OpenBPS: A New Building Performance Simulation Tool

Livio Mazzarella – Politecnico di Milano – livio.mazzarella@polimi.it
Martina Pasini – Politecnico di Milano – martina.pasini@polimi.it

Abstract

A new generation building energy performance simulation program, OpenBPS™, is currently under development. It overcomes some of the drawbacks typical of many of the popular building energy simulation programs around the world. This Building Performance Simulation Tool is primarily a set of libraries dedicated to building energy analysis and performance simulation, which can be included in any user-oriented interface or commercial software that aims to perform such analysis. The basic goal of the project is to provide a robust, validated, and high-performing calculation engine that can be shared, and grow with the contribution of a community of developers and users. To maximize its possible deployment and to facilitate its development and extension by a growing community, it has been built as an open source cross-platform (Windows, Mac, and Linux) software library. For this reason, OpenBPS™ will be distributed under a Copyleft Software License (EUPL) and is coded with a cross-platform object oriented programming language, C#, which is an open source language for .NET Framework based on ECMA standards. The main features of this tool are the object-oriented modelling of physical phenomena and building and HVAC system components, the native code parallelization to take advantage of multi-thread/multi-core processors today available, the multi-scale calculation time step (each object can work using its own time step, scaling down or up with respect to the chosen simulation time step), etc. Not only the technical systems are described and simulated modularly, being their components objects, but also the building fabric is natively modular. Any building envelope component is an object that interacts with other objects, which represent the world around it (air node included). This allows the use of different modelling approaches for different wall components during the same simulation (linear, non-linear, with phase-change, ventilated, etc.). The input and output data structures are tailored to facilitate third party integration with high efficiency, using today's technologies. Other planned capabilities include multi-zone airflow simulation and dynamic models for HVAC system components.

1. Introduction

Over the last years at the Technical University of Milan (Politecnico di Milano), a research group has been working on developing a "next generation" building energy performance simulation tool. In 2009 a preliminary research project, carried out through a Ph.D. work (Pasini, 2009), traced the route to the re-conceptualisation and development of an object-oriented model for the simulation of the building system (Mazzarella and Pasini, 2009). The main idea was to combine the power of the object-oriented programming languages today available with the object-oriented nature of a building. In developing the tool, great care has been taken in its design to ensure modularity and maintainability through an open source (OS) development approach. The development methodology was itself part of the development work, aimed at creating a common framework for a community of developers able to manage the complete software development lifecycle (Mazzarella and Pasini, 2015a). Code validation was then a second key point along the development of the tool: both analytical and comparative tests have been employed to assess the quality of the implemented algorithms. The first results of a comparative validation done on such tool, following the BESTEST standard, have been presented at the 2013 IBPSA International Conference (Mazzarella and Pasini, 2013). Some other comparative tests between different numerical solutions of the heat conduction differential equation have been presented at the 6th International Building Physics Conference, IBPC 2015 (Mazzarella and Pasini, 2015b). Most of the

Part of
Pernigotto, G., Patuzzi, F., Prada, A., Corrado, V., & Gasparella, A. (Eds.). 2018. Building simulation applications BSA 2017. bu,press.
https://doi.org/10.13124/9788860461360

work has since been documented in technical reports, which will be made available with the tool in the future. Since part of the developing work was financed by Regione Lombardia, the final decision, about how to deploy and support it, is still pending. The name of the tool has already been chosen and is OpenBPS (Open Building Performance Simulator). This paper describes the structure, features, and capabilities of OpenBPS.

2. What is OpenBPS

OpenBPS is a new building performance simulation program primarily designed as an open source cross-platform (Windows, Mac, and Linux) software library. It is a simulation engine and there is no formal user interface. For developing and testing purposes, a simple GUI is provided, which is able to import building geometry provided by the OpenStudio plug-in for SketchUp or to directly import projects defined through EnergyPlus input files (.idf), although not yet fully implemented. It is coded from scratch with a cross-platform object oriented programming language, C#, which is an open source language for .NET Framework based on the ECMA standards. This language has easily allowed native code parallelization to take advantage of multi-thread/multi-core processors today available.

