Modelica-json: Transforming energy models to digitize the control delivery process

Michael Wetter[1], Jianjun Hu[1], Anand Prakash[1], Paul Ehrlich[2], Gabe Fierro[3], Milica Grahovac[1], Marco Pritoni[1], Lisa Rivalin[4*], Dave Robin[5]

[1]Lawrence Berkeley National Laboratory, Berkeley, CA
[2]Building Intelligence Group, Portland, OR
[3]Gabe Fierro, University of California at Berkeley, Berkeley, CA
[4]Facebook, Menlo Park, CA
[5]BSC Softworks, Marietta, GA

Abstract

Building simulation models are typically not used to generate the documentation required for bidding and project delivery of commercial building systems, or for their semantic modeling and commissioning. This paper presents a software tool that aids in digitizing the control delivery process, spanning simulation during design to implementation and formal verification during commissioning. The tool can generate from Modelica models digital documentation of control sequences. This digital documentation, along with other project drawings and specifications can be used for project bidding. It can also be used for implementation of control sequences through machine-to-machine translation to commercial legacy control products, for which we are currently developing the proposed ASHRAE Standard 231P based on the presented work. Moreover, as-installed control sequences can be formally verified against the design specification, and a semantic model in Brick can be exported to aid in configuration of building analytics and fault detection. The paper presents what we believe is the first translation of a Modelica-implemented control sequence to a native implementation on a commercial control platform, using the webCTRL product line from Automated Logic. The paper also shows how a webCTRL implementation can be formally verified against its Modelica specification. These use cases have all been demonstrated with a prototype implementation that is now being further developed.

Key Innovations

- Prototyped code for end-to-end digital control delivery process with formal verification of as-installed control sequences.
- Reuse of simulation models for bidding and implementation of control sequences, documentation of sequences, semantic modeling and formal verification of sequences during commissioning.

Practical Implications

The presented tool, and the digital language that the tool is based on, provides a foundation to digitize the currently manual, error-prone, paper-based control delivery process. Through the accompanying proposed ASHRAE Standard 231P, this will bridge simulation with operation, allowing performance testing of control sequences during design, export of control documentation, and implementation on commercial building control platforms. In view of the increased complexity of control sequences and the current failure to deliver robust high-performance control sequences at scale, this should not only reduce cost and time for control delivery, but also lead to robust implementation of control sequences at the scale needed to significantly reduce energy use of buildings.

Introduction

Optimized HVAC control sequences are critical to achieving high performance buildings. Use of high performance control sequences has shown to reduce energy consumption, with a range of 20% to 30% being common (Fernandez et al., 2017). However, software programming errors have been reported to be the subcategory of control-related problems that have the largest energy impact (Barwig et al., 2002). Moreover, the complexity of control sequences is increasing: ASHRAE published control sequences for variable air volume flow (VAV) systems in 2006 and in 2016, with the higher-performing sequence from 2016 requiring close to one order of magnitude more code, as measured by an implementation in the Modelica Buildings Library (ASHRAE, 2006, 2018; Wetter et al., 2014). Moreover, as new sequences also need to provide load flexibility to the electrical grid, such as through integration of Model Predictive Control (Drgoňa et al., 2020), their complexity is likely to increase even further.

Today's building control delivery process is largely based on a verbose natural language specification of the control sequences. The process has not changed

*This work was conducted while the author was an employee of Lawrence Berkeley National Laboratory.

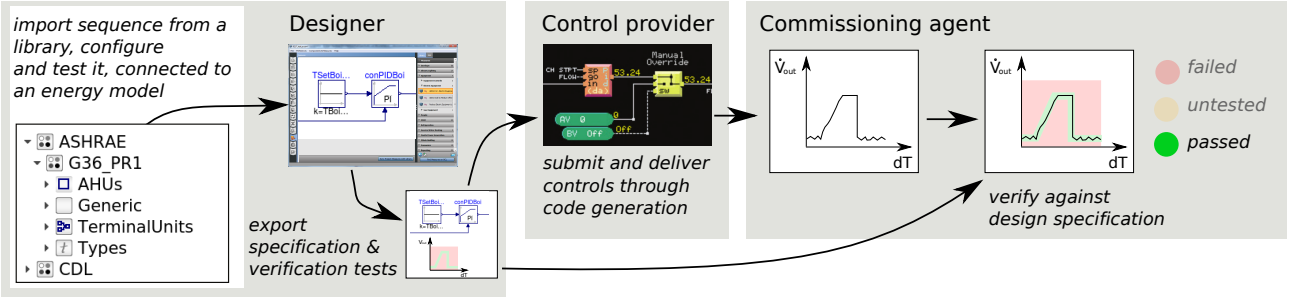


Figure 1: Overview of process for control sequence design, export of a specification, implementation on a control platform and verification against the specification.

