Title

OpenBuildingControl: Digitizing the control delivery from building energy modeling to specification, implementation and formal verification

Authors

Wetter, Michael

Ehrlich, Paul

Gautier, Antoine

et al.

Publication Date

2022

DOI

10.1016/j.energy.2021.121501

Peer reviewed

Energy 238 (2022) 121501

Contents lists available at ScienceDirect

Energy

journal homepage: www.elsevier.com/locate/energy

OpenBuildingControl: Digitizing the control delivery from building energy modeling to specification, implementation and formal verification

Michael Wetter [a, *], Paul Ehrlich [b], Antoine Gautier [a], Milica Grahovac [a], Philip Haves [a], Jianjun Hu [a], Anand Prakash [a], Dave Robin [c], Kun Zhang [a]

[a] Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA, USA
[b] Building Intelligence Group, Portland, OR, USA
[c] BSC Softworks, Atlanta, GA, USA

ARTICLE INFO

Article history:
Received 25 February 2021
Received in revised form
27 June 2021
Accepted 13 July 2021
Available online 29 July 2021

Keywords:
Control
Building
HVAC
Simulation

ABSTRACT

The current process for specifying, installing and commissioning building control sequences is largely manual and based on ambiguous natural language specifications. It lacks a formal end-to-end quality control and it has been shown not to deliver high performance sequences at scale. While high-performance HVAC control sequences enable significant reductions in energy consumption, errors in implementing the control logic are common even for less advanced sequences. To improve this situation, we present a digitized building control delivery workflow with formal end-to-end verification, a Control Description Language for the digital specification of building control sequences within this workflow, and software tools that enable digitization of this process. Using the process and tools introduced here, mechanical designers can customize, test and improve these sequences within annual energy simulation, store them in a library for use in other projects, and export them for bidding. Control providers can implement the sequences on existing control product lines through code generation. Commissioning providers can formally verify whether as-installed sequences conform to the digital design specification that was exported by the mechanical designer. Moreover, control product development teams can use the reference implementations of these libraries within their product testing to ensure that their products reproduce the behavior of the reference implementations. This paper presents this process, the language and the supporting software, together with examples of all of the above steps. The presented work has given rise to a new proposed standard, ASHRAE 231P, that will allow digitizing the building control delivery process through the standardization of a control-vendor independent format for exchanging control logic that we pioneered through the here presented work.

1. Introduction

High performance building control sequences have been shown to significantly reduce energy consumption, with savings in the range of 23 %–30 % being common for most building types [11]. This requires control sequences to be properly designed and implemented. However, it also has been shown that in large commercial buildings with built-up HVAC systems, programming errors are the leading cause of control related problems [4]. The fact that control related problems are a key contributor to missed energy savings has been confirmed by a subsequent study from different authors, which identified in existing commercial buildings 481 operational issues. This study estimates the correction of control related problems to account for more than 75 % of the potential energy savings obtained in commissioning [9]. This energy savings potential is not surprising as today's process to specify, implement and verify control sequences is based on ambiguous and often incomplete English language specification of the controls intent that produces low quality implementations. Energy efficiency, occupant- and grid-responsiveness of control sequences are difficult to quantify and realize. As a result, the expected energy performance is often not achieved. This poses risks to building owners

* Corresponding author.
E-mail address: mwetter@lbl.gov (M. Wetter).

https://doi.org/10.1016/j.energy.2021.121501

as the return of investment of energy saving measures may not be achieved, the HVAC system may be oversized to compensate for malfunctioning and occupant comfort may not be achieved. Missed energy savings can be estimated as follows: An LBNL study identified 16 % median actual savings from retro-commissioning [24]. Therefore, if one assumes that around 75 % of the 16 % expected energy savings associated with commissioning relate to controls, then simply by ensuring that controls work as intended, about 12 % energy can be saved. Assuming that the workflow described in this paper is primarily applicable to built-up HVAC systems and these are used in buildings larger than 50, 000 sf, which account in the US for 50 % of the commercial floor area, the estimated savings average 6 % across all US commercial buildings.

The current controls design process is such that the mechanical designer may at best specify building control sequences in an English language specification. However, such a specification cannot be tested formally for correctness. It is also ambiguous, leaving room for different implementations, including variants that were not intended by the designer or may not work correctly. The implementation of the sequences is often done by a controls contractor who either attempts to implement the sequence as specified, or use a sequence of a similar project that appears to have the same control intent. During commissioning, the lack of an executable specification of the control sequence against which the implementation can be tested makes commissioning of the control sequences expensive and limited [15].

To change this status quo, we are working on digitizing the control delivery process. In support of this effort, we have been developing tools and a process that allow for a digital control specification, performance assessment using whole building energy simulation, and delivery and implementation on existing building automation product lines. The technical environment for such a digital control delivery starts to fall in place: For communication of control signals, standards such as BACnet, LonWorks and EIB/KNX are widely used. For semantic modeling, Haystack and Brick are increasingly used, and ASHRAE Standard 223P aims to standardize semantic modeling, building on these previous efforts. However, what is missing is a means to express control sequences in a way that enables simulation during design, export of control specification and documentation, translation to commercial product lines through code generation and reuse for formal verification of the correct implementation of the control sequences. This gap is what the OpenBuildingControl project attempts to close. A key element of OpenBuildingControl is the Control Description Language (CDL) that has been developed in the project. CDL is a declarative language for expressing control sequences through block diagram modeling. To enable simulation of closed loop control as part of annual energy modeling during building design or control research, we designed CDL to be a proper subset of the Modelica language, an open standard for an equation-based object-oriented modeling language [23,26]. As CDL is a declarative language, the control specification can be exported in a vendor-independent json format that serves as an intermediate format to produce English language documentation including control point lists, and that can serve as inputs to a code translator to a particular control product line. The control specification can also be exported for use in a formal workflow that verifies that the control signal of the actual implementation is within a user-selected tolerance of the simulated control signal. This provides a workflow with an end-to-end verification as shown in Fig. 1. Therefore, CDL complements communication (BACnet) and semantic modeling (ASHRAE 223P) by expressing the control logic, with the goal of standardizing this missing part of the control representation. To the best of our knowledge, this is the first demonstration where the performance of a building control sequence has been tested in annual simulation,

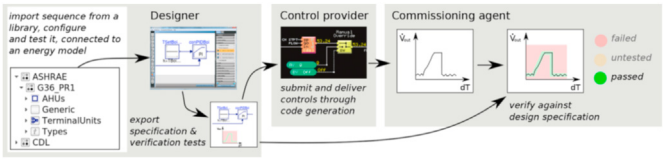

Fig. 1. Overview of the workflow for control sequence design, export of a specification, implementation on a control platform and verification against the specification.

the sequence has been translated through code generation to the language used natively in a commercial control product line, and the sequence computed by the controller has been formally verified against its digital representation. Thereby, this demonstration bridges the gap between energy modeling and control deployment and controls commissioning.

We believe that the time for such an effort is ideal due to the convergence of various technologies. These include the declarative open standard Modelica that allows closed loop control modeling and annual performance simulation, advances in code generation that ease machine-to-machine translation of declarative languages, and semantic modeling. The latter promises to enable the use of a semantic model that was automatically generated from a declarative Modelica model [13] for subsequent semi-automatic connection to an actual building in which a digital twin of the control and of the building systems could be used to support building analytics, for example using the MORTAR framework [12]. Due to the trend towards all electric buildings, which, to increase 2nd law efficiency, should no longer decouple subsystems through large temperature lifts (as is customary in fossil-fuel based heating systems), and the resulting need for more complex control which, in addition, also need to provide grid flexibility, we believe such a convergence of technology will help the industry achieving higher system-level performance.

The paper is structured as follows: Section 2 discusses related work. Section 3 describes the methodology, starting with the workflow and then explaining the Control Description Language on which the methodology is based on. This section also describes the library of user-configurable control sequences, and our supporting work on modeling and simulation environments to enable closed loop performance evaluation. It also describes tools for translating control sequences to control product lines and for formal verification of control sequences that we have been developing. In Section 4, we present examples for closed-loop performance evaluation using annual simulation, code generation of control sequences to a commercial control product line, and formal verification of control sequences implemented on a commercial controller. The paper ends with a discussion in Section 5 and concluding remarks in Section 6.

2. Related work

Related work includes the following:

ISO 16484−3 [20], Building automation and control systems (BACS), specifies the characteristics of software and functions for building automation systems, and a method for documentation of the design. It defines function blocks with their inputs, outputs and parameters, but is vague about the logic that is in these function blocks.[1] ISO 16484-3 Annex A provides a building automation control system function list (BACS FL), represented as a

[1] For example, there is no specification of the anti-windup of a PI controller. Moreover, while it defines an optimum start up, it does not provide an implementation and is vague about some of its parameters.

spreadsheet. A BACS FL documents what equipment has what I/O, is controlled by what function and how it is monitored (e.g., a supply air temperature has one analog input, is controlled by a P-controller and is on a dynamic display). Such functionality is then added up in the spreadsheet. However, BACS FL cannot fully describe the required control methods, especially in case of sophisticated and non-standard control functions. These will need to be documented in additional documents. Thus, BACS FL are not meant to provide an executable specification and they appear to be not suited as a basis to provide a formal language that meets our needs. However, elements of ISO 16484-3 such as Function Blocks (FB) that are used to describe the inputs, outputs and parameters of BACS functions are also used in CDL. Moreover, it appears that BACS FL could be generated from CDL.

In contrast to ISO 16484−1, CDL allows systems to be tested during the design phase, while in the ISO project implementation process, the design approval is done prior to any testing, as Fig. 1 in ISO 16484−1 [19] shows.

IEC 61131−3 [18] has been developed for industrial process control. It standardizes Programmable Organization Units (POU), which can be extended for object oriented programming including support for polymorphism. It also standardizes a Sequential Flow Charts (SFC) model that can be used to implement state machines. It standardizes the graphical languages Ladder Diagram (LD) and Function Block Diagram (FBD). The sequential function chart (SFC) elements can be used in conjunction with either of these languages. IEC 61131−10 [17] standardizes an XML format for the textual language, instruction list (IL), the textual language, structured text (ST), the graphical language, ladder diagram (LD), the graphical language, function block diagram (FBD), and sequential function chart (SFC). As IEC 61131-3 defines executable languages with complex objects and rules for composing control algorithms, it appears difficult if not impossible to use IEC 61131−3 as a basic format that is then translated to building automation systems, as building automation systems often provide significantly fewer capabilities. We therefore designed CDL, which better matches the programming tools and techniques in place in the buildings industry. It seems feasible to translate CDL to IEC 61131−3, but we have not yet attempted to do so.

Donida and Leva [8] developed within the AutoEdit project a graphical application that converts LD and SFC that adhere to the IEC 61131-3 standard to Modelica. Their approach is to generate algorithmic Modelica code, which can then be linked up to equation-based Modelica models of the plant for closed loop simulation.