2.1 Object Oriented Code

One of the main goals for OpenBPS is to create an enhanced modular structure that facilitates adding new features and allows the library to be used by any hosting program. An object-oriented programming language, as C#, was selected to achieve this goal because it:

- is a rich implementation of the object-oriented paradigm, which includes encapsulation, inheritance, polymorphism, and method overriding;
- is an easy and efficient object oriented language: developers can translate their ideas/algorithms to solve complex problems more easily than with C++;
- is one of the leading languages which works on cross-platform using .NET framework: it is available in Windows, Linux and MacOS operating systems;
- is more type safe than C++; the only implicit conversions by default are those that are considered safe, such as widening of integers; the managed memory cannot be explicitly freed; it is instead automatically garbage collected;
- may produce applications that run as fast as C++ applications, using the Just In Time (JIT) compiler, which can finely tune code optimization on the running machine hardware;
- is today a standard and open-source programming language (ECMA-334 and ISO/IEC 23270:2006).

An object-oriented structure is natively modular and simplifies the reuse of pieces of code through the concept of class inheritance. Different kinds of objects often have a certain amount of “attributes” and “behaviours” in common with each other. Phase-change walls, breathing walls and ventilated walls, for example, all share the characteristics of walls (geometry, layers composition, orientation, etc.). Yet each also defines additional features that make them different: phase-change walls have phase-change material properties to account for together with nonlinear performance; breathing walls have additional properties like porosity and their solver has to account for advection through the wall porous material; ventilated walls have additional channel properties which account for inner wall mass forced/natural ventilation. Object-oriented programming allows classes to inherit commonly used states and behaviours from other classes. In this example, Wall is the *superclass* of the classes “phase-change walls”, “breathing walls”, and “ventilated walls”.

Objects are class instances. For instance, the class “Building” describes the concept of what a building is. When the class “Building” is specified, by loading specific data, the object, “Building-XYZ” is built, representing a particular instance of the class Building.

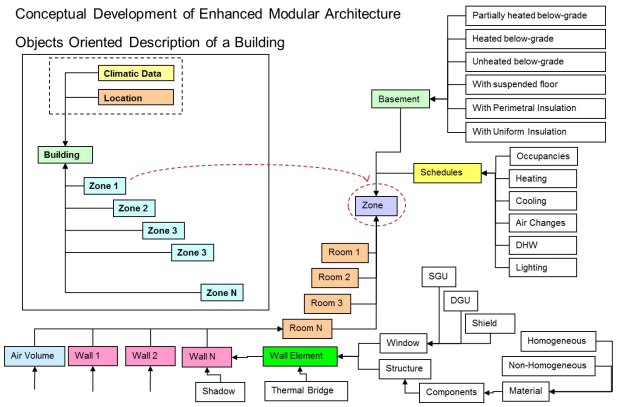

Fig. 1 – Enhanced modular structure of OpenBPS based on the building's object-oriented descriptions

The object-oriented structure helps parallel calculations at the object's management level. Since each wall in a building is a separate object, at each time step its performance can be calculated in parallel to other walls (how many depends on how many CPUs are available) (Mazzarella et al., 2014). Also, the object-oriented approach naturally allows the use of different integration time steps inside each object and lets them optimize their performances regardless of the global simulation time step imposed by the user.

2.2 OpenBPS Structure

The enhanced modular structure of OpenBPS, shown in Fig. 1, follows a hierarchical organization, with the "Building" class at the top, hierarchically encapsulating all the instances of the other classes that contribute to its definition. To implement such structure in a manageable code, OpenBPS has been developed inside MS Visual Studio development framework, and is organized in a "Solution" that contains all the necessary elements to build its exe code, debug, and test, or continue its development. This Solution comprises several "projects" that have been created to manage different aspects of software development, according to the modular nature of the whole project. As shown in Fig. 2, there are seven projects in the solution, each of them with a specific functionality:

- *GUI*, the executable project containing a simple user interface provided for developing and testing purposes only;
- *ExtendedMath*, a dll (dynamic link library) project containing the math algorithms used for simulation;
- *SimulationManager*, a dll project containing the simulation manager implementation;
- *SimulationComponents*, a dll project containing the description of the building system components;
- *InputOutputUtilities*, a dll project containing the tools for the management of inputs and outputs;
- *ModelingProject1*, an executable project that supports the development stage by checking for interdependence among code components, and assessing if they comply or not with predefined dependency rules;
- *TestProject1*, an executable project that implements a set of automatized tests with known solutions to verify code integrity after code modifications.

