Building`, extend from `ExternalObject` to communicate with code implemented in C. In C, a data structure stores all building instances, and for each building instance, keeps track of which of the above objects belongs to that building instance. This data structure is set up when invoking the constructors of these Modelica instances via the `ExternalObject`. After all constructors are called, an FMU is generated for each building.

3.2 Constructor Synchronization

A key challenge was to enforce that all constructors are called *before* the FMU is generated. The Modelica Lan-

guage Specification 3.5 does not guarantee that all constructors in a model are called before any Modelica function that uses a return value of a constructor is being used. In early research code, some Modelica tools invoked a function that uses the return value from the constructor of the `ExternalObject` before all instances called their constructor. As a consequence, the FMU exposed interface variables for some but not all instances, and the simulation terminated. Therefore, we changed the Modelica implementation to enforce the execution sequence shown in Algorithm 1.

Algorithm 1 Required execution sequence for generation and simulation of envelope model.

Data	Let $\mathscr{I}$ be the set of all instances that communicate with EnergyPlus.
Step 1:	For all instances $i \in \mathscr{I}$, call constructor for i.
Step 2:	For all instances $i \in \mathscr{I}$, initialize i. If first call to any initialization, construct and load FMU, setup experiment.
Step 3:	For all instances $i \in \mathscr{I}$, assign Modelica parameters by getting their values from the FMU.
Step 4:	For all instances $i \in \mathscr{I}$, at each synchronization step, set inputs, time and get outputs from FMU.
Step 5:	For all instances $i \in \mathscr{I}$, call destructor for i. If last call to any destructor terminate and unload the FMU.

A key challenge was to enforce that in Algorithm 1, Step 1 is completed before Step 2 begins. While this would have been easy to enforce by using one constructor for the whole building model, such a centralized specification is impractical. To enforce this calling sequence, we therefore synchronized all objects using a **`connector`** that uses a potential and flow variable, together with **`inner`** and **`outer`** constructs that hide this complexity from the user. Note that these **`inner`** and **`outer`** constructs are different from the ones described in Section 3.1. Our implementation is based on the code provided by Beutlich (2021), which was motivated by Elmqvist et al. (2015).

Listing 1 to 9 describe this implementation, using a minimum representative example that has only one building and two thermal zones. The actual implementation is considerably larger and can be found in the Modelica Buildings Library 8.0.0, package `Buildings.ThermalZones.EnergyPlus`. Listing 1 shows the package with the `SynchronizeConnector` whose **`flow`** variable will be assigned by every thermal zone. The `SynchronizeConnector` is instantiated at the building level, as shown in Listing 2. The building sets its potential variable, which is needed for the model to be well defined, and it declares a variable `isSynchronized` whose value is set to the flow variable of the connector. Listing 3 shows the implementation of the thermal zone which extends `ObjectSynchronizer` and through this **`extends`** statement, gets a reference to the **`outer`** `building` and an instance of synchronization connector `synBui`. The call to `initialize` takes as an argument `building.isSynchronized`, which is computed by the **`outer`** `building` instance, and this computation requires the return value `nZ` of `initialize` which is assigned to `building.synchronize.done` via the `ObjectSynchronizer`. The other code in `ThermalZone` is a standard use of an external function interface that returns `adapter` which encapsulates a pointer to the C structure that contains the data structure needed to orchestrate the FMU coupling. This external function interface is shown in Listing 4. The two Modelica functions that communicate with the C implementation are shown in Listings 5 and 6, and the C implementation is shown in Listings 7 and 8.