Husaunndee et al. [16] developed a MATLAB/Simulink-based toolbox of models of HVAC components and plants for the design and test of control systems called SIMBAD. SIMBAD has been used for testing and emulation of building control sequences, and is commercially distributed by CSTB France. Bonvini and Leva [5] developed an industrial control library in Modelica. Yang et al. [39] developed a tool chain that maps Simulink and Modelica models into an intermediate format, and then refined it for implementation in distributed controllers. Our approach borrows ideas from their methodology. Schneider et al. [30] implemented a Modelica library with standardized control functions for building automation. They use control functions from VDI 3813−2:2011 and state graph representations from VDI 3814−6:2009, implemented by extending the models from the Modelica.StateGraph package [28]. Our approach differs from their work as they document the control using Unified Modeling Language (UML) class and activity diagrams. Schneider et al. [29] formalized the approach through the introduction of CTRLont, and ontology for modeling control logic in building automation systems. Our approach differs in that our representation uses block diagrams that can be composed hierarchically using a predefined set of elementary blocks. These are then mapped to implementations of commercial control product lines. In our approach, a related project is working on extracting ontological information from such models and exporting it to Brick [3]. Thus, our approach has similarities to that of Schneider et al. [29], but in our case, the English language description of the sequences and the ontological information are exported from CDL. The latter can be added to the CDL sequences, or be autogenerated if the sequences are part of a model that also contains the HVAC system, as shown in Fierro et al. [13]. Our approach differs from Donida and Leva [8], Bonvini and Leva [5], Yang et al. [39] and Schneider et al. [30] in that we use elementary control blocks that form a basic library of control functions, and simple composition rules that we believe suffice for composing building control sequences. As we will see in Section 4.2, the control blocks are preserved in our translation from CDL to the input of the code generator of a control product line. This decision was made to enable the reuse of existing control product lines, possibly with minor modifications, in our workflow.

3. Methodology

3.1. Workflow

Before discussing the main elements of the workflow, we first provide in this section a high-level overview of the end-to-end workflow. Given regulations and efficiency targets, labeled as (1) in Fig. 2, a design engineer selects, configures, tests and evaluates the performance of a control sequence using building energy simulation (2), starting from a control sequence library that contains ASHRAE Guideline 36 sequences, as well as any user-added sequences (3), linked to a model of the mechanical system and the building (4). If the sequences meet closed-loop performance requirements, the designer exports a control specification, including the sequences and functional verification tests expressed in the Controls Description Language CDL (5). Optionally, for reuse in similar projects, the sequences can be added to a user-library (6). This specification is used by the control vendor to bid on the project (7) and to implement the sequence (8) in product-specific code. Prior to operation, a commissioning provider verifies the correct functionality of these implemented sequences by running functional tests against the electronic, executable specification in a commissioning and functional verification tool (9). If the verification tests fail, the implementation needs to be corrected and the tests repeated until the tests pass.

For closed-loop performance assessment, step (2) in the figure, Modelica models of the HVAC systems and controls [34] can be linked to a Modelica envelope model [33] or to an EnergyPlus envelope model. This can currently be done through Spawn of EnergyPlus [38]. Library of control sequences, step (3), have been released with the Modelica Buildings Library 7.0.0 and more sequences are currently added to this library. To export control sequences in a vendor-neutral format, step (5), a translator from CDL to a json intermediate format has been developed [22]. The json intermediate format is to be used as input for cost estimation tools and for translators to vendor-specific product lines. This translator also outputs an English language description of the control sequence, including its block diagram representation.

3.2. Control Description Language

This section describes the Control Description Language (CDL) that we developed to support the OpenBuildingControl workflow described in this paper.

The requirements of CDL include that it can be used within

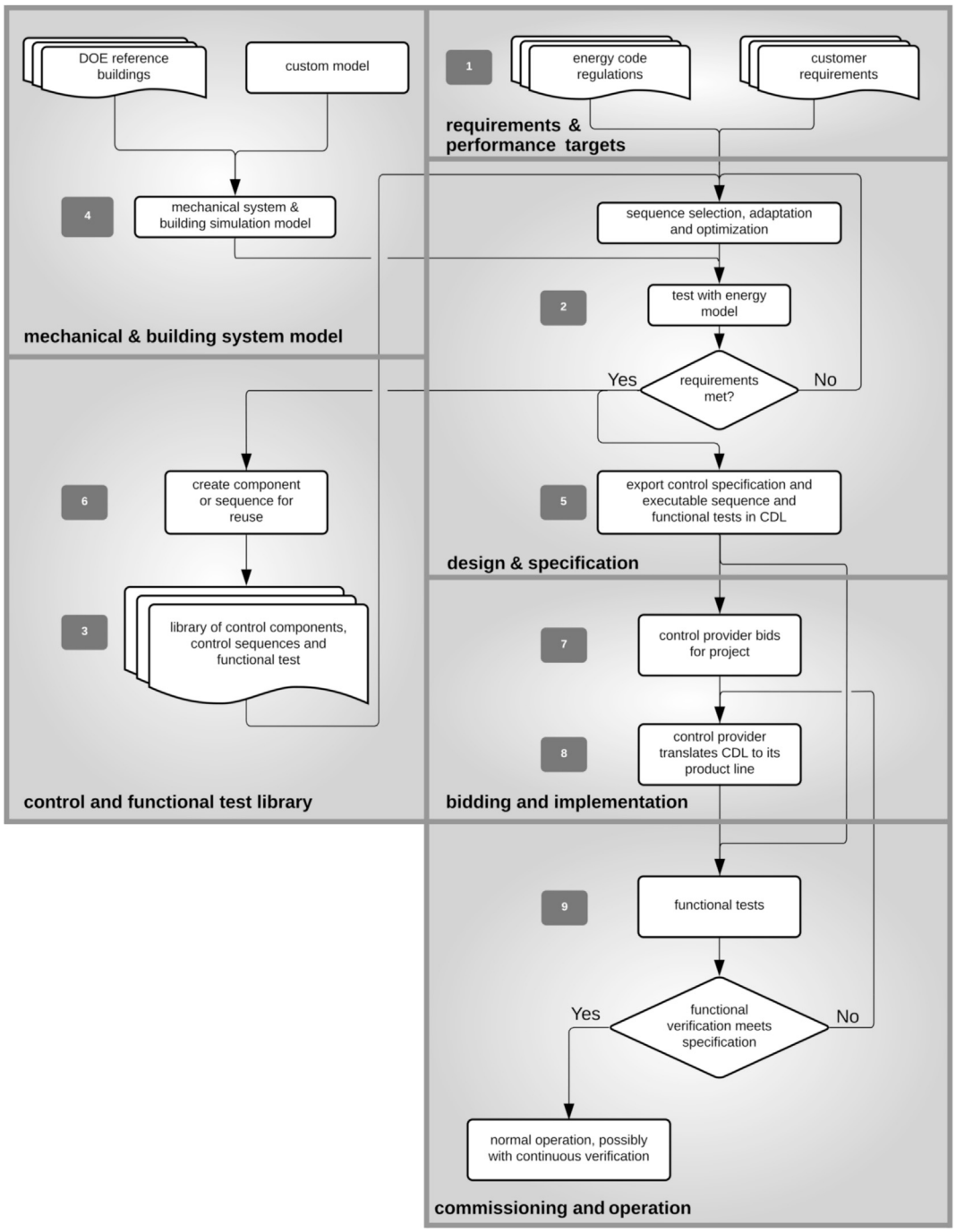

Fig. 2. Overview of the OpenBuildingControl control design, deployment and verification workflow. See text for explanation of the numbers in the grey boxes.

annual whole building energy simulations, and that it can be executed on various building automation systems, either through code generation that translates CDL into a representation needed by the particular control system, or through translation to C code, which may be encapsulated as a Functional Mockup Unit (FMU) in order to use the open FMI standard [27] or the emerging eFMI standard [10]. Therefore, the language needs to be declarative (in order for it to be translated to other representations), simple (to lower the barrier to develop translators) yet expressive enough to be able to express control sequences and to be used as part of an annual building energy simulation. Moreover, as we intend to formally verify that the sequences installed in a building control system meet the design specification, we require control sequences that conform to CDL to allow a deterministic computation. By deterministic computation we mean that control signals, computed by implementations of the sequence in two different control systems or simulators, need to be identical as defined by Broman et al. [6].

A key technical challenge that we encountered was that, due to a lack of standards, existing building control product lines are heterogeneous. They differ in their functionality for expressing control sequences, in their semantics of how control output gets updated, and in their syntax, which ranges from graphical languages to textual languages. Code generation for a variety of products is

common in the Electronic Design Automation industry, which develops software tools for designing electronic systems such as integrated circuits and printed circuit boards. However, in the Electronic Design Automation industry, engineers write models and controllers are built to conform to the models. If this workflow were to be applied to the buildings industry, then control providers would need to update their product lines. We do not expect that such costly product line reconfigurations can reasonably be expected in the next decade. Therefore, for the immediate future, the CDL translation process will need to involve the building of models of control sequences that can conform to their implementation on target control product lines, while ensuring that, as new product lines are being developed, they can invert the paradigm and build controllers that conform to the models. The project team has, therefore, selected the path of designing CDL in such a way that it provides a minimum set of capabilities that can be expected to be supported by control product lines, yet allows for use of recent design automation workflows based on the FMI or eFMI standards.

We designed CDL as a declarative, modular language for expressing block diagrams. CDL allows hierarchical modeling to encapsulate and reuse through object instantiation preconfigured control sequences. CDL also defines syntax for connecting inputs to outputs, for propagating the values of parameters and for encapsulating composite blocks that can be stored in a library for reuse. CDL also has annotations that declare how to graphically render the block diagrams. It supports the provision of documentation in HTML format. For simplicity and for compatibility with building control platforms, CDL currently does not support object-inheritance and use of state machines, although the former could easily be added if objects are extended prior to translation.

For the implementation of CDL, we used a small subset of the Modelica language that is needed for convenient declaration of block diagrams. We selected Modelica as it is an open standard, as it provides various open-source and commercial modeling and simulation environments, as it allows to generate highly efficient code for simulation, and because it is increasingly used to simulate building energy and control systems. In particular, the US Department of Energy has been funding the redesign of the EnergyPlus simulation software so that it uses Modelica for the HVAC and control simulation (see Section 3.4). As the model of computation, CDL uses the synchronous data flow principle and single assignment rule, which is consistent with the Modelica 3.4 Language Specification [26, Sec. 8.4]. Therefore, all variables keep their value until the value is explicitly changed, values are always present (and hence can be accessed at any time instant), computation and communication at an event instant do not take time, and every input connector must be connected to exactly one output connector.

Besides these rules, CDL also prescribes a library of elementary building blocks that need to be supported by implementations that support CDL. The functionality of elementary building blocks, but not their implementation, is part of the CDL specification. Thus, in the most general form, elementary building blocks can be considered as functions that for given parameters p, time t and internal state $x(t)$, map inputs $u(t)$ to new values for the outputs $y(t)$ and states $x'(t)$, e.g.,

$$(p, t, u(t), x(t)) \mapsto (y(t), x'(t)). \tag{1}$$

Control providers who support CDL need to be able to implement the same functionality as is provided by the elementary CDL blocks. CDL implementations are allowed to use a different implementation of the elementary building blocks because the implementation is language specific. However, implementations shall have the same inputs, outputs and parameters, and they shall compute the same response for the same value of inputs and state variables. Users are not allowed to add new elementary building blocks. Rather, users can use them to implement composite blocks. Fig. 3 shows an actual implementation of an elementary building block. Such elementary building blocks are organized into the packages shown in Table 1. Using blocks from these packages and the CDL syntax for connecting inputs to outputs and for propagating parameter values, users and library developers can create composite blocks that encapsulate more complex control sequences and store them in a library for reuse. For example, if one would like to implement a custom proportional controller with output limiter that computes the output signal $y = \max(k\, e, y_{max})$, where k is a parameter, and e and y_{max} are control inputs, one can implement the code shown in Fig. 4 and store it in a file CustomPWithLimiter.mo. A visual editor may render the block as shown in Fig. 5. The specification of the CDL language was introduced by Wetter et al. [35].