much over the past several decades, and often does not produce robust implementations even for the simpler sequences from 20 years ago that were common when Barwig et al. (2002) conducted their study. Therefore, in view of the control complexity that we cannot tame with today's paper-based process, the OpenBuildingControl (OBC) project works on digitizing the control delivery process. The project addresses the need to deploy high performance, grid-flexible control sequences at scale, taking advantage of the fact that control sequences are, by their very nature, expressible as code.

OBC develops a process and tools that allow to

1. use and customize control sequences from libraries,
2. test and contrast their performance using annual energy modeling with Spawn of EnergyPlus (Wetter et al., 2020),
3. export natural language sequence description for documentation of the control intent and for instructing the building operator,
4. export point lists for bidding,
5. export a control-vendor neutral representation of the sequence for implementation on building control product lines through machine-to-machine translation,
6. export of semantic models for use with portable building analytics software (Fierro et al., 2020, 2019), and
7. export of specifications for semi-automated commissioning of as-installed sequences (Wetter et al., 2019).

These artifacts will be produced through the translation of control sequences that are implemented using the Control Description Language (CDL), which is a subset of the Modelica language that we introduced in Wetter et al. (2018). This translation is done using the `modelica-json` software that is introduced in this paper.

This paper presents the `modelica-json` software, describes its implementation and explains the different formats into which control sequences can be translated. The paper also shows an example of a machine-to-machine translation of CDL sequences to a commercial building control product line, an example of exporting a semantic model to the Brick Schema representation (Balaji et al., 2016), and the verification of a control sequence as-implemented in hardware relative to its CDL reference implementation.

Workflow

We will now explain the digital control delivery workflow for which we have been developing various software, including the here introduced `modelica-json` translator. The workflow is applicable to new buildings or to retrofits that will receive new control sequences. Figure 1 shows a high-level view of the workflow. Starting from the left of the figure, building energy tool developers will provide libraries with customizable control sequences. These sequences are implemented in CDL. Sequences from these libraries can be customized by energy modelers or controls designers. Also, energy modelers or controls designers can extend this collection of libraries, for example to add sequences that are commonly used by their company. Using a control sequence selection and configuration tool that is being developed for mechanical designers, or using any general-purpose Modelica modeling environment, a mechanical designer can select and customize the control sequence. Using building simulation, the mechanical designer can improve the closed loop performance of the sequence, verify its correct operation for the particular building, compare the performance among different control sequences or sequence configurations and ultimately select the sequences to be used. Next, the mechanical designer exports a digital specification of the sequences, together with verification tests. The control contractor then uses this specification, along with other documents provided for the project, to develop their pricing proposal. The control contractor can use the CDL directly in their product line (if compatible) or alternatively translate the sequence from CDL to their proprietary control platform through code generation. The commissioning agent will take the digital specification and the verification tests to formally test whether the as-installed sequences produce a response that is within specified bounds of the response of the

original specification. Optionally, and largely subject of ongoing work, the designer or controls contractor may also output a semantic model. This can then aid in configuring the control logic to actual equipment, in setting up point-mapping for the commissioning using a process as described in Wetter et al. (2019), and in bootstrapping self-learning elements of the control, building analytics and system-level fault detection and diagnostic methods. For example, if a module of a supervisory set point scheduler requires knowledge of the system dynamics that is acquired through machine learning, then this module could be initially trained on the simulation model from the design phase, the module's I/O can be abstracted through the semantic model, and then the module can be semi-automatically ported from the training environment to the actual building control system if this is also embodied with a semantic model.