Listing 1. Package that synchronizes all objects that belong to the building.

```
within BuildingRooms;
package Synchronize
  connector SynchronizeConnector
    Real do "Potential variable";
    flow Real done "Flow variable";
  end SynchronizeConnector;

  model SynchronizeBuilding
    SynchronizeConnector synchronize;
  end SynchronizeBuilding;

  model ObjectSynchronizer
    outer Building building;
    SynchronizeBuilding synBui;
  equation
  connect(building.synchronize,
      synBui.synchronize);
  end ObjectSynchronizer;
end Synchronize;
```

Listing 2. Model that declares building-level parameters.

```
within BuildingRooms;
model Building
  "Model that declares a building"
  Synchronize.SynchronizeConnector
      synchronize;
  Real synchronization_done =
      synchronize.done;
  Real isSynchronized;
equation
  synchronize.do = 0;
algorithm
  isSynchronized := synchronization_done;
end Building;
```

Listing 3. Model that implements the thermal zone.

```
within BuildingRooms;
model ThermalZone
```


DOI 10.3384/ecp21181325

```
  extends Synchronize.ObjectSynchronizer;
  constant String name=getInstanceName();
  ZoneClass adapter = ZoneClass(name,
      startTime);

  parameter Real startTime(fixed=false);
  parameter Integer nZ(
    fixed=false, start=0)
    "Total number of zones in building";
  constant Real k=1;
  Real tNext(start=startTime, fixed=true);
  Real T(start=293.15, fixed=true);
  Real Q_flow;

initial equation
  startTime=time;
  nZ=initialize(
    adapter=adapter,
    startTime=time,
    isSynchronized=building.isSynchronized)
        ;
equation
  when {initial(), time >= pre(tNext)} then
    (tNext, Q_flow) =exchange(
      adapter,
      time,
      T,
      nZ);
  end when;
  k*der(T) = Q_flow;
  nZ =synBui.synchronize.done;
end ThermalZone;
```

Listing 4. Model that implements the thermal zone.

```
within BuildingRooms;
class ZoneClass extends ExternalObject;

  function constructor
    input String name "Name of the zone";
    input Modelica.SIunits.Time startTime;
    output ZoneClass adapter;
  external "C" adapter=ZoneAllocate(name)
    annotation (
      Include="#include <thermalZone.c>",
      IncludeDirectory="modelica://
          BuildingRooms/Resources/C-Sources
          ");
  end constructor;

  function destructor
    input ZoneClass adapter;
  external "C" ZoneFree(adapter)
    annotation (
      Include="#include <thermalZone.c>",
      IncludeDirectory="modelica://
          BuildingRooms/Resources/C-Sources
          ");
  end destructor;
end ZoneClass;
```

Listing 5. Model that implements the thermal zone.

```
within BuildingRooms;
function initialize
  input ZoneClass adapter;
  input Real startTime;
  input Real isSynchronized;
  output Integer nZ "Number of zones";
  external "C" ZoneInitialize(adapter,
      startTime, nZ)
  annotation (
    Include="#include <thermalZone.c>",
    IncludeDirectory="modelica://
        BuildingRooms/Resources/C-Sources")
        ;
end initialize;
```

Listing 6. Model that implements the thermal zone.

```
within BuildingRooms;
function exchange
  input ZoneClass adapter;
  input Real t;
  input Real T;
  input Integer nZ;
  output Real tNext;
  output Real Q_flow;
  external "C" ZoneExchange(adapter, t, T,
      tNext, Q_flow)
  annotation (Include="#include <
      thermalZone.c>",
              IncludeDirectory="modelica://
                  BuildingRooms/Resources/
                  C-Sources");
end exchange;
```

Listing 7. Header file for C code that is a mock-up for the code that instantiates and communicates the FMU for the building envelope.

```
#ifndef thermalZone_h
#define thermalZone_h

typedef struct Zone{
  char* name;
} Zone;

#endif
```