3.3. Library of control sequences

Using the composition rules defined in the CDL language specification, library developers and end-users such as mechanical engineers can compose control sequences for a given system, and optionally store them in a library for use in other projects or to share them with others.

In the Modelica Buildings library, we used CDL to implement a subset of the sequences published in ASHRAE Guideline 36 [2]. Fig. 6 shows an overview of the implemented sequences. The implementation is structured hierarchically into packages for air handler units, into constants that indicate operation modes, into generic sequences such as for a trim and respond logic, and into sequences for terminal units. For every sequence, there is a validation package that illustrates the use of the sequence.

Using CDL, we also implemented sequences for outdoor lights and for shading control, and implementations for primary plants according to ASHRAE Research Project 1711 and for radiant systems are in progress.

Our vision is that the simulation community will create and share packages of ready-to-use control sequences. Similarly, if a design firm uses their own, possibly proprietary sequences, they can build a library and share them within their company. Hence, the library will become a means to share expertise in building control sequences, and also to continually improve the sequences from one project to another.

3.4. Modeling and simulation environment

As CDL conforms to the Modelica Language Specification, various Modelica modeling and simulation environments can be

```
block AddParameter
  "Add a parameter to the input signal"
  parameter Real p "Value to be added";
  parameter Real k "Gain of input";
  Interfaces.RealInput u
    "Connector of Real input signal";
  Interfaces.RealOutput y
    "Connector of Real output signal";
equation
  y = k*u + p;
  annotation(Documentation(info(
  "<html> ... [omitted] </html>"));
end AddParameter;
```

Fig. 3. Implementation of an elementary CDL block. (Graphical annotations are omitted.)

Table 1
Structure of the CDL library.

Package name	Description
Constants	Package with constants, such as π
Continuous	Package with blocks for continuous variables, such as an adder, gain, integrator with reset, etc.
Conversions	Package with blocks for type conversion, such as for Boolean to Integer conversion
Discrete	Package with discrete blocks, such as a time-sampled zero-order hold, or a sampler with a trigger input signal
Integers	Package with blocks for Integer variables
Logical	Package with logical blocks, such as an "and" block and a delay of a Boolean signal
Psychrometrics	Package with psychrometric blocks
Routing	Package with blocks that combine scalar signals into vectorized signals, and extract a scalar signal from a vectorized signal
Utilities	Package with utility functions, such as to report a warning
Types	Package with type definitions, such as used to configure a controller as P, PI, PD or PID, or to declare whether a controller allows reset of its integrator to an input signal, to a parameter or to disable the reset option
Interfaces	Package with connectors for input and output signals

used to model and simulate CDL conforming control sequences. The CDL implementation has been tested with the OPTIMICA, Dymola, OpenModelica and JModelica.org programs, which all provide a modeling and simulation environment for Modelica. The US Department of Energy is also funding a substantial redesign of the EnergyPlus whole building energy simulation program called Spawn of EnergyPlus [38]. In this redesign, the HVAC and control models are implemented using the Modelica Buildings Library, and the EnergyPlus envelope model is refactored to allow a tight coupling with the HVAC, control and room air models. Spawn is being developed in such a way that the co-simulation between the Modelica HVAC and control simulation and the EnergyPlus envelope simulation is set up automatically to allow an integrated closed loop performance assessment. Spawn enables the user to

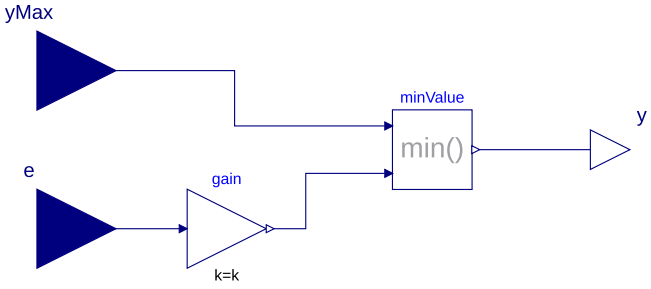

Fig. 5. Graphical rendering of a the composite control block shown in Fig. 4.

use drag and drop modeling in any Modelica-compliant model authoring tool.

For example, Fig. 7 shows a very simple configuration of a thermal zone, linked to a simple HVAC model that recirculates room air and heats it up, based on a CDL control sequence that computes the supply air set point temperature. While this very simple model illustrates the use for a building with one thermal zone, Spawn supports modeling any number of thermal zones, and also any number of buildings, thereby also supporting the development of district energy systems. In Section 4.1, we show the annual closed-loop performance assessment of a more detailed HVAC system model with CDL control sequences.

3.5. Translation of control sequences

CDL is a language that is convenient for modeling of controls. However, while Modelica environments can typically be used to export CDL conforming control sequences as FMUs or as C code, possibly using the emerging eFMI Standard [10], building automation systems have not been designed to execute FMUs or to upload C code. Moreover, building operators, as well as service

```
block CustomPWithLimiter
  "P controller with variable output limiter"
  parameter Real k "Constant gain";
  CDL.Interfaces.RealInput yMax
    "Maximum output value";
  CDL.Interfaces.RealInput e "Control error";
  CDL.Interfaces.RealOutput y "Control signal";
  CDL.Continuous.Gain gain(final k=k) "Constant gain";
  CDL.Continuous.Min minValue
    "Outputs the minimum of its inputs";
equation
  connect(yMax, minValue.u1);
  connect(e, gain.u);
  connect(gain.y, minValue.u2);
  connect(minValue.y, y);
  annotation (Documentation(info="<html>
<p>
Block that outputs <code>y = min(yMax, k*e)</code>,
    where
<code>yMax</code> and <code>e</code> are real-valued
input signals and <code>k</code> is a parameter.
</p>
</html>"));
end CustomPWithLimiter;
```

Fig. 4. Implementation of the composite control block that outputs $y = \max(k\,e, y_{max})$ where k is a parameter that is shown in Fig. 5. (Graphical annotations are omitted.)

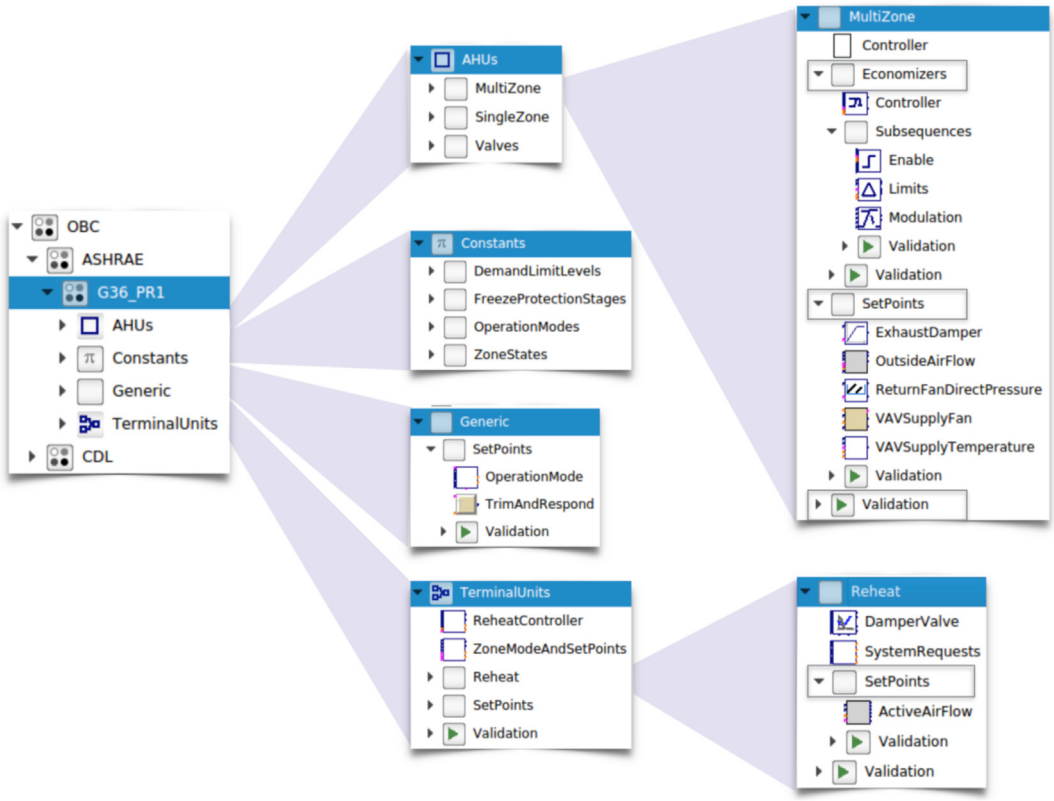

Fig. 6. Overview of the ASHRAE Guideline 36 package implemented in the Modelica Buildings library 7.0.0.

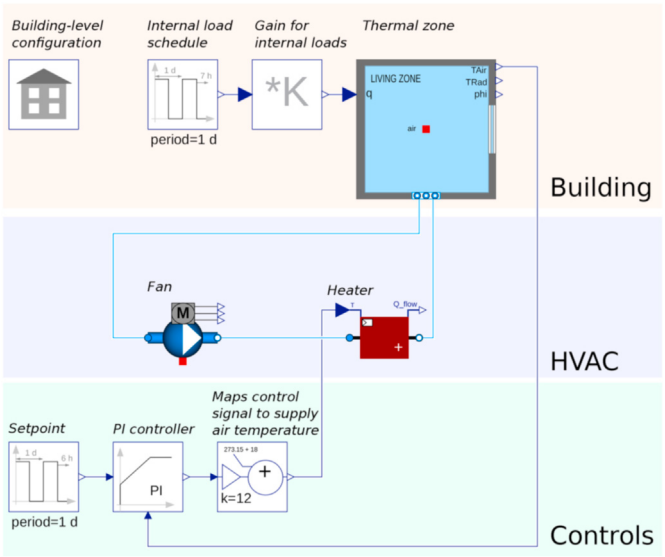

Fig. 7. Schematic view of a simple Spawn model with one thermal zone, an ideal heating system that recirculates air, and a PI controller.

technicians, require implementation of the control sequences in a particular product line, and building automation systems also provide functionality that is generally handled differently among the control providers, often using proprietary approaches, such as distribution of the code to field devices, communication and access control, trending, alarming, etc. In addition, for project submittals and for control documentation for building operators, an English language documentation needs to be provided.

Therefore, to support this workflow, we implemented the software *modelica-json* that translates CDL to a JSON format [22]. This JSON format can then be used as input to a code generator that translates CDL to the syntax used by the particular control platform. This is demonstrated in Section 4.2. The *modelica-json* translator also uses this JSON intermediate format to test the conformance of the control sequence specification with the CDL syntax definition, and to generate the English language documentation of the control sequence together with the corresponding point list, which may be used for bidding, and the graphical representation of the block diagram in HTML or Microsoft Word format. While the translation via a JSON intermediate format has been developed to accommodate the use of CDL with legacy building control product lines, it does not preclude companies that develop new control product lines to directly generate code from CDL using the eFMI Standard and through this support CDL natively.