There are also a few related workflows for other stakeholders. For example, professional organizations such as ASHRAE may choose to provide reference implementation of control sequences, such as for ASHRAE Guideline 36 (ASHRAE, 2018), in CDL. These can then be tested for correctness and for closed-loop performance, translated by control providers to their product lines, and used by energy modelers during building design. Code setting bodies such as the California Energy Commission may choose to not only provide in natural language certain control requirements, but also provide a CDL implementation for use by designers in energy simulation, by control providers for implementation in their product lines, and by code officials for code compliance checks.

All these workflows require translation of control sequences from CDL to other representations. For this purpose, we have been developing the `modelica-json` software that is introduced in the next section.

Modelica-JSON software

The `modelica-json` software consist of two elements that are distributed as one application: A main application that is implemented using the JavaScript runtime environment `Node.js`, and a Modelica parser that uses Java and the Antlr package (Parr, 2012). The software works on Windows, Linux and macOS.

The two elements of the software are as follows: The main application takes via command line arguments the Modelica filename and the output format specification. It then invokes the Java component to translate the Modelica files into an internal JSON format that we refer to here as raw-JSON. Because this raw-JSON representation is very verbose and hard to parse, the main application simplifies it into a more compact JSON representation that is easier to work with for downstream applications. Depending on the user-provided command line arguments, this compact JSON representation is written to disk, or it is further processed by the main application to generate other output formats such as HTML, SVG diagrams or MS Word. The user can also specify two modes via the command line arguments: A Modelica mode and a CDL mode. In the Modelica mode, a usual Modelica model can be translated, whereas in the CDL mode, the translator also checks the provided model for conformance with the CDL specification.

The second element of the software is the Java component that is automatically invoked by the main application. The Java component uses the Antrl package to translate Modelica into the raw-JSON format. Using the Antlr lexical grammar file developed for the Modelica language, it generates an abstract syntax tree (AST) for the Modelica file. Next, it visits each node of this AST and processes it to produce the verbose raw-JSON output. This output is then further processed with the main application as described above. The Java component is compiled into a Java Archive file that is distributed with `modelica-json`. It only needs to be updated when a developer makes changes to the parser. Future work may convert this part of the tool from Java to JavaScript.

Output formats

This section discusses the different output formats that can be generated with `modelica-json`.

All output formats are generated by running

```
node modelica-json/app.js \
  -m cdl -f fileName -o format
```

where `fileName` is the name of the Modelica file that is the top-level controller, and `format` is the output format, which can either be `json`, `html`, `docx` or `svg`. For export of a Brick model, the `json` output is used together with the JSON to Brick translator.[1]

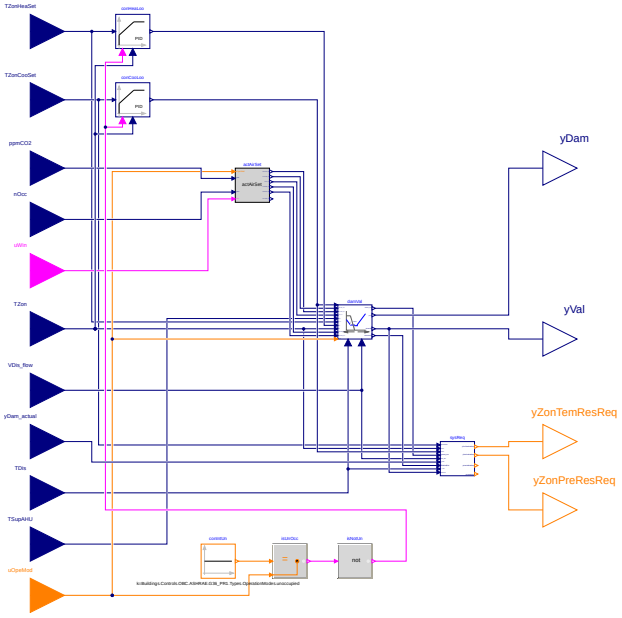
JSON

The JSON output serves as representation from which the output formats HTML, SVG, MS Word, Brick and translation to proprietary control programming languages such as ALC webCTRL are generated. In CDL mode, with the `json` output option, `modelica-json` will create one JSON file of the controller. This file also includes the information of the control blocks that are used to hierarchically compose the top-level controller. The file has all information required to translate the controller to downstream applications.