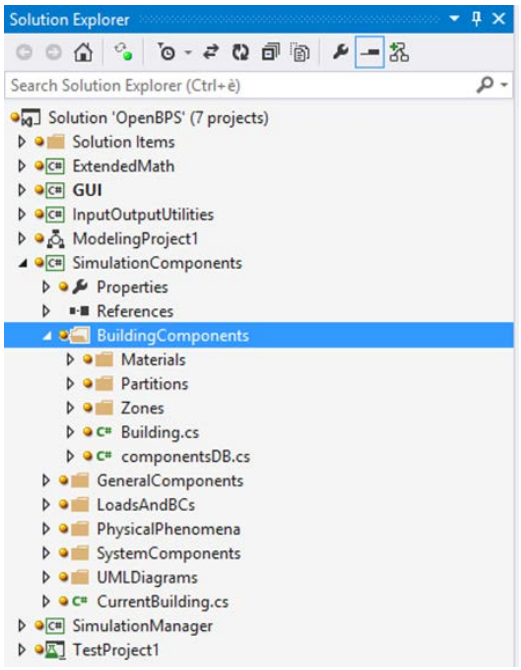

Fig. 2 – OpenBPS Solution modular structure

2.3 Simulation Management

Among the four main libraries, which constitute the simulation engine of OpenBPS, the *SimulationManager* dll project, contains the Simulator code. This is the top-level object that manages the simulation, controlling the interactions among all the objects constituting the building. Due

to the object-oriented nature of the whole code, the components management is handled via iterations until convergence is reached inside each global simulation time step. This approach allows separating the global simulation time step (the time step the Simulator uses to update its information) from each local time step any single object may use. In this way, it is also possible to implement in each object strategies to prevent the user to force such component to work with an inconsistent time step (in respect to the characteristic time of that component). Basically, the Simulator organizes and synchronises the work of all objects, instructing them to take actions such as initialize, simulate, export results, etc. It can manage from sub-hourly up to several-hour time steps over a simulation period ranging from a day to several years.

The simulation manager that concentrates in one object all management rules, allows:

- to better define the priorities in objects execution;
- to allocate the parallelization to a specific group of objects, which can benefit more than others of such technique;
- to consistently control the work flow;
- to easily add new classes and objects.

In Fig. 3 a simplified scheme shows how the parallel calculations are applied to the building: a first parallelization is applied among zones, given the priority of some calculations at their level, and a second parallelization is applied among all the other simulation components involved in the calculation.

2.4 Energy and Mass Balance

The underlying building thermal zone calculation method in OpenBPS is an integral enthalpy balance model in which room air is modelled under the fully mixed assumption, i.e. with the assumption of uniform temperature throughout the room space (Mazzarella, 2013). The object-oriented structure of OpenBPS of course allows more detailed room air convection calculations, such as CFD or zonal methods, which can be added in a future development. The "sensible" part of the enthalpy balance constitutes the so-called "air node" ordinary differential equation, here simply referred to as the Air-Node object.

The Air-Node object deals with various advective mass flows such as ventilation air, exhaust air, and infiltration, other than the convective heat transfer with the room surfaces (walls, windows, ceilings, and floors), assumed with uniform surface temperature. It accounts for the thermal capacity of room air and evaluates direct convective heat gains from people and equipment.

The heat transfer through each building fabric component, here simply called Partition, determines the room surface temperatures used in the Air-Node object to calculate the convective heat transfer. The Partition class allows different solvers to integrate the second-order differential equation that describes the diffusive heat conduction: finite difference method, conduction transfer functions and harmonic quadrupole (for comparison tests purposes). Any Partition object can use a different solver when a simulation is run, according to its needs (i.e. one wall can be modelled using finite difference explicit scheme, another full implicit, and another one conduction transfer function).