Listing 8. C code that is a mock-up for the code that instantiates and communicates the FMU for the building envelope.

```
#ifndef thermalZone_c
#define thermalZone_c

#include <string.h>
#include <stdbool.h>

#include "thermalZone.h"

static int nZon = 0; /* Number of zones  */
static bool buildingIsInstantiated = false;

void* ZoneAllocate(const char* name){
  Zone* ptrZone;

   /* Allocate zone and assign name */
  ptrZone = (Zone*) malloc(sizeof(Zone));
  ptrZone->name =
    malloc((strlen(name)+1) * sizeof(char))
        ;
  strcpy(ptrZone->name, name);
  /* Increment counter for zones */
```

```
  nZon++;

  ModelicaFormatMessage(
    "Allocated zone %s\n", name);
  return (void*) ptrZone;
}

void ZoneInitialize(
    void* object,
    double startTime,
    int* nZ){
    Zone* zone = (Zone*) object;
    *nZ = nZon;
    if (!buildingIsInstantiated){
      /* Here, the actual implementation
         constructs an FMU that
         is shared by all zones.
         This requires that all zones
         executed ZoneAllocate().
      */
        buildingIsInstantiated = true;
        ModelicaFormatMessage(
          "Initialized zone %s.
              Instantiated building, nZ = %
              d.\n",
          zone->name, nZon);
    }
    else{
        ModelicaFormatMessage(
          "Initialized zone %s, nZon = %d\n
              ",
          zone->name, nZon);
    }
}

void ZoneExchange(
    void* object,
    double time,
    double T,
    double* tNext,
    double* Q_flow){
    Zone* zone = (Zone*) object;
    /* In the actual implementation,
       this is computed in an FMU.
    */
    *Q_flow = 283.15-T;
    *tNext = time + 1;
    ModelicaFormatMessage(
      "Exchanged with zone %s at time=%f,
          nZon = %d\n",
      zone->name, time, nZon);
}

void ZoneFree(void* object){
    Zone* zone = (Zone*) object;
    free(zone->name);
    free(zone);
}

#endif
```

For the user, the complexity of the synchronization is hidden. A building and its elements can be configured using the same Modelica constructs as are used for other instances, as Listing 9 shows.

Listing 9. Model that instantiates a building and two thermal zones that belong to this building.

```
within BuildingRooms;
model MyBuildingInstance
  "Building with two thermal zones, e.g.,
     nZ=2"
  inner Building building;
  ThermalZone t1;
  ThermalZone t2;
end MyBuildingInstance;
```

Simulating this model will give an output such as

```
Allocated zone MyBuildingInstance.t2
Allocated zone MyBuildingInstance.t1
Initialized zone MyBuildingInstance.t1.
    Instantiated building, nZ = 2.
Initialized zone MyBuildingInstance.t2,
    nZon = 2
Initialized zone MyBuildingInstance.t1,
    nZon = 2
Initialized zone MyBuildingInstance.t2,
    nZon = 2
Exchanged with zone MyBuildingInstance.t1
    at time=0.000000, nZon = 2
Exchanged with zone MyBuildingInstance.t2
    at time=0.000000, nZon = 2
...
```

3.3 C API

To control the FMU that contains the EnergyPlus envelope model, we developed a library in C which uses the FMI Library (*FMI Library* 2021) to interact with the FMU. Figure 2 shows the UML sequence diagram. Each Modelica object that communicates with EnergyPlus extends from the Modelica built-in class `ExternalObject`. Through its constructor, the Modelica instance calls the C code which registers the object in a `static struct`, and stores parameters that are declared in Modelica. These parameters include for example the name of a thermal zone so that it can be matched to the thermal zone object in the EnergyPlus model. Through the name of the **`outer`** instance of `Buildings`, objects that belong to the same building are registered accordingly. After all constructors are called, the first call to `initialize` will invoke a program that generates the FMU. Next, through the Modelica function `getParameters`, parameters such as the volumes of a thermal zone that are computed by EnergyPlus are retrieved from the FMU and assigned to Modelica parameters. During the simulation, the Modelica `exchange` function exchanges data and synchronizes time with the FMU. Finally, the destructor of the Modelica `ExternalObject` terminates and unloads the FMU.

3.4 FMU Generation

During the `initialize` step, an executable program `spawn` is invoked to generate a unique FMU for each `Building` configuration. `spawn` is invoked via a command line interface, which accepts a JSON file that specifies the contents of the resulting FMU. All configuration is specified by the Modelica classes described in Section 3.1.