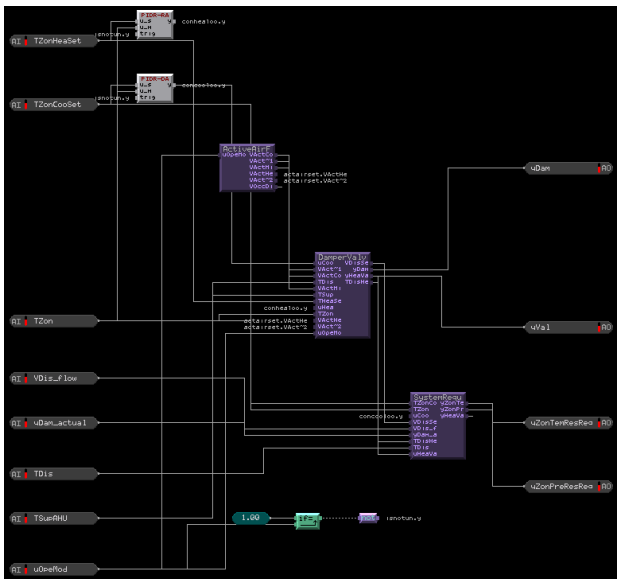
There is also a schema file available that can be used to validate the JSON files of control sequences that were generated from CDL.

Compared to the CDL representation, the JSON representation is significantly larger. For example, consider the VAV terminal unit controller of ASHRAE Guideline 36 that is im-

[1]The JSON to Brick translator is available at `https://github.com/gtfierro/shepherding-metadata`.



(a) Schematic view of the VAV terminal unit controller generated by `modelica-json` *based on the Modelica model.*



(b) Implementation in ALC webCTRL of the controller shown above as translated by `modelica-json`*. The graphic is the visual representation of the EIKON software.*

Figure 2: VAV terminal unit controller that is part of the control sequence which has been translated from CDL to EIKON for use in ALC webCTRL.

plemented in `Buildings.Controls.OBC.ASHRAE.G36_PR1.TerminalUnits.Controller` in the Modelica Buildings Library. Figure 2a shows the top-level block diagram of the controller. The controller is composed hierarchically and consists of 42 Modelica blocks, of which 38 are elementary CDL blocks. The Modelica files that are used to implement this controller, including all control blocks that are needed to hierarchically compose the controller, consist of about 7,500 lines and 260,000 characters (not counting leading and trailing white spaces). In comparison, the JSON file has about 100,000 lines and 2,200,000 characters. Hence, it is about 10 times larger than the Modelica representation. While it is significantly larger, many tools exists to parse and manipulate JSON. On the other hand, Modelica, while easy to read by humans, is harder to parse programmatically. Therefore, to limit the barrier for other tool developers to write applications, JSON has been used as an intermediate format which can then be used by downstream applications.

HTML

The HTML output format is used to document control sequences in a format that does not require any knowledge of Modelica by the user. For example, the HTML output could be used for operator instructions or for documentation of repositories of control sequences.

The `html` output option will create one HTML file for the sequence. It will also create image files of the control block diagrams in SVG format, and display them as part of the HTML representation.

Furthermore, point lists can be added to the HTML output. These can be used by the mechanical designer to specify installation requirements, and by the control provider as input for pricing calculations.

The HTML documentation is structured by first listing the top-level block, and then listing recursively all blocks that are used to implement the controller. For each block, the `info` section of the Modelica implementation is processed and hyperlinks to all parameters, inputs, outputs and other blocks are added automatically. Next, tables with all parameters, inputs and outputs are added. Afterwards, public and protected blocks and their configurations are listed, and hyperlinks to their respective documentation are added. Lastly, all connections are listed, and the block diagram is included before the documentation of the next block starts.

SVG Diagram

With the `svg` option, an SVG diagram of the control sequence can be generated. This can for example be used for documentations that do not require the full HTML output. For example, Figure 2a has been generated by the SVG generator of `modelica-json`.

MS Word

The MS Word output format is used to generate an editable MS Word document of the control sequence. The MS Word output is generated by translating the HTML output using the JavaScript `html-docx-js` library. This document contains the same structure and content as the HTML output, including SVG files and hyperlinks.