3.6. Verification of installed sequences

The last step of the deployment is a formal verification that tests whether the control sequences conform to the specifications that were exported during the design phase. This step verifies whether the sequences compute similar control actions and set points as the specification, using trended data as input. Because proper functioning mechanical equipment and communication is orthogonal to the execution of the control logic, these aspects are not tested in this step and need to be done separately as part of the commissioning and are not discussed here.

Fig. 8 shows the verification flow diagram. The verification starts with trending control inputs and outputs from the actual building automation system, with an input excitation large enough to cover

the typical operation of the sequences. The trended inputs and outputs are stored, for example, in a CSV file, at a frequency high enough to resolve the dynamic transients of the input and outputs. These trended data are then read into a model that conducts the verification. This model can be constructed from components of the Modelica Buildings Library. The block labeled *input file reader* reads the archived data. The block labeled *unit conversion* converts the units to be compatible with the units used in the CDL specification. These signals are then fed into the CDL specification. Finally, the outputs from the CDL specification and the control outputs with units converted to be compatible with output signals of the CDL specification are saved to a file.

To compare the trended and simulated control outputs, we developed an open-source, cross-platform software called funnel [21]. The software reads two CSV files, one containing the simulated and the other the trended control signals. It then computes a tolerance around each simulated control point, using the L_1-norm, e.g., a rectangle, with user-specified tolerances for time and the control signal. Next, the software selects which corners of the tolerance rectangles are to be used to build the envelope around all these points. The comparison then consists of interpolating the upper and lower envelopes at the trended control signals $y : [t_0, t_1] \to \Re$, where $[t_0, t_1]$ is the time range for which the data have been trended. This yields an upper bound $y_{up} : [t_0, t_1] \to \Re$ and lower bound $y_{low} : [t_0, t_1] \to \Re$. Finally, the error is computed as

$$e(t_k) = \max(0, y(t_k) - y_{up}(t_k)) - \min(0, y(t_k) - y_{low}(t_k)), \qquad (2)$$

where $t_k \in [t_0, t_1]$ are the time instants of the reference trajectory. These errors are then reported in a time plot and a file, and if all $e(t_k) = 0$, the sequences conform to the specification. In addition to these time series comparison, the process shown in Fig. 8 also allows creating sequence diagrams. These can further aid in understanding why a sequence failed the verification.

For an example of this verification, see Section 4.3, and for a more in-depth discussion, see Wetter et al. [37].

3.7. Verification as part of product line development

Another important use case for OpenBuildingControl is to support verification as part of product line development by control manufacturers. Control manufacturers who do not have a sequence translation as described in Section 3.5 will manually program control sequences. For this use case, we prototyped a sequence verification tool that uses the steps shown in Fig. 9.

The input to this process are the CDL specifications of the control sequences, together with test input trajectories, tolerances for accepted deviations of output values, and indicator variables. Indicator variables can be used to suspend a test during certain conditions. For example, when the fan is commanded off, the verification of supply air temperature set points may be suspended, and restarted when the fan switches on.

The workflow for the verification is as follows: Using the modelica-json translator, the CDL sequence is converted from Modelica to its JSON representation. From this translation, the software extracts the names, units and values of the parameters, along with the names and the units of the input and the output points of the sequence. Next, the software uses the OpenModelica simulator [14] to generate simulated reference output time-series for the provided set of input time-series values. After the user sets up a mapping from the point names and units of the inputs, outputs and parameters in the CDL sequence to their corresponding points on the real controller, the reference input time series are converted to the units used by the controller. Next, the BACnet writer sets these values to the corresponding points of the controller. Periodically, the BACnet writer reads values from the controller and trends them to a file. Finally, the trended values of the output variables from the real controller are converted to the same units as the CDL reference outputs. Now, the reference and trended output time series are compared using the funnel software, as described in Section 3.6. A chart detailing the comparison for each output variable is also generated at the end of each test. This verification can also be done for outputs of subsequences that are used to compose the top-level control sequences, thereby providing a means to debug where deviations first occur.

For an example of this verification, see Section 4.4.

4. Examples

We will now present three examples that illustrate the end-to-end workflow. The example in Section 4.1 demonstrates how the library of control sequences (Section 3.3) can be used within an energy modeling environment (Section 3.4) to test the correctness and performance of control sequences coupled to a building energy model. This is step (2) in Fig. 2. Next, the example in Section 4.2 shows how sequences from such an energy model have been translated to a commercial control product line, using the workflow described in Section 3.5. This is step (8) in Fig. 2. Lastly, the examples in Section 4.3 and 4.4 demonstrate how a sequence implemented in that commercial control product line can be formally verified using the workflow described in Section 3.6 and

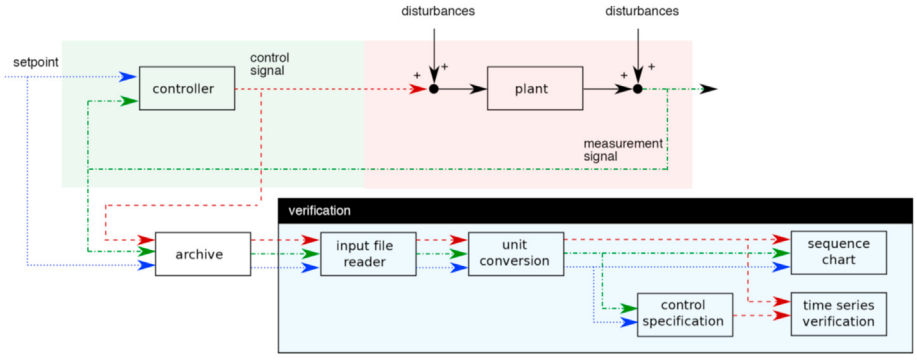

Fig. 8. Overview of the verification that tests whether the installed control sequence meets the specification.

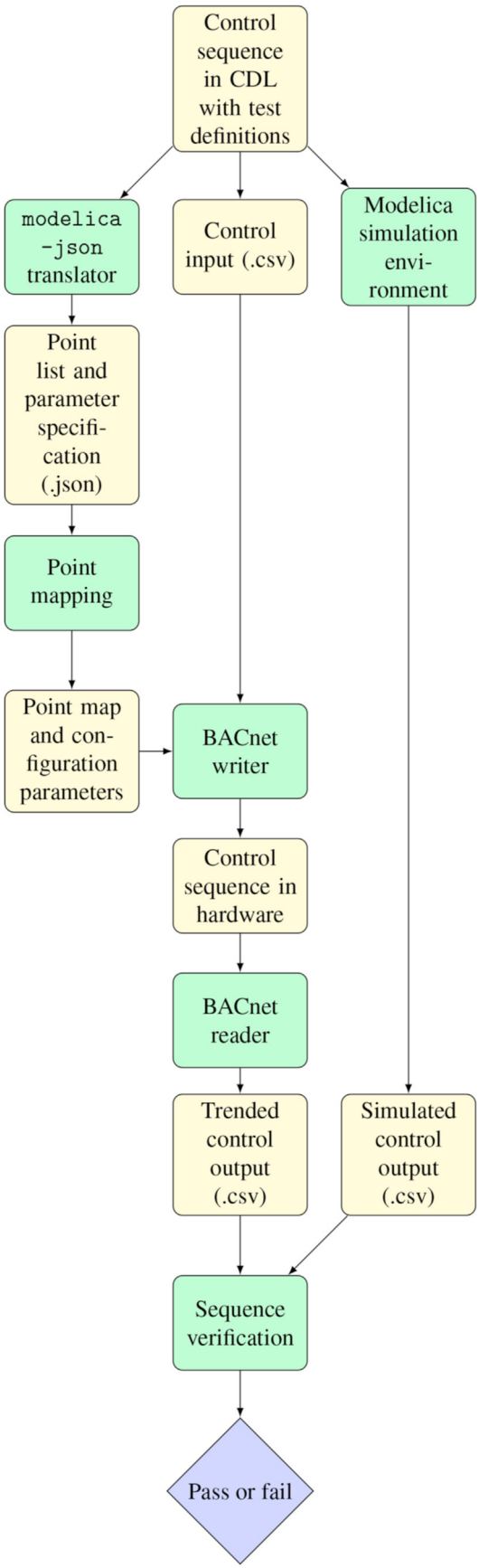

Fig. 9. Different files (yellow) and processes (green) involved in the sequence verification tool.

3.7. This is a part of step (9) in Fig. 2.

4.1. Performance comparison of control sequences

In this section, we compare the performance of two different control sequences. The objectives are to demonstrate the setup for closed loop performance assessment, to demonstrate how the control performance can be benchmarked, and to assess the difference in annual energy consumption.

The two compared cases use the same building model. It consists of a single floor with five thermal zones from the prototypical medium office building described in the set of DOE Commercial Building Benchmarks [7]. The two cases also use the same HVAC system model, representing a variable air volume (VAV) system with terminal reheat. The weather data corresponds to a typical meteorological year (TMY3) in Chicago, IL.

The two cases only differ by the control sequence they use. For the base case, we implemented a control sequence published in ASHRAE's Sequences of Operation for Common HVAC Systems [1]. For the other case, we implemented the control sequence published in ASHRAE Guideline 36 [2]. The main conceptual differences between the two control sequences are as follows:

- *Duct static pressure:* The base case resets the supply air static pressure set point based on the maximum VAV box damper opening, using a PI control loop. The Guideline 36 sequence uses a trim and respond reset logic based on the zone pressure requests sent by the VAV terminal unit controllers.
- *Supply air temperature:* The base case uses a constant supply air temperature set point during occupied hours, whereas the Guideline 36 sequence uses a trim and respond reset logic based on the outdoor air temperature and the zone cooling requests sent by the VAV terminal unit controllers.

In both cases, the set point is tracked with a control loop whose output is mapped to sequence the heating coil, the economizer dampers, and the cooling coil. In the base case, the outdoor air and return air dampers are actuated in tandem (with complementary positions) whereas Guideline 36 prescribes to actuate them in sequence, with first opening the outdoor air damper and then closing the return air damper as the control loop output increases.

- *Minimum outdoor air flow rate:* The base case uses a fixed minimum outdoor air flow set point, computed per ASHRAE Standard 62.1. The Guideline 36 sequence solves dynamically the equations from ASHRAE Standard 62.1 based on the actual operating conditions. In our case, the option for room occupancy sensors is disabled so the variable that drives the set point variation is the zone air distribution effectiveness, which switches from 1.0 to 0.8 when the discharge air temperature becomes higher than the zone air temperature.

In the base case, the set point is tracked with a PI control loop which directly modulates the position of the economizer dampers, whereas the Guideline 36 sequence maps the loop output to the minimum and maximum opening of the outdoor air and return air damper, respectively.

- *Room air temperature:* In heating demand, the base case uses a constant air volume flow rate set point to modulate the VAV box damper opening, whereas Guideline 36 uses a variable set point modulated by the zone temperature control loop output.