DOI 10.3384/ecp21181325

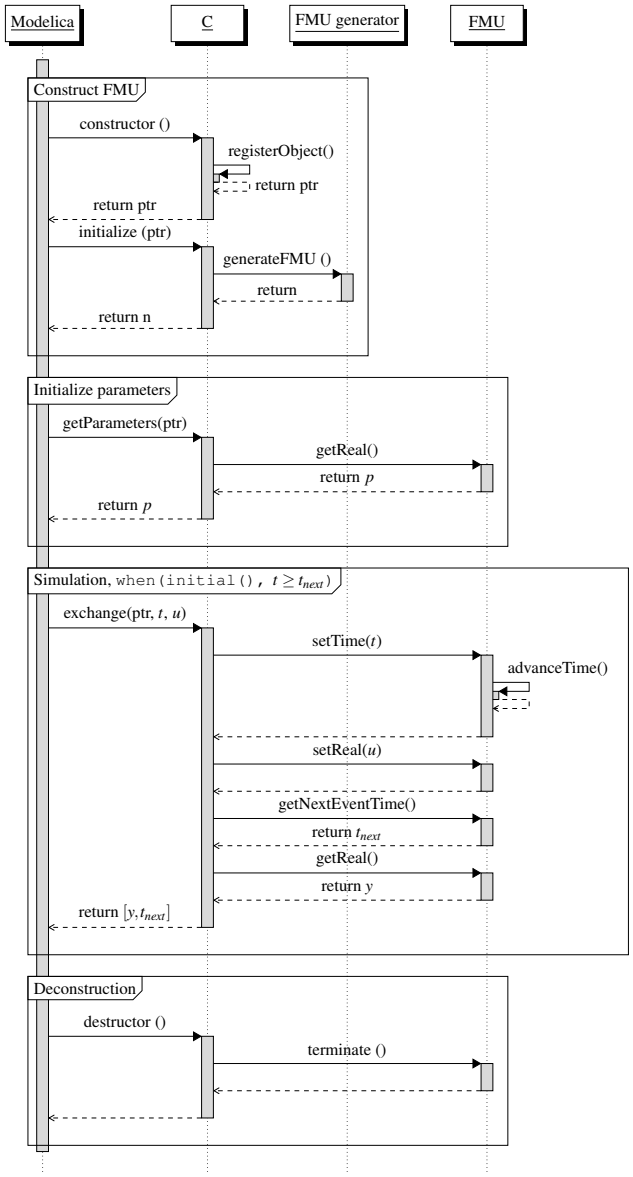


Figure 2. Sequence diagram for interaction with FMU.

The following steps are taken by the `spawn` program during FMU generation.

1. Create a staging directory.
2. Copy required resources into the staging directory, including the EnergyPlus input data file (IDF), and weather files. These files are specified by the user via parameters of the Modelica `Building` model.
3. Modify the given EnergyPlus IDF file so that it conforms to Spawn's requirements. The primary modification is to remove any EnergyPlus HVAC and control related objects.
4. Copy a custom EnergyPlus based shared library into the staging directory.
5. Generate a `modelDescription.xml` file, according to the variables that are requested via JSON input.
6. Compress the staging directory into zip format.

The resulting FMU is a self contained package with all of the resources required for an EnergyPlus based building simulation. Although it is possible to interact with the command line tool directly, it is currently not supported as a stand-alone tool.

3.5 Changes to EnergyPlus

The coupling of EnergyPlus with Modelica introduces unique requirements that EnergyPlus did not originally address. First, EnergyPlus was its own simulation manager and controlled the progression of simulated time; in Spawn, time is managed by Modelica. Second, although EnergyPlus included an External Interface feature for runtime data exchange, the existing capability was insufficient for Spawn. Most importantly, the External Interface feature limited the communication step to that of the zone time step, a limitation that derived from EnergyPlus' internal HVAC and control system models. In Spawn, HVAC and control system models are simulated using Modelica.