The long wave internal radiative heat transfer among room surfaces is modelled using the Grey Body Model based on mutual radiation factors, which can be calculated at the initialization stage from the view factor (geometrically defined) and surface emissivities. The mutual radiative heat transfer among all internal surfaces is accounted for when the boundary conditions (BCs) of each object (component) are updated. When the BCs are updated, also short wave irradiation is determined for both external and internal Partition surfaces through the solar radiation processor and the Short-Wave Radiation module (SWRadModule).

This module has the responsibility of:

- setting shaded perimeters in external surfaces by considering external obstructions, such as other buildings, self-shadings, overhangs and fins, (without, however, considering them also as possible direct and diffuse reflectors)
- setting the solar radiation transmitted through the transparent envelope component and non-uniformly distributed among the room surfaces.

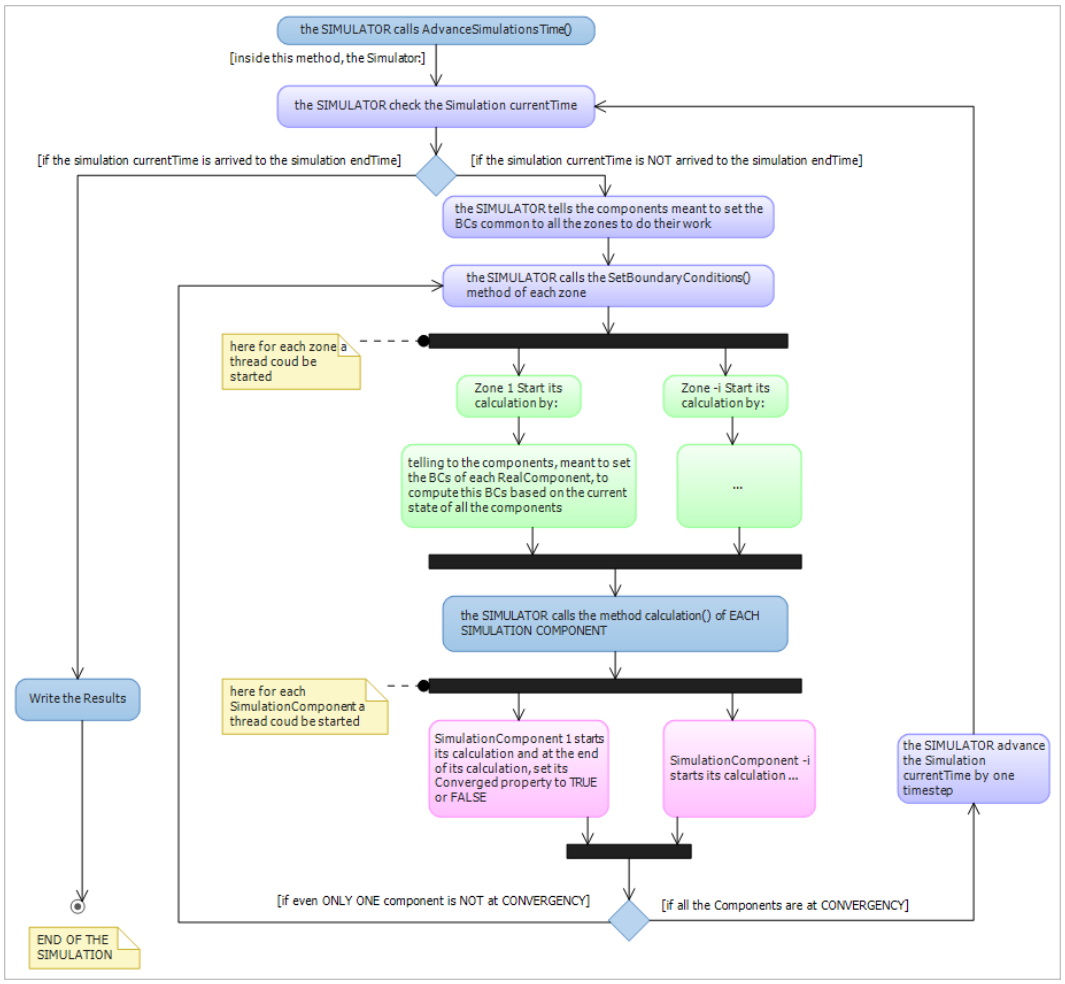

Fig. 3 – OpenBPS: Parallelism handling at different levels, simplified scheme

When dealing with transparent Partitions (windows, etc.), OpenBPS performs an accurate calculation using angular dependence of transmission and absorption for both solar and visible radiation, and temperature dependent U-value calculated at run-time entering a layer-by-layer window description, or simplified calculation entering the windows description taken from third party projects, such as the Berkeley Lab WINDOW 7.4.

Solar control has not yet been introduced, but new models for shadings and shadings control could be introduced easily thanks to the object-oriented structure of the project. A finite number of view factors might be pre-calculated for different positions of the shading and a “transparent layer like” description could be used for handling the SW radiation part, while air convection should be considered with simplified or detailed calculations.

An improved calculation for diffuse solar radiation on tilted surfaces has been implemented as described by (Perez et al, 1990). This non-isotropic model accounts for circumsolar, horizon brightening, and isotropic diffuse radiation through empirically derived "brightening coefficients". These coefficients, function of sun position and cloud cover, have been implemented following the last curve fit performed by Perez in 1999, as reported in both the technical documentation of TRNSYS 17 and EnergyPlus 8.6.0.

2.5 Building Systems Simulation

The Simulator prioritizes the building fabric components before starting to manage the building system simulation. The objects that represent the components of HVAC and electrical systems, energy conversion equipment, and any other needed building technical systems component, are then