Fig. 10 shows the schematic diagram of the VAV system that is

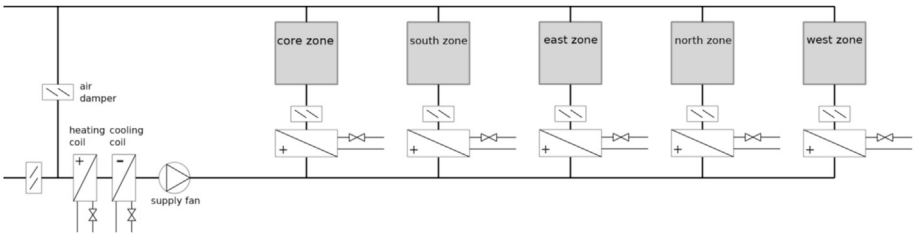

Fig. 10. Schematic diagram of the VAV system used in both cases.

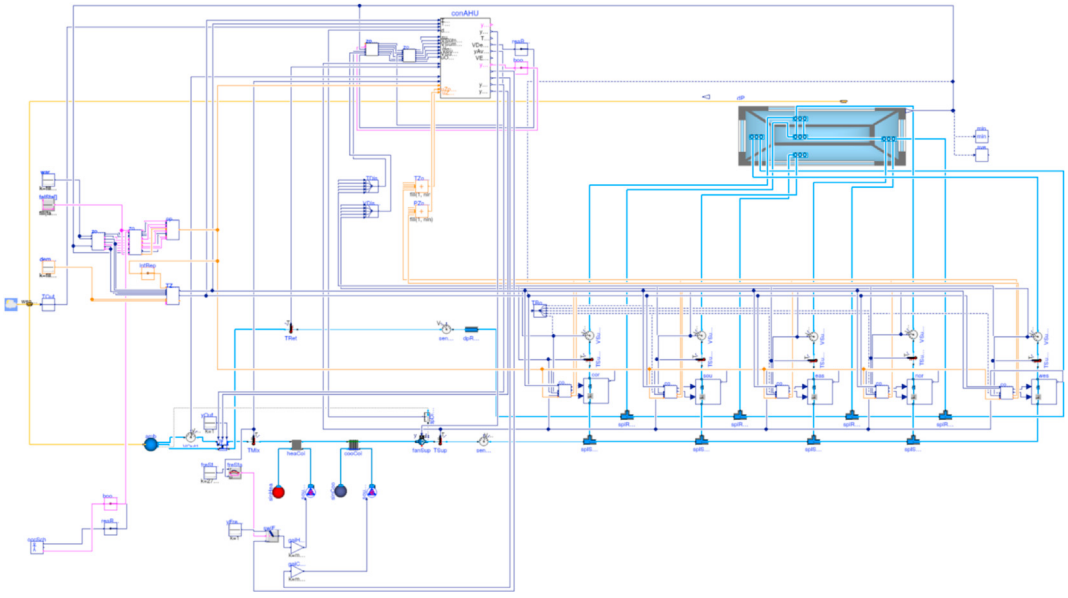

Fig. 11. Top level view of Modelica model for the Guideline 36 case.

used for both cases.

We will now summarize the modeling and simulation approach, compare the annual performance of the two control sequences and discuss the results. For a more detailed discussion, see Wetter et al. [36].

4.1.1. Modeling and simulation

All models are implemented using the Modelica Buildings Library and are available in version 8.0.0 (commit 9c680a6) of the library [34] as part of the package Buildings.Examples.VAVReheat. Fig. 11 shows the Guideline 36 model. In these models, the fan air flow rate is computed based on the fan speed input signal, the fan curve and the pressure distribution in the duct network [32]. The air flow rate in each leg of the flow network is computed based on friction, which depends on air flow rate, damper positions and wind pressure on the building. The air flow rate among the thermal zones is computed based on the pressure difference caused by wind pressure, HVAC operation and temperature differences in each of the thermal zones, which can induce bi-directional air flow through open doors [31]. This system of equations is coupled to the thermal models of the building envelope and the HVAC equipment, and solved at every time step. All simulations were run using OPTIMICA Compiler Toolkit (release tag r19089) on Ubuntu 18.04 64 bit. We used the CVode solver with a tolerance of 10^{-6}. This solver adaptively changes the time step to control the integration error and to properly simulate time events and state events.

Table 2 provides an overview of the model and simulation statistics. The differences in the number of variables and in the number of time varying variables indicate that the Guideline 36 control is significantly more detailed than the base case control sequence. To illustrate the complexity of the two control sequences, Fig. 12 shows the number of lines of the flat Modelica representation of Dymola 2021. The figure shows that the control sequence of Guideline 36 required about 6–7 times more code than the base case, and the size of the code required for the Guideline 36 controls logic is larger than the code needed to implement the HVAC system.

To conduct the simulations, it was essential that models were implemented carefully. Early model implementations experienced chattering due to frequent control switches, causing slow progress during part of the simulation, and large fan air flow rates and temperature raise as we used an idealized fan model that maintained set points for flow rates or head. The first issue was corrected through the use of hysteresis or timers for switches of continuous time controllers as one would need to use in real controllers. The second issue was corrected by using a fan model that uses speed as an input and removing a return fan that was not needed in view of the building static pressure and the low return duct friction. Robust models that do not experience these issues have been implemented in the library so they can be used in future experiments. For more experiences regarding proper model configuration based on a similar model, see Wetter et al. [36].

4.1.2. Results and discussions

Fig. 13 and Table 3 compare the annual site electricity use

Table 2
Model and simulation statistics.

Quantity	Base case	Guideline 36
Number of components	2929	5275
Number of variables (prior to translation)	34,129	44,026
Number of state variables	183	188
Number of continuous-time algebraic variables	3172	4008
Time for annual simulation in minutes	58	116

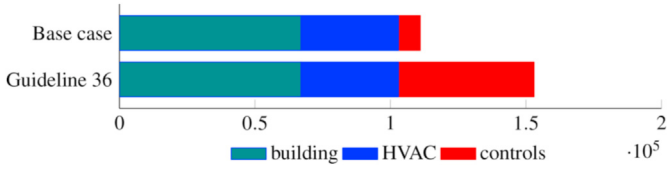

Fig. 12. Number of lines of code for building (including multi-zone air exchange and infiltration), HVAC system, and controls.

between the annual simulations with the base case control and the Guideline 36 control. To compute site electricity use, the heating load and cooling load at the coil was converted to electricity using a constant coefficient of performance of $COP_h = 4.0$ and $COP_c = 3.2$.

The simulations show a reduction in HVAC site electrical energy of 14 % yielded by the Guideline 36 control sequence.[2] To understand better where those savings stem from, Fig. 14 provides the density heat map and marginal histograms of the outdoor air flow rate and the supply air flow rate. The Guideline 36 sequence manages to operate the system consistently with both a lower air flow rate and a lower outdoor air fraction. The first effect is mainly due to the terminal unit control sequence. The base case uses a constant volume in heating demand, which is revealed on the heat map by the dense area around (0.4, 0.15), whereas the Guideline 36 sequence modulates the discharge air flow set point based on the actual heating demand. There are two reasons for the second effect pertaining to the outdoor air flow. First, the Guideline 36 sequence allows a lower minimum outdoor air flow rate when the terminal units supply air at a temperature below the room temperature,

[2] The results previously published in Wetter et al. [36] showed a significantly higher reduction of 30 %. This is due to the following three main changes, listed below by order of importance, that were applied to the models while investigating the features explaining the energy savings. *Minimum outdoor air flow control:* In the base case, the sizing of the minimum outdoor air flow rate set point was updated according to ASHRAE Standard 62.1, which led to reducing the outdoor air intake by more than 50 %. This had the highest impact on the savings. In the Guideline 36 case, the previous model had a high integral time of 1200 s in the PI loop that controls the outdoor air flow rate. With this setting, the integral term nearly vanishes, which essentially renders the PI controller as a P controller that had a significant steady-state error. For detailed closed loop models with complex control sequences, this illustrates the need for systematic testing of each control function, similar to the Functional Performance Testing performed on HVAC systems. The development of a tool chain that automates this verification process will serve both the simulation and the commissioning needs. *Room air temperature control:* In the base case, the box control logic was modified with an air flow set point in heating mode distinct from the minimum air flow set point. This is because the box control logic is not specified in Ref. [1] and the newly implemented sequence allows a more representative comparison with the Guideline 36 sequence. *Supply air temperature control:* In the base case, the control function was implemented from the following description: "The heating coil valve, mixed air dampers, and cooling coil valve shall modulate in sequence to maintain the supply air temperature set point" [1]. To represent this logic, the model had three different control loops, with distinct set point values. This was refactored to use a single control loop, whose output signal is mapped to sequence the three controlled devices, avoiding by principle, rather than by parameter setting, any unwanted simultaneous operation of these devices. This further illustrates the need for a formal description of the control logic in a way that eliminates the risk of diverging implementations which can increase the energy consumption if the building operator mistunes the parameter setting.

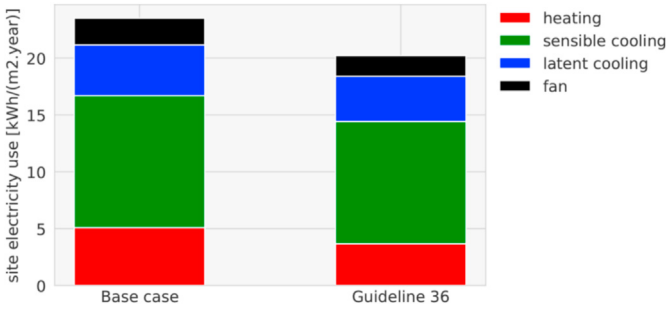

Fig. 13. Comparison of HVAC site energy use between the Guideline 36 and the base case control sequence.

leveraging the better air distribution effectiveness. Secondly, the supply air temperature reset prescribed by Guideline 36 triggers the economizer cooling mode less often than in the base case where the dense area around (0.2, 0.25) reveals an operating point near 100 % outdoor air with no cooling demand from the terminal units (that would result in an increased supply air flow rate). This directly affects the heating energy since the base case frequently supplies air at a cooler temperature than the one needed to balance the space loads, requiring additional terminal reheat. Fig. 15 illustrates the positive effect of the alternate sequence on the fan pressure head, which is reduced by both the modulation in sequence of the economizer dampers (as opposed to a modulation in tandem) and the trim and respond reset logic of the duct static pressure, which allows the VAV dampers to be operated closer to their full opening (as opposed to the most open damper being controlled with a set point strictly lower than the full opening, which is a requirement when using a PI control loop so that the sign of the tracking error can vary and the integral term can decrease).

Regarding the implementation effort that the two sequences require, the Guideline 36 stands out as challenging due to the complexity caused by the various mode changes, interlocks, timers and cascading control loops and large code size. This complexity makes verification of the correctness through inspection of the control signals difficult. As a consequence, various programming errors and misinterpretations or ambiguities of the Guideline 36 were only discovered in closed loop simulations, despite of having implemented open-loop test cases for each control block. We therefore believe it is important to provide robust, validated implementations and reference trajectories of the sequences published in Guideline 36. Such implementations would encapsulate the complexity of the sequences and provide assurances that energy modeler and control providers have correct implementations. With the implementation in the package Buildings.Controls.OBC.ASHRAE.G36_PR1, we made a start for such an implementation and laid out the structure and conventions, but have not covered all of the Guideline yet.