We modified EnergyPlus to allow the EnergyPlus HVAC loop to be bypassed, leaving the core EnergyPlus zone heat balance calculation engine which, based on our modifications, can be invoked at any simulated time even below the traditional zone time step limit of one minute.

We also added a software layer on top of EnergyPlus that facilitates simulation in which EnergyPlus is advanced through time by another program, and data is exchanged at each step. The former External Interface approach to co-simulation was based on a client-server architecture operating over a TCP/IP socket. However, socket based communication involves serialization and de-serialization at the endpoints that introduces a performance penalty with every exchange. The new approach implemented for Spawn is based on a co-routine design pattern. The co-routine is implemented using two threads. One thread contains the conventional EnergyPlus routine, and a second thread is a control thread that implements functions such as `setTime`, `setReal`, and `getReal`. In the co-routine, only one thread is active at any moment in time, and the two threads share memory, making data exchange between them efficient. The co-routine works by ping-pong'ing between the two threads. The EnergyPlus thread is blocked until a signal from the control thread is sent to advance in time; in turn the control thread is blocked until EnergyPlus signals that it has advanced to the desired time. When the EnergyPlus thread is blocked, the control thread can access EnergyPlus state and respond to requests for data. This results in an efficient data exchange with EnergyPlus and limited modifications to EnergyPlus code. This software layer combined with EnergyPlus is compiled as a shared library and included in the generated FMU described in Section 3.4.

4 Examples

We will now show two examples. The first example shows how translation and simulation time compares between a scalable model that uses an identical Modelica HVAC and control model with the envelope model of either the Modelica Buildings Library (Wetter, Zuo, and Thierry Stephane Nouidui 2011; Thierry Stephane Nouidui et al. 2012) or of EnergyPlus. The second example shows how to configure a Modelica model that uses the EnergyPlus envelope model to control a shade.

4.1 Translation and Simulation Time

This example shows how translation and simulation time changes between a native Modelica implementation and the EnergyPlus-Modelica coupled implementation for a building model with detailed HVAC system of varying size. For this example, we created a scalable model of the Modelica Buildings Library's `ThermalZones.Detailed.MixedAir` thermal zone model and the EnergyPlus envelope model `ThermalZones.EnergyPlus.ThermalZone`. Both cases model multiple floors that are representative of the large office building from the commercial reference building models for Chicago, IL (Deru et al. 2011). Each floor has 4 perimeter zones and a large core zone. Each floor is served by its own VAV system that includes an economizer, heating and cooling water-to-air coils and terminal reheat boxes. The system controls the ventilation, heating and cooling of all five zones based on ASHRAE Guideline 36 (ASHRAE 2018). Both cases use the same HVAC model. The hot- and cold-water loops are modeled with idealized heat sources and sinks.

The template models are scaled in size by varying the number of floors as shown in Table 1. As each floor is served by one HVAC system and has 5 thermal zones, the case with 10 floors has, for example, 10 HVAC systems and 50 thermal zones.

To have different state trajectories for each floor, each floor was configured to have a slightly different design air flow rate. This measure ensures that each floor triggers state events that are not simultaneous to state events from other floors, and that the adaptive time step solver computes indeed different error estimates for each floor, which overall may lead to more time steps as the number of diverse floors increases. Without this measure, the scaling may have been non-representative as temperatures in different floors typically evolve on different trajectories.

The models are available from `https://github.com/lbl-srg/modelica-buildings`, commit 15b90ae8bd5c4f3d6de23eee66b2efaab0c78b60.[1] The translated model with 10 thermal zones has 1700 continuous states and 48800 time varying variables if the `MixedAir` model is used, and 810 continuous states (about half of the native Modelica implementation) and 36800 time varying variables if the `EnergyPlus` model is used. All models were simulated for the days indicated in Table 1, using the Chicago TMY3 weather file. We used Dymola 2021 on Ubuntu 18.04 with the CVode solver, a tolerance of 10^{-5} and the sparse solver unless indicated otherwise in the table.