ALC webCTRL

webCTRL is a commercial control product line from Automated Logic, a Carrier company (ALC). It is based on a block diagram language called EIKON that has predefined elementary blocks, and a line programming language for adding custom-blocks.

Translating a control sequence from CDL to ALC webCTRL involves two steps. First, the JSON representation is generated from CDL using `modelica-json` as described above. This step only uses open-source software. Next, a separate software program is used to translate this JSON representation to EIKON code. Because this step is specific to the control product line, and may require proprietary knowledge that is specific to the control product line, this step may be done with proprietary code, which is the case for the JSON to EIKON translation.

If this becomes commercially available, it provides to ability to greatly reduce the cost, complexity, and errors involved in moving a control sequence from design into implementation.

To standardize this translation, an ASHRAE standards project team has been formed to develop ASHRAE Standard 231P "CDL - A Control Description Language for Building Environmental Control Sequences". This standard is intended to make the exchange of information between controls design, simulation and implementation open and interoperable.

Brick

Brick is a graph-based data model. It describes the equipment, assets, subsystems and data sources in buildings and the relationships between them. Data-driven applications can use the formal axioms and semantic definitions of Brick to discover and retrieve the metadata and data required for their operation.

To generate a Brick model, `modelica-json` is used to parse the Modelica system model that contains the HVAC, building and control models, and output its JSON representation. This JSON representation is then further processed using the JSON to Brick translator. This second software identifies the entities and the relationships between them, as this closely resembles the structure of a Brick model. It then maps Modelica classes to Brick classes, for which we developed a mapping dictionary for a limited set of classes of the Modelica Buildings Library. For example, every instance of the Modelica class `Buildings.Fluid.Sensor.Temperature` can be translated into a `brick:Temperature_Sensor` entity. Relationships between Brick entities are discovered through inspecting the Modelica `connect` clause statements which describe how the ports of Modelica objects relate. Based on type information associated with the ports, the translator determines which Brick relationship is most appropriate for connecting two Brick entities. In Brick, these relationships can be sequential, declared in Brick as `brick:feeds`, or compositional, declared as `brick:hasPart` or `brick:hasPoint` Fierro et al. (2020). To generate a useful Brick model, a Modelica model that also contains the HVAC and building, as opposed to only the controls, should be used because relationships are inferred from the HVAC and building model.

Examples

We will now present three examples that use the `modelica-json` translator. The first two examples use a multizone VAV system with a control sequence from ASHRAE Guideline 36, of which the control sequence shown in Figure 2a is part of. The first example translates the control sequence from CDL to ALC webCTRL. The second example exports a Brick semantic model from the Modelica model that contains this control sequence. The third example uses a single zone VAV control sequence from ASHRAE Guideline 36. It uses its JSON output from `modelica-json` together with its CDL representation to verify the implementation of the control sequence in an ALC webCTRL controller.

Export of CDL to ALC webCTRL

With the cooperation of ALC, we completed a proof of concept demonstration to show the feasibility of translating from CDL into the proprietary commercial control language ALC webCTRL. We used the VAV multi-zone air handler control sequence from ASHRAE Guideline 36, public review draft 1. The same approach can also be used for other sequences.

The air handler unit (AHU) control sequence, and the terminal unit controller that is shown in Figure 2a, was exported to the JSON representation. From this JSON representation, the control blocks were translated to EIKON. This step included the following further steps, which are all specific to the target control platform: For composite control blocks that include in a hierarchical structure lower level controllers, the lower level controllers were further translated recursively. Once only elementary blocks were encountered, they were translated to EIKON code. For some blocks, a one-to-one mapping to native EIKON object was performed. For example, for a logical `or`-block, a one-to-one mapping is possible. Other blocks required 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_2 and inputs u_1 and u_2, whereas in EIKON, the na-

tive add-block outputs $y = u_1 + u_2$. For such cases, OCL line programming code has been generated unless $k_1 = k_2 = 1$. Next, parameter values were assigned, and in some cases evaluated because CDL allows statements for parameter propagation that are not supported by EIKON. Lastly, components were laid out graphically, and inputs and outputs were connected and laid out graphically. CDL declares the graphical layout, and CDL allows lines to be on top of each other, while EIKON does not. Hence, some graphical changes were needed at this step. All these operations are specific to the control target platform, and some may require proprietary knowledge of the control target platform. Therefore, they reside in the second step of the translator which is proprietary.