A key shortcoming from an implementer point of view was that the sequence was only available in English language, and as an implementation in WebCTRL, a commercial control product line from ALC (Automated Logic, a Carrier company), that was "close to the currently used version of the Guideline". Neither allowed a validation of the CDL implementation because the English language version leaves room for interpretation (and cannot be executed) and because EIKON simulation lacks the ability to easily inject externally generated input values that can be used for testing the dynamic response of control sequences for different input trajectories. Therefore, a benefit of the Modelica implementation is that such reference trajectories can now easily be generated to validate alternate implementations.

Table 3
Heating E_h, cooling E_c, fan E_f and total E_{tot} site energy, and savings of Guideline 36 case versus base case.

	E_h [$kWh/(m^2a)$]	E_c [$kWh/(m^2a)$]	E_f [$kWh/(m^2a)$]	E_{tot} [$kWh/(m^2a)$]	[%]
Base case	5.1	16.1	2.4	23.5	–
Guideline 36	3.7	14.7	1.8	20.2	14

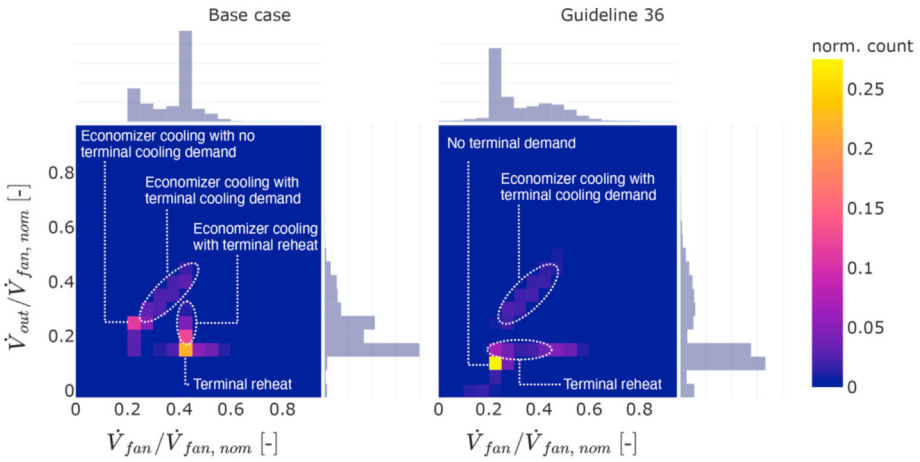

Fig. 14. Density heat map and marginal histograms of the outdoor air flow rate and the supply air flow rate, normalized by the nominal supply air flow rate.

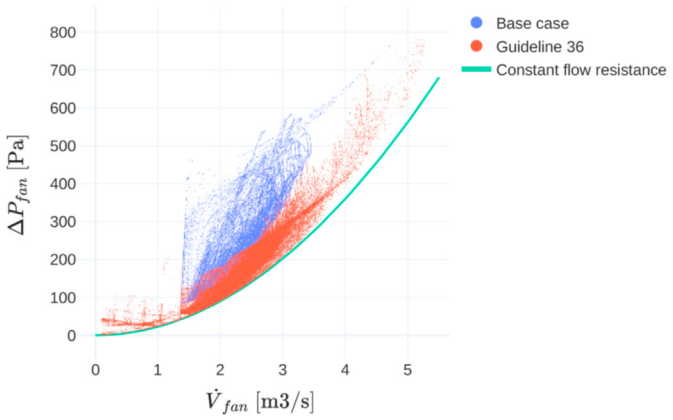

Fig. 15. Fan pressure rise and volume flow rate compared to an ideal system curve with constant flow resistance (VAV dampers fully open).

An additional benefit of the simulation based assessment was that it allowed detecting potential issues such as a mixed air temperature below the freezing point and chattering due to hard switches, which both led to improvements to Guideline 36.

While higher efficiency of the baseline may be achieved through supply air temperature reset or different economizer control, such potential improvements were only recognized after seeing the results of the Guideline 36 sequence. Thus, regardless of whether a building is using Guideline 36, having a baseline control against which alternative implementations can be compared and benchmarked is a valuable feature enabled by a library of standardized control sequences. Without a benchmark, one can easily claim to have a good control, while not recognizing what potential savings one may miss.

4.2. Code generation

In this section, we show an example translation of a control sequence for a VAV terminal unit controller of ASHRAE Guideline 36 from CDL to WebCTRL, a commercial control product line from ALC, using code generation. WebCTRL is based on a block diagram language called EIKON that has predefined elementary blocks, and a line programming language that can be used to add custom blocks. Translating from CDL to WebCTRL involves two steps. First, the JSON representation is generated from CDL using modelica-json as described in Section 3.5. Next, the *EIKON Add-on* shown in Fig. 16 that we have been developing is used to translate the JSON representation of the control sequence to EIKON code. As an input, the *EIKON Add-on* uses the JSON representation of the control sequence together with the *Rules* files. The *Rules* files specify the criteria for EIKON block selection, CDL to EIKON parameter mapping and conversion, and the OCL (Operators' Control Language) code for custom blocks. The *Translator* takes the JSON output of modelica-json together with the *Rules* files to create an *Instructions* JSON file. The *Instructions* file specifies which EIKON blocks to instantiate. It also specifies connection endpoints, and parameter override values. Composite control blocks that include a hierarchical structure of lower level control blocks are translated recursively. Once only elementary blocks are encountered, they are translated to EIKON code. For some blocks, a one-to-one mapping to native

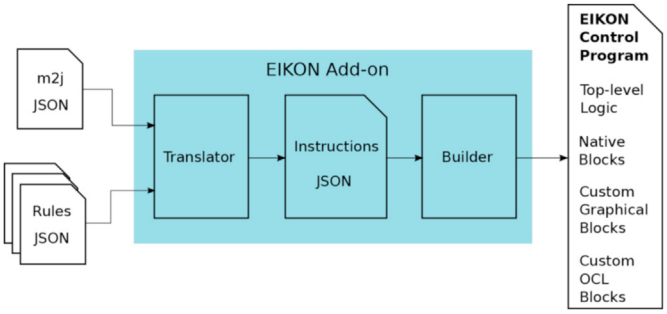

Fig. 16. Steps of translating JSON representation of the CDL sequence to EIKON sequence.

EIKON objects is performed. For example, for a logical "or"-block, a one-to-one mapping is possible. Other blocks require generation of OCL line programming code that is used by EIKON. For example, CDL has an adder that outputs $y = k_1\, u_1 + k_2\, u_2$ for gains k_1 and k_1 and inputs u_1 and u_2, whereas in EIKON, the native add-block outputs $y = u_1 + u_2$. For such cases, OCL line programming code is generated unless $k_1 = k_2 = 1$. Next, parameter values are assigned, and in some cases evaluated because CDL allows statements for parameter propagation that are not supported by EIKON. The complete *Instructions* JSON file is then input to the *Builder*, which creates the EIKON control program. In the *Builder*, components are laid out graphically, and inputs and outputs are connected and laid out graphically. While CDL declares the graphical layout, the position of the I/O on the mapped EIKON blocks may be different. Therefore the connections are rerouted in this step. All these operations are specific to the control target platform, and some may require proprietary knowledge of the control target platform. Therefore, the *Builder* portion of the process is proprietary. For ease of use, a single add-on to EIKON bundles together the *Translator* and *Builder* portions into one action so that the instructions can remain internal and are not written out as a file. Fig. 17 shows the block diagram view of the VAV terminal unit in CDL at the top of the figure and its generated EIKON control program below it. Once translated, the EIKON implementation need not know anything about CDL.

4.3. Verification of installed sequences

We will now demonstrate an example in which we verified conformance to a CDL specification of an installed control sequence. This uses the workflow described in Section 3.6. We used the controller OptiFlex G5CE from Automated Logic, which has been loaded with a set of control sequences programmed for a single zone VAV system. The sequence controls the fan speed based on the cooling control signal, heating control signal, zone temperature setpoint, measured zone temperature, outdoor air temperature and supply air fan status. The verification tool communicates with the controller using the BACnet/IP protocol. For this benchtop test, we allowed all BACnet points on the controller to be read/write over the network.

In the Modelica Buildings Library, this sequence is implemented in the model Buildings.Controls.OBC.ASHRAE.G36_PR1.AHUs.SingleZone.VAV.SetPoints.Supply, which we used as the reference implementation. We created a 1 h validation test. Fig. 18 shows the configuration file that specifies the validation test, the control sequence, the output signals and their tolerances, and the controller's communication settings. Fig. 19 shows a snippet of the point and unit mapping between the CDL points and the corresponding BACnet points on the controller. The verification tool takes these input files and identifies the input and output BACnet points for the sequence under test. Then, communicating with the controller via the BACnet writer and reader, it saves the trended control points. Next, the tool uses the trended input points from the controller and runs the CDL simulation using OpenModelica to create the reference trajectories. As the last step, it invokes the sequence verification, which will produce the charts that compare reference and trended outputs for each variable. Fig. 20 shows such a chart produced for the fan speed set point of the verified sequence.

Fig. 20 shows that around $t = 250$ s, the trended signal is outside the tolerance of the reference signal. As this control sequence has neither continuous nor discrete state variables, this suggests a difference in the implementation of the sequence or its configuration. As a next step, the control logic and its configuration parameters would need to be inspected to identify the root cause for this difference. For this controller, part of the difference around $t = 250$ may originate from fact that the implementation in the Automated Logic controller computes internally a heating and cooling control signal using a PI controller based on zone air temperature set point and measurement, whereas in the CDL implementation, this control signal is an input to the sequence that is computed in a different module. This difference in control input and output variables shows that verifying control logic conformance to ASHRAE Guideline 36 would be facilitated if the Guideline were to be explicit about the input and output variables of the individual control sequences.

4.4. Verification as part of product line development

We will now demonstrate an example in which we verified a control sequence that is installed on a controller using the process described in Section 3.7. For this example, we used the controller OptiFlex G5CE controller from Automated Logic. The controller has a sequence loaded that computes the active set point temperature of a VAV terminal unit from ASHRAE Guideline 36. We will verify this sequence against the CDL specification Buildings.Controls.OBC.ASHRAE.G36_PR1.TerminalUnits.SetPoints from the Modelica Buildings Library. The configuration files for the test configuration and the point mapping have the same structure as shown in Figs. 18 and 19 and hence are not reproduced here for this controller configuration. These two items are the inputs to the BACnet writer, together with the CSV control input file, as shown in the flow chart in Fig. 9. The verification tool takes these input files, communicates with the controller via the BACnet writer and reader, and outputs the trended control output file. As the last step, it invokes the sequence verification, which will produce the charts that compare reference and trended output for each variable. Fig. 21 shows a magnified section of such a chart for the verification of the active heating set point that is output of the sequence. For this illustration, we introduced an error near $t = 60$ s to illustrate how the result of a failed test is displayed.