Table 1 show the translation and simulation times. The simulation time corresponds to the total CPU time required to simulate the compiled model. [2] Figure 3 shows the CPU time as a function of model time, and the relative computing time for each day. As can be seen in the figure, the slope of the CPU time is not constant over the model time. These change in slope are attributed to the change in dynamics of the state trajectories that occurs during certain parts of the model time. As shown in the plot, there are no step changes in the CPU time. A step change would have indicated a numerical problem, which may distort the total computing time as the numerical error is not an artifact of the different envelope model but rather of the resulting differential algebraic system of equations.

[1] Modelica package Examples.ScalableBenchmarks.ZoneScaling.

[2] This version of Spawn computes the numerically expensive shadow calculations from January 1 to the start day of the simulation. For the cases where the simulation starts in summer, this time is substantial. Because this is planned to be corrected in future releases, we subtracted this time in all reported results.

DOI 10.3384/ecp21181325

Table 1. Translation and simulation time for the `MixedAir` and EnergyPlus thermal zone model.

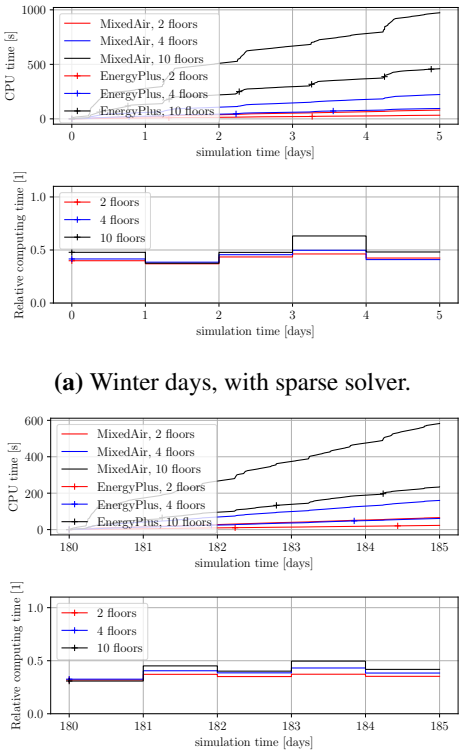
Model	Floors	Translation Time [s]	Simulation Time [s]
MixedAir days 1-5	2	47	81
	4	89	223
	10	230	973
EnergyPlus days 1-5	2	31 (66%)	35 (43%)
	4	57 (64%)	97 (43%)
	10	139 (61%)	462 (48%)
MixedAir days 1-5, non-sparse	2		102
	4		361
	10		2570
EnergyPlus days 1-5, non-sparse	2		39 (38%)
	4		115 (32%)
	10		677 (26%)
MixedAir days 180-185	2		66
	4		160
	10		583
EnergyPlus days 180-185	2		30 (45%)
	4		74 (46%)
	10		264 (45%)

In summary, the models with the EnergyPlus thermal zones translate about 35% faster. Their simulation time is also about 50% faster for the cases with the sparse solver. For the model with 10 zones, disabling the sparse solver increases the computing time by a factor of 2.5 for the case with the `MixedAir` model, and by about 1.5 for the case with the `EnergyPlus` model.

4.2 Shade Control

This example illustrates how to interface with EnergyPlus from different Modelica models. Figure 4 shows a model of a building with three thermal zones, one of which has a window with a shade. These are simulated in EnergyPlus. Modelica models the window shade control sequence, for which it obtains incident solar radiation from EnergyPlus and sends the actuation signal back to EnergyPlus. Modelica also models the fresh air supply and an idealized cooling system in each thermal zone. The air heat and mass balances for each room are modeled in Modelica, and the envelope heat transfer is modeled in EnergyPlus.