asked to expose their outputs at the actual time stamp in organized sequences. The natural modularity of the adopted object oriented scheme supports the realization a fully integrated simulation of loads, systems, and plant, but at the same time it can raise stability issues. To avoid or limit, as much as possible such instabilities, the objects belonging to a specific technical system are organized in "queues" as in the real word, i.e. solved in sequence according to their energy transfer direction. With "queue" we mean a sequence of objects that are solved sequentially one after the other as they interact mainly sequentially, like, for instance, an emitter, the distribution pipes, the boiler, and their thermostatic controls. Of course, networks of pipes and ducts represent close-loop "queues". Thus, each global time step, the Simulator has to iterate among all components, managing to solve at first the unknowns for the building fabric components (mainly in parallel) and then for the building technical systems (mainly serially). This integrated simulation allows capacity limits and control strategies to be modelled more realistically and provides a tighter coupling between the building fabric components and the technical system components, allowing a specific object requirement to override this general management structure, as in active double-skin façades where a HVAC system closely interacts with an envelope component.

The building system part is currently under development because, instead of using performance maps and/or steady state models for the HVAC and plant components, true dynamic models have been targeted, though simplified. This to overcome one of the most significant drawbacks that characterize many of the popular building energy simulation programs: the use of steady state modelled equipment with few-minute simulation time steps. The introduction of a component characteristic time can avoid this misleading use that can lead to unreliable results.

2.6 Input, Output and Weather Data

As mentioned several times, OpenBPS is a set of dynamic link libraries that are mainly designed to be used by third party software. Thus, the user front end is not in the project goal, even though a simple GUI has been developed for developing and testing purposes. This interface allows to import EnergyPlus input files, with some limitations, or files created with the TRNSYS17 or OpenStudio plug-in in SketchUp, always with some limitations. Drag-and-drop features are also implemented to manage tests on components, and assign to them, for instance, different solvers, and so on. For the same reason an output manager allows to print out almost all the necessary variables to analyse the building performance.

The real input to the code is anyhow the Building Object itself: the GUI is just filling up all the required properties of all included objects, simulation requirements included, and then passes it to the *SimulationManager* dll library for simulation execution. During the simulation, the Output Object is filled with the required output variables, warnings and any other information. At the end of the simulation, the Output Object is exposed to the caller program that takes over the task of producing graphs, synopses, and any other specifically formatted output.