5. Discussion

To our best knowledge, this is the first implementation of a set of modular, but interdependent, computing tools that pave the path towards digitization of the controls development, specification, delivery and verification, under the major technical constraint of reusing existing building control product lines. This workflow is possible because declarative modeling languages and system simulation technologies sufficiently advanced over the past years. They now allow representation of actual control sequences in a declarative way, simulation-based assessment of their energy and comfort performance using an energy model in the loop, and translation of the control sequence to other computer languages using machine-to-machine translation. The significance of such a digitized workflow is that it allows performance assessment early on during design to improve system-level performance, non-ambiguous specification of the control sequence to avoid ambiguity for the controls contractor, automated translation to control product lines to reduce engineering time and avoid coding errors, and formal verification relative to the specification from the design phase to ensure correctness of the implementation. Moreover, best-in-class control sequences can be shared through reusable libraries, and these sequences can be continuously improved and deployed through such libraries to industry.

While we demonstrated such a machine-to-machine translation using one commercial control product line, the intermediate exchange format is vendor independent and has not been designed specifically for this target platform. Translation of this intermediate format to other control product lines is currently examined through

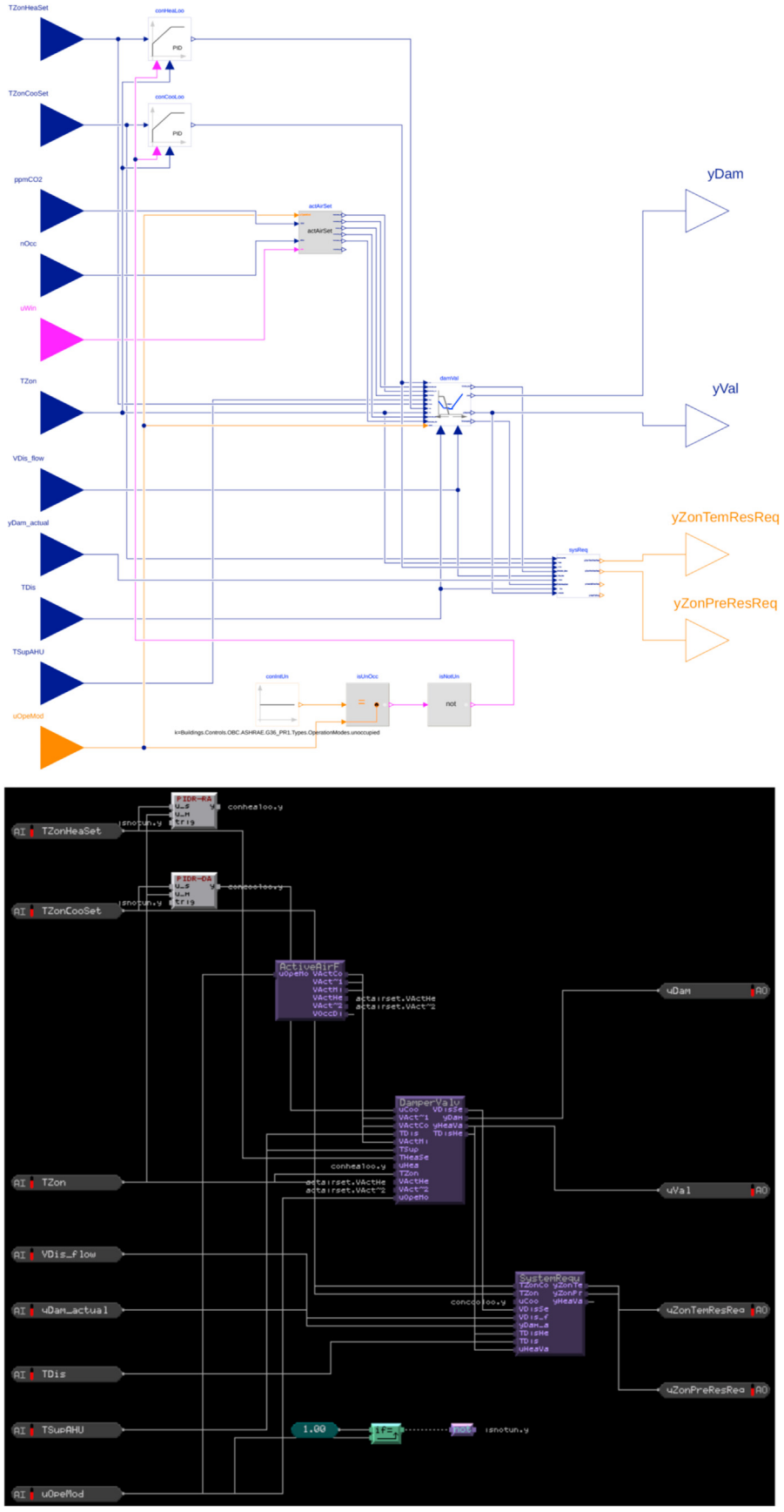

Fig. 17. VAV terminal unit controller that is part of the control sequence which has been translated from CDL (top) to EIKON (bottom) for use in ALC WebCTRL.

```
{
 "references": [
   {
     "model"           : "TestVAVSequence",
     "generateJson"    : false,
     "sequence"        : "setPoiVAV",
     "pointNameMapping": "realControllerPointMapping.json",
     "runController"   : false,
     "controllerOutput": "test/real_outputs.csv",
    }
 ],
 "modelJsonDirectory": "test",
 "tolerances": { "rtolx": 0.002, "rtoly": 0.002, "atolx": 10, "atoly": 0 },
 "sampling": 120,
 "controller": {
   "networkAddress": "192.168.0.115/24",
   "deviceAddress": "192.168.0.227",
   "deviceId": 240001
 }
}
```

Fig. 18. Configuration file used by the sequence verification tool.

```
[
  {
    "cdl":    {"name": "uHea",                                  "type": "float"},
    "device": {"name": "htg_pct_1",                            "type": "float"}
  },
  {
    "cdl":    {"name": "uCoo",                                  "type": "float"},
    "device": {"name": "clg_pct_1",                            "type": "float"}
  },
  {
    "cdl":    {"name": "TZonSet",              "unit": "K",     "type": "float"},
    "device": {"name": "Zone Temp Setpoint_1", "unit": "degF", "type": "float"}
  },
  ...
  {
    "cdl":    {"name": "y",                                     "type": "float"},
    "device": {"name": "vspt_1",                               "type": "float"}
  }
]
```

Fig. 19. Snippet of an example point and unit mapping file between the CDL variable and the real controller point.

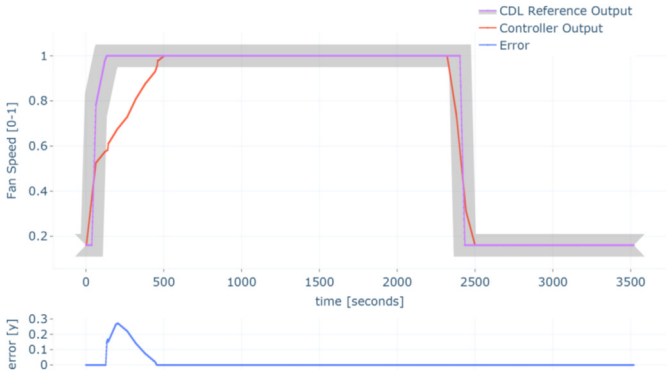

Fig. 20. Illustration of the output chart generated by the sequence verification tool for a zone heating set point. The shaded region is the region of acceptance, and the blue points show the error between reference and actual control output. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

an ongoing standardization process within ASHRAE Standard 231P "CDL - A Control Description Language for Building Environmental Control Sequences", complementing BACnet for communication, Guideline 36 for natural language documentation of control sequences, and ASHRAE Standard 223P for semantic modeling. To facilitate adoption by the building controls industry who is not expected to migrate any time soon to new product lines, the intermediate exchange format was designed to be a common denominator across control product lines. This resulted in limited functionality compared to what is available in products for model-based design that include controller code generation, typically via C code. For example, state machines that would be convenient to express control sequences and that modern simulation

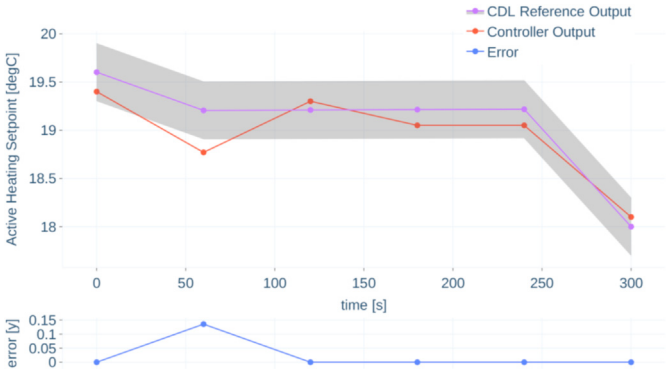

Fig. 21. Illustration of the output chart generated by the sequence verification tool for a zone heating set point. The shaded region is the region of acceptance, and the blue points show the error between reference and actual control output. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

environments generally support are currently not allowed in our control represenation because building control companies expressed to the authors that they cannot support state machines in their product lines.

6. Conclusion

Properly designed conventional building control sequences can significantly reduce energy consumption. However, correct implementation of such sequences is difficult and time-consuming. To digitize this process, we developed a Control Description Language (CDL) that allows expressing building control sequences in a digital, machine readable language. We showed that CDL is suitable for (i) documentation of controls intent in English language and in machine-readable digital format, (ii) closed loop simulation for performance assessment, (iii) code generation to commercial control product lines, and (iv) formal verification of installed control sequences.

By providing ready-to-use libraries of control sequences in CDL that can be configured to a particular building, as demonstrated with our implementation, the complexity of control implementation can be hidden from the end-user. Therefore, a library of carefully designed and implemented control sequences has the potential to substantially reduce energy consumption. We demonstrated that with CDL-conforming control sequences, coupled to whole building energy models, these energy savings can be quantified, allowing to compare different control strategies. In our case study, annual simulations revealed the main underlying reasons for the energy savings, and they revealed potential issues which helped improving the sequence itself. However, for the numerical simulations to be efficient and robust, the control sequences need to be properly implemented. For example, for continuous time control, all switches need hysteresis or a time delay, and the solver needs to be able to handle time and state events.

We also showed that CDL control sequences can be translated to a commercial control product line using code generation. This promises to significantly reduce effort for programming of control sequences, and it allows elimination of programming errors when implementing the control specification. Finally, we introduced a process and supporting tools for the formal verification of building control sequences relative to their digital specification in CDL, thereby providing an end-to-end quality control. To standardize this process, the here presented work initiated the development of ASHRAE Standard 231P, which aims to standardize the digital

representation of control logic, thereby complementing standards for communication and for semantic modeling.

In future work, we will expand a library of control sequences. We will also further develop a tool chain to

1. enable mechanical designers to create specifications based on the simulation model,
2. allow control providers to translate CDL-conforming control sequences to their product line, and
3. allow the commissioning provider to verify correct implementation by executing the control model with inputs and parameters obtained from the real implementation, and comparing the simulated with the real actuator signals.

We also recommend implementing Guideline 36 using the CDL language. This would allow a non-ambiguous, executable specification against which vendor-specific implementations could be tested and certified using free, open-source tools.

Credit author statement

Michael Wetter, Principal Investigator, Conceptualization, and Software development. Paul Ehrlich, Conceptualization. Antoine Gautier, Conceptualization, and Software development. Milica Grahovac, Conceptualization, and Software development. Philip Haves, Conceptualization. Jianjun Hu, Conceptualization and Software development. Anand Prakash, Software. Dave Robin, Conceptualization and Software development. Kun Zhang, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

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, and the California Energy Commission's Electric Program Investment Charge (EPIC) Program.