In the figure, the instance `building` specifies building-level settings, such as the EnergyPlus IDF file. The three blue icons in the middle connect to three EnergyPlus thermal zones. The instance `incBeaSou` reads from EnergyPlus the incident beam solar radiation on the window, and the instance `actSha` actuates the window shade. In the EnergyPlus model, the west-facing thermal zone has a window blind that is open if its control signal is 0 or closed if it is 6. The control sequence obtains the room air temperature of the west-facing zone from the Modelica instance `zonWes`, and connects it to a hysteresis block that switches its output to `true` if the zone temperature is above 24°C, and to false if it drops below 23°C. The instance `incBeaSou` obtains from EnergyPlus the incident solar beam radiation on the outside of the window, and feeds it into a hysteresis block that outputs `true` if its in-



(a) Winter days, with sparse solver.

(b) Summer days, with sparse solver.

Figure 3. CPU time and relative computing time for Modelica Buildings Library MixedAir and EnergyPlus thermal zone model. The relative computing time is the ratio of CPU time it took to simulate the indicated day for Spawn compared to the native Modelica model.

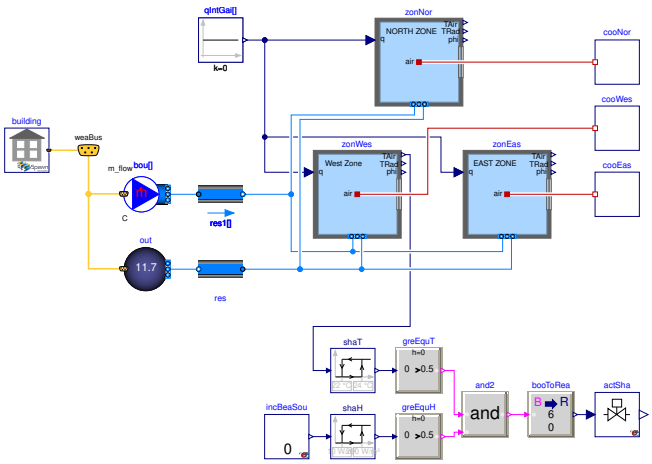


Figure 4. Schematic diagram of the Spawn model with shade control, available from the Buildings Library as `Buildings.ThermalZones.EnergyPlus.Examples.SingleFamilyHouse.ShadeControl`. Note that the thermal zone models `zon*`, the output variable reader for the incident solar radiation `incBeaSou` and the actuator for the shade `actSha` all communicate with the same EnergyPlus model via C functions. Thus, the control loop from shade control to zone temperature `zonWes.TAir` is closed via EnergyPlus.

put exceeds $200\,\mathrm{W/m^2}$, and switches to `false` if it drops below $10\,\mathrm{W/m^2}$. The instance `actSha` connects to the actuator in EnergyPlus that activates this shade. If both outputs of the hysteresis blocks are `true`, then the EnergyPlus shade actuator is deployed by setting the input of `actSha` to 6. Otherwise, the input is set to 0. To the right of the model, there are three idealized cooling systems that keep the room air temperature below 25°C in each of the three zones. Also, each zone is connected to a constant, unconditioned outside air supply.

5 Conclusions

Through the use of **`inner`**/**`outer`** constructs and a **`flow`** variable, we were able to ensure a correct synchronization of Modelica models that communicate with a common data structure via C functions that each use a distinct pointer to memory obtained through a Modelica `ExternalObject`. This was essential for enabling model authoring in the same way as one typically does with Modelica models that are instantiated in a distributed manner within a larger Modelica system model. The implementation ensures that each constructor is called before the first Modelica instance calls its initialization function that generates and imports the FMU, which is then accessed for input and output by the different Modelica instances. The resulting implementation allows simulating one or several buildings, where each building is represented by an FMU that can have any number of objects that are synchronized with Modelica.

For a Modelica model that consists of a variable air volume flow system and detailed control sequence, coupled to a multi-zone building envelope model that is implemented either in Modelica or in EnergyPlus, the version that uses EnergyPlus translates about 35% faster and simulates about 50% faster.

Acknowledgements

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.

The authors would also like to thank Thomas Beutlich for pointing out and adapting a similar synchronization problem from Elmqvist et al. (2015).