The other major data input is weather, provided through weather data files directly read by the simulation engine. The code can directly read standard weather data formats, like TMY and EPW, or custom-made data format. In any case, the weather processor is able to produce the required quantities regardless of the matching between provided data frequency and required data frequency. After a time alignment, the weather processor produces via interpolation the required data if required by an object, with a frequency higher than the recording frequency. If, instead, this frequency is lower than the recording frequency, we have two possibilities. We can communicate to all the simulation components all the climatic data and let them decide if they want to perform multiple calculations without iteration, or if they prefer to store and manipulate the data in accordance with their numerical scheme, and perform their calculation only once. Or we can manipulate the climatic data before exposing them to the components, in which case two different cases arise: the frequency is a multiple of the recording frequency or is not. In the first case, non-integral

data are directly provided to the object, while integral data are cumulated before exposing them to the object. In the second case an additional interpolation is performed to provide the required information.

2.7 Contributing to New Developments

One of the main goals for OpenBPS is to encourage continuous development and enrichment with new features. To achieve this goal, it was decided to adopt an open source approach and to build around it a community of developers who can take care of that. This idea and the tools provided to realize it were already described in a previous paper (Mazzarella and Pasini, 2015a). Nevertheless, the contributing procedure can be summarised as follows: Anyone can download the source code and can do, according to the open source license agreement, what he likes. In order to be a recognized contributor (i.e. to be able to upload to the developing repository his own code that will become part of the version following the official version), a developer has to join the community and follow its rules. The production of new classes of building components is highly encouraged, and it is in principle quite easy. Due to the object-oriented structure and the developing environment (MS Visual Studio under Windows, MonoDevelop under Linux, and MacOS), the process is relatively simple. First, a developer defines a new component by writing down its mathematical and numerical model and identifies the model parameters and needed equations, the specialized coefficients and any other needed data. Next the developer writes the code (using the OpenBPS programming standard) and identifies the "parent" class from which to inherit common properties and methods. If a detailed class is not available, this father class must be the "SimulationComponent" class. In fact, the Simulator at the beginning of the simulation scans all the components to find the one that inherits from that class, thanks to polymorphism. The simulation component class exposes methods that are always called by the Simulator, such as the initialization method, the calculation method, the methods needed to manage convergence and the method for output writing. All new components, while implementing their specific properties and methods, have to inherit from their particular "father" class or override the available implementation of those particular methods defined for the "Simulation Component" class, to be correctly called by the simulator to get input from other objects and to deliver their output.

3. Validation

OpenBPS is continuously under validation, since it is still in its developing phase. Some references to earlier validations can be found in Mazzarella and Pasini (2013, 2015b). Fig. 4 shows several ASHRAE 140 Standard (ASHRAE, 2014) cases: case 600FF, free floating internal temperature (FF) and light walls; case 650FF, i.e. 600FF with night ventilation; case 900FF and case 950FF, respectively free floating with heavy wall without and with night ventilation. These BESTEST validations have been carried out with the actual version of OpenBPS and show quite a good agreement with the provided information.

4. Conclusion and Perspectives

The OpenBPS object oriented building performance simulation tool is not yet ready to be deployed within the energy simulation community because it lacks the building system component models. This part is currently under development by trying to model the system components as dynamic components, even if simplified. On the other side, a decision has to be made on how to initiate the developers' community, to create a true open source project. The main idea that can support and increase its development and diffusion, is to launch it as a "standard de facto" replacing the whole EPBD standard set that will be in force at the end of 2017. This new EPBD set of standards implies that each technical software house that sells programs to assess building energy performance, has to rewrite its calculation engine according to the new standards. Thereafter, they have to require a legal validation before putting it on the market. A joint venture between OpenBPS and the technical software houses may solve the problem of both: a unique full validated calculation engine that can be

used inside any software on any platform, completely documented and expandable time by time.