We like to thank the following people and organization that contributed through the OpenBuildingControl project to this work: Troy R. Maeder (Automated Logic); Caleb Clough, David Pritchard, Amy Shen, Paul Switenki (ARUP); Brent Eubanks (Carbon Lighthouse); Brian Turner (ControlCo); Rick Stehmeyer (Cx Associates); Jay Santos, Jamie Nickels (Facility Dynamics); Mark Hydeman (Google); John Bruschi, Dave Guerrant, Andrea Traber, John Nelson, Fiona Woods (Integral Group); Jonathan Schoenfeld (Kodaro); Jim Kelsey (kW Engineering); David H. Blum, Janie Page, Mary Ann Piette, Marco Pritoni, Lisa Rivalin (Lawrence Berkeley National Laboratory); Francisco Ruiz, Rich Rockwood (Oracle); Yan Chen, Karthikeya Devaprasad (Pacific Northwest National Laboratory); Gerry Hamilton (Stanford Facilities Energy Management); and Hwakong Cheng, Brandon Gill, Reece Kiriu, Steven T. Taylor (Taylor Engineering).

References

[1] ASHRAE. Sequences of operation for common HVAC systems. Atlanta, GA: ASHRAE; 2006.
[2] ASHRAE. Guideline 36-2018, high-performance sequences of operation for HVAC systems. Atlanta, GA: ASHRAE; June 2018.
[3] Balaji B, Bhattacharya A, Fierro G, Gao J, Gluck J, Hong D, Johansen A, Koh J, Ploennigs J, Agarwal Y, Bergés M, Culler D, Gupta RK, Kjærgaard MB, Srivastava M, Whitehouse K. Brick: metadata schema for portable smart building applications. Appl Energy 2018;226:1273–92. https://doi.org/10.1016/j.apenergy.2018.02.091. ISSN 0306-2619, http://www.sciencedirect.com/science/article/pii/S0306261918302162.
[4] Barwig FE, House JM, Klaassen CJ, Ardehali MM, Smith TF. The national building controls information program. In: Summer study on energy efficiency in buildings. In: Pacific Grove, CA; Aug. 2002. http://aceee.org/files/proceedings/2002/data/papers/SS02_Panel3_Paper01.pdf.
[5] Bonvini M, Leva A. A Modelica library for industrial control systems. In: Proc. of the 9-th international modelica conference. Munich, Germany: Modelica Association; Sep. 2012. p. 477–84. https://doi.org/10.3384/ecp12076477.
[6] Broman D, Greenberg L, Lee EA, Massin M, Tripakis S, Wetter M. Requirements for hybrid cosimulation standards. In: 18th international conference on Hybrid systems: computation and control. ACM press; Apr. 2015. http://simulationresearch.lbl.gov/wetter/download/2015-BromanEtAl_Hybrid-Cosimulation_HSCC.pdf.
[7] Deru M, Field K, Studer D, Benne K, Griffith B, Torcellini P, Liu B, Halverson M, Winiarski D, Rosenberg M, Yazdanian M, Huang J, Crawley D. U.S. Department of Energy commercial reference building models of the national building stock. Technical Report NREL/TP February 2011:5500–46861. National Renewables Energy Laboratory, 1617 Cole Boulevard, Golden, Colorado 80401.
[8] Donida F, Leva A. Modelica as a host language for process/control co–simulation and co–design. In: Bachmann B, editor. Proc. of the 6-th international modelica conference. Bielefeld, Germany: Modelica Association and University of Applied Sciences Bielefeld; Mar. 2008. p. 401–8.
[9] Effinger J, Friedman H, Morales C, Sibley E, Tingey S. A study on energy savings and measure cost effectiveness of existing building commissioning. Technical report, IEA Annex 2009;47.
[10] Emphysis. Functional Mock-up Interface for Embedded systems (eFMI) version 1.0.0-alpha.3 (draft)vol. 2021; January 27, 2021. https://emphysis.github.io/pages/downloads/efmi_specification_1.0.0-alpha.3.html.
[11] Fernandez N, Xie Y, Katipamula S, Zhao M, Wang W, Corbin C. Impacts of commercial building controls on energy savings and peak load reduction. Technical Report 25985, PNNL May 2017. https://buildingretuning.pnnl.gov/publications/PNNL-25985.pdf.
[12] Fierro G, Pritoni M, Abdelbaky M, Lengyel D, Leyden J, Prakash AK, Gupta P, Raftery P, Peffer T, Thomson G, Culler DE. Mortar: an open testbed for portable building analytics. ACM Trans Sens Netw 2019;16(7):1–7. https://doi.org/10.1145/3366375. 31.
[13] Fierro G, Prakash AK, Mosiman C, Pritoni M, Raftery P, Wetter M, et al. Shepherding metadata through the building lifecycle. In: Proceedings of the 7th ACM international conference on systems for energy-efficient buildings, cities, and transportation, BuildSys '20, pages. New York, NY, USA: Association for Computing Machinery; 2020. p. 70–9. ISBN 9781450380614, https://doi.org/10.1145/3408308.3427627.
[14] Fritzson P, Pop A, Abdelhak K, Ashgar A, Bachmann B, Braun W, et al. The OpenModelica integrated environment for modeling, simulation, and model-based development. Model Identif Control 2020;41(4):241–95. https://doi.org/10.4173/mic.2020.4.1.
[15] Gnerre B, Fuller K. When building controls veer off course. In *Facility Executive*, volume 462707, page 32. Group C Media, Inc., Tinton Falls, NY Dec. 2017. https://facilityexecutive.com/2017/12/building-automation-veers-off-course/.
[16] Husaunndee A, Lahrech R, Vaezi-Nejad H, Visier J. Simbad: a simulation toolbox for the design and test of HVAC control systems. In: Roux JJ, Woloszyn M, editors. *Proc. Of the 5-th IBPSA conference*, pages 269–276; 1997. www.ibpsa.org/proceedings/bs1997/bs97_p022.pdf.
[17] IEC 61131-10. Programmable controllers – Part 10: PLC open XML exchange format. Apr. 2019.
[18] IEC 61131-3. Programmable controllers – Part 3: programming languages. Feb. 2013.
[19] ISO 16484-1. Building automation and control systems (BACS) – Part 1: project specification and implementation. ISO 2010;16484–1 (E), Nov. 2010.
[20] ISO 16484-3. Building automation and control systems (BACS) – Part 3: Functions. ISO Nov. 2007;16484–3:2005.
[21] LBNL. Funnel software to compare time series within user-specified tolerances. 2021. https://github.com/lbl-srg/funnel.
[22] LBNL. Modelica to JSON parser. https://github.com/lbl-srg/modelica-json; 2021.
[23] Mattsson SE, Elmqvist H. Modelica – an international effort to design the next generation modeling language. In: Boullart L, Loccufier M, Mattsson SE, editors. *7th IFAC Symposium on computer aided control systems design*, pages 1–5, gent, Belgium; Apr. 1997. http://www.modelica.org/publications/papers/CACSD97Modelica.pdf.
[24] Mills E. Building commissioning: a golden opportunity for reducing energy costs and greenhouse gas emissions in the United States. Energy Effic 2011;4:145–73. https://doi.org/10.1007/s12053-011-9116-8.
[26] Modelica. Modelica – a unified object-oriented language for systems modeling, language specification, version 3.4. Modelica Association Apr. 2017. https://www.modelica.org/documents/ModelicaSpec34.pdf.
[27] MODELISAR Consortium. Functional Mock-up Interface for model-exchange and Co-simulation version 2.0. 2014. https://www.fmi-standard.org/downloads.
[28] Otter M, Årzén K-E, Dressler I. StateGraph – a modelica library for hierarchical state machines. In: Schmitz G, editor. Proceedings of the 4th modelica

conference. Hamburg, Germany: Modelica Association and Hamburg University of Technology; Mar. 2005. http://www.modelica.org/events/Conference2005.

[29] Schneider GF, Pauwels P, Steiger S. Ontology-based modeling of control logic in building automation systems. IEEE Trans Ind Inf 2017;13(6):3350–60. https://doi.org/10.1109/TII.2017. 2743221.

[30] Schneider GF, Peßler GA, Steiger S. Modelling and simulation of standardised control functions from building automation. In: Schmitz G, editor. In Proc. of the 12-th international modelica conference, pages 209–218, prague. Czech republic: Modelica association; May 2017. https://doi.org/10.3384/ecp17132209.

[31] Wetter M. Multizone airflow model in Modelica. In: Kral C, Haumer A, editors. *Proc. Of the 5-th international modelica conference*, volume 2, pages 431–440. Vienna: Austria; 2006. Modelica Association and Arsenal Research, http://www.modelica.org/events/modelica2006/Proceedings/sessions/Session413.pdf.

[32] Wetter M. Fan and pump model that has a unique solution for any pressure boundary condition and control signal. In: Roux JJ, Woloszyn M, editors. *Proc. Of the 13-th IBPSA conference*, pages 3505–3512; 2013. http://simulationresearch.lbl.gov/wetter/download/2013-IBPSA-Wetter.pdf.

[33] Wetter M, Zuo W, Nouidui TS. Modeling of heat transfer in rooms in the Modelica "Buildings" library. In *Proc. of the 12-th IBPSA Conference*, pages 1096–1103. International Building Performance Simulation Association Nov. 2011. http://www.ibpsa.org/.

[34] Wetter M, Zuo W, Nouidui TS, Pang X. Modelica buildings library. J Build Perform Simulat 2014;7(4):253–70. https://doi.org/10.1080/19401493.2013.765506.

[35] Wetter M, Grahovac M, Hu J. Control Description Language. In: 1st American modelica conference. MA, USA: Cambridge; Aug. 2018. https://doi.org/10.3384/ecp.1815417.

[36] Wetter M, Hu J, Grahovac M, Eubanks B, Haves P. OpenBuildingControl: modeling feedback control as a step towards formal design, specification, deployment and verification of building control sequences. In: Proc. of building performance modeling conference and SimBuild, pages. Chicago, IL, USA: Sept; 2018. p. 775–82. https://simulationresearch.lbl.gov/wetter/download/2018-simBuild-OpenBuildingControl.pdf.

[37] Wetter M, Gautier A, Grahovac M, Hu J. Verification of control sequences within OpenBuildingControl. In: 16-th IBPSA conference. International building performance simulation association, sept; 2019. http://simulationresearch.lbl.gov/wetter/download/2019-ibpsa-OpenBuildingControl.pdf.

[38] Wetter M, Benne K, Gautier A, Nouidui TS, Ramle A, Roth A, Tummescheit H, Mentzer S, Winther C. Lifting the garage door on Spawn, an open-source BEM-controls engine. Proc. of Building Performance Modeling Conference and SimBuild, Chicago, IL, USA 2020. https://simulationresearch.lbl.gov/wetter/download/2020-simBuild-spawn.pdf.

[39] Yang Y, Pinto A, Sangiovanni-Vincentelli A, Zhu Q. A design flow for building automation and control systems. In: 2010 31st IEEE Real-Time Systems Symposium, pages 105–116; Nov. 2010. p. 26. https://doi.org/10.1109/RTSS.2010.26.