References

ASHRAE (2018-06). *ASHRAE Guideline 36-2018 – High Performance Sequences of Operation for HVAC systems*.

Beutlich, Thomas (2021). *Modelica Specification issue 2842*. URL: https://github.com/modelica/ModelicaSpecification/issues/2842#issuecomment-776194950 (visited on 2021-03-08).

Crawley, Drury B. et al. (2001). "EnergyPlus: creating a new-generation building energy simulation program". In: *Energy and Buildings* 33.4. Special Issue: BUILDING SIMULATION'99, pp. 319–331. ISSN: 0378-7788. DOI: https://doi.org/10.1016/S0378-7788(00)00114-6.

Deru, Michael et al. (2011-02). *U.S. Department of Energy Commercial Reference Building Models of the National Building Stock*. Tech. rep. National Renewable Energy Laboratory.

Ellis, Peter G., Paul A. Torcellini, and Drury B. Crawley (2007). "Simulation of Energy Management Systems in EnergyPlus". In: *Proc. of the 10-th IBPSA Conference*. Ed. by Jiang Yi et al. International Building Performance Simulation Association and Tsinghua University. URL: http://www.ibpsa.org/.

Elmqvist, Hilding et al. (2015-09). "Generic Modelica Framework for MultiBody Contacts and Discrete Element Method". In: *11-th International Modelica Conference*. Ed. by Peter Fritzson and Hilding Elmqvist. Modelica Association. Paris, France, pp. 427–440. DOI: 10.3384/ecp15118427.

Jorissen, Filip, Glenn Reynders, et al. (2018). "Implementation and Verification of the IDEAS Building Energy Simulation Library". In: *Journal of Building Performance Simulation* 11 (6), pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

Jorissen, Filip, Michael Wetter, and Lieve Helsen (2015-09). "Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation". In: *11-th International Modelica Conference*. Ed. by Peter Fritzson and Hilding Elmqvist. Modelica Association. Paris, France, pp. 59–69. DOI: 10.3384/ecp1511859.

FMI Library (2021). URL: https://github.com/modelon-community/fmi-library (visited on 2021-03-08).

Nouidui, Thierry Stephane et al. (2012-09). "Validation and Application of the Room Model of the Modelica Buildings Library". In: *Proc. of the 9-th International Modelica Conference*. Modelica Association. Munich, Germany, pp. 727–736. DOI: 10.3384/ecp12076727.

Nytsch-Geusen, Christoph et al. (2013). "Modelica BuildingSystems eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme". In: *Bauphysik* 35.1, pp. 21–29. ISSN: 1437-0980. DOI: 10.1002/bapi.201310045.

Wetter, Michael, Kyle Benne, et al. (2020-09). "Lifting the Garage Door on Spawn, an Open-Source BEM-Controls Engine". In: *Proc. of Building Performance Modeling Conference and SimBuild*. Chicago, IL, USA, pp. 518–525. URL: https://simulationresearch.lbl.gov/wetter/download/2020-simBuild-spawn.pdf.

Wetter, Michael and Christoph van Treeck (2017-09). *IEA EBC Annex 60: New Generation Computing Tools for Building and Community Energy Systems*. ISBN: 978-0-692-89748-5. URL: http://www.iea-annex60.org/pubs.html.

Wetter, Michael, Christoph van Treeck, et al. (2019-09). "IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges". In: *IOP Conference Series: Earth and Environmental Science* 323, p. 012114. DOI: 10.1088/1755-1315/323/1/012114.

Wetter, Michael, Wangda Zuo, Thierry S. Nouidui, et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Wetter, Michael, Wangda Zuo, and Thierry Stephane Nouidui (2011-11). "Modeling of heat transfer in rooms in the Modelica "Buildings" library". In: *Proc. of the 12-th IBPSA Conference*. International Building Performance Simulation Association. Sydney, Australia, pp. 1096–1103. URL: https://simulationresearch.lbl.gov/wetter/download/2011-ibpsa-BuildingsLib.pdf.