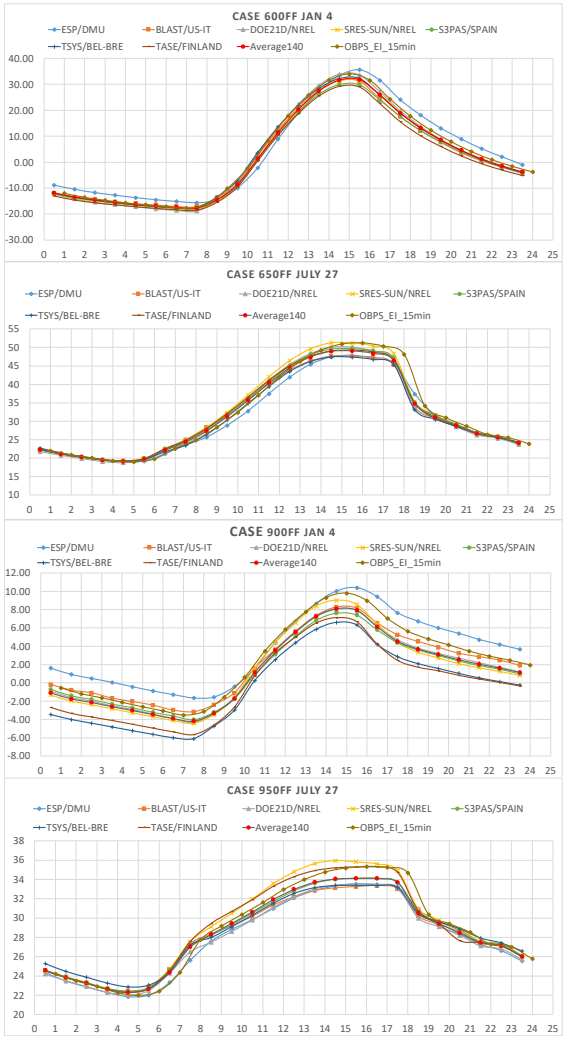

Fig. 5 – ASHRAE 140 Standard OpenBPS validations

Acknowledgement

The development of the code presented in this paper was partially supported by Regione Lombardia through financing under the main project "TRIBOULET: Carbon Footprinting of Products".

References

ASHRAE. 2014. *ASHRAE Standard 140:2014 -- Standard Method of Test for the Evaluation of Building Energy Analysis Computer Programs*. Atlanta, U.S.A.: ASHRAE.

Mazzarella, L., M. Pasini. 2009. "Building energy simulation and object-oriented modelling: review and reflections upon achieved results and further developments". In: *Proceedings of Building Simulation 2009*. Glasgow, UK: IBPSA.

Mazzarella, L. 2013. "The air energy balance equation paradox". In: *Proceedings of Building Simulation Applications BSA 2013, the 1st IBPSA-Italy conference"*. Bolzano, Italy: BUPRESS.

Mazzarella, L., M. Pasini. 2013. "Development of a new tool for the co-simulation of multiple autonomous object". In: *Proceedings of Building Simulation 2013*. Chambéry, France: IBPSA.

Mazzarella, L., M. Pasini, N. Shahmandi Hoonejani. 2014. "Challenges, limitations, and success of cloud computing for parallel simulation of multiple scenario and co-simulation". In: *Proceedings of ASHRAE/IBPSA-USA Building Simulation Conference*. Atlanta, U.S.A.: ASHRAE.

Mazzarella, L., M. Pasini. 2015a. "Advancement in the development of an Open Source Object Oriented BPSt: development methodology". In: *Proceedings of Building Simulation Applications BSA 2015, the 2nd IBPSA-Italy conference"*. Bolzano, Italy: BUPRESS.

Mazzarella, L., M. Pasini. 2015b. "CTF vs FD based numerical methods: accuracy, stability and computational time's comparison". *Energy Procedia* 78: 2620–2625. doi: 10.1016/j.egypro.2015.11.324.

Pasini, M. 2009. *Towards an Enriched Modularity of Building Performance Simulation's Programs: Reconceptualisation and development of an Object-Oriented model*. Ph.D. Thesis, Milano, Italy: Politecnico di Milano.

Perez, R., P. Ineichen, R. Seals, J. Michalsky, R. Stewart. 1990. "Modeling Daylight Availability and Irradiance Components from Direct and Global Irradiance". *Solar Energy* 44, 271-289. doi: 10.1016/0038-092X(90)90055-H.