At the end of this translation process, an implementation of the control logic in EIKON is available. For example, Figure 2b shows the translation of the CDL control logic shown in Figure 2a. Once translated, the EIKON implementation need not know anything about CDL.

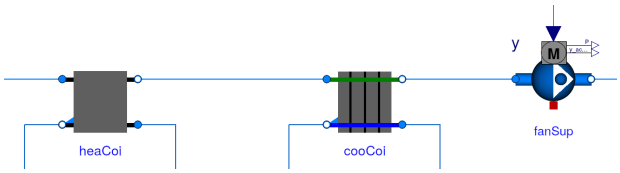
This proof of concept demonstrated the ability to have a digital workflow that started with CDL, exported it to JSON, and then translated it into the ALC proprietary programming language.

Export of a semantic model using Brick

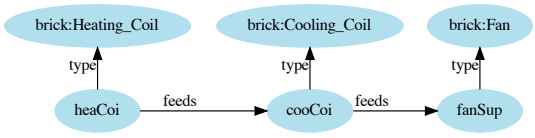
For this example, we translated the model `Buildings.Examples.VAVReheat.Guideline36` from the Modelica Buildings Library. Figure 3 shows the heating coil, the cooling coil and the supply air fan of the AHU in the Modelica model and its corresponding Brick translation. To generate the Brick model, we first generated the JSON representation of the Modelica model using the Modelica mode of `modelica-json`. This JSON file, together with a dictionary that maps Modelica classes to Brick classes, were then used as input to the Brick translator. As mentioned previously, during the translation process, Modelica classes were mapped to Brick equivalents for equipment, devices and sensors. For example, the heating coil that is modeled using the Modelica `DryCoilCounterFlow` model was mapped to the Brick `Heating_Coil` entity. Through such mappings, instances of Modelica classes become Brick entities in the exported Brick graph. By traversing the Modelica `connect` statements, the relationships between the different Brick entities were discovered. For instance, the connect statements between the `cooCoi` and `fanSup` was extracted as a `brick:feeds` relationship. For further discussions of the translation and use of this Brick model, see Fierro et al. (2020).

Verification of as-installed controller

Another tool that uses `modelica-json` is the sequence verification tool that is being developed as part of the OpenBuildingControl project at LBNL. The objective of this tool is to formally verify that control sequences installed in actual control hard-



(a) Partial view of an air handler unit in the Modelica model.



(b) Corresponding Brick instance of the components shown above.

Figure 3: Excerpt from `Buildings.Examples.VAVReheat.Guideline36` in (a) Modelica and (b) its translation in Brick.

ware conform to their CDL specification. Figure 4 describes the different steps involved in testing a controller loaded with a control sequence. The steps are as follows: First, `modelica-json` is used to convert the CDL sequence from Modelica to its JSON representation from which the tool extracts the names, units and values of parameters, and the names and units of the input and output variables of the control sequence. Using the OpenModelica simulator (Fritzson et al., 2020), the CDL representation of the sequence is simulated to produce the simulated reference control output time series. Next, to test the real controller, the names and units of all input variables, output variables and parameters are extracted from the JSON representation and mapped to the corresponding point names and units of the controller. Now, the controller can be executed. Using BACnet, input values are set and output values are trended to a file. Finally, the sequence comparison step compares the output time series from the controller with the simulated reference output time series. This comparison is done using the `pyfunnel` software.[2]

Figure 5 is an example output for the active heating set point after a failed verification of a zone set point temperature sequence for a single zone variable air volume controller from ASHRAE Guideline 36. Note that for this illustration, we manually introduced a control error near $t = 60$ seconds to show how a failed test looks like. The shaded region represents user-configurable tolerance bounds. Around $t = 60$ seconds, the controller output exceeds this bounds, leading to a failed test.

Not shown here is a related use case for which the verification software may be used: If a controller is already in operation to control a real HVAC system, then its input time series and output time series may

[2]The `pyfunnel` software is available from `https://github.com/lbl-srg/funnel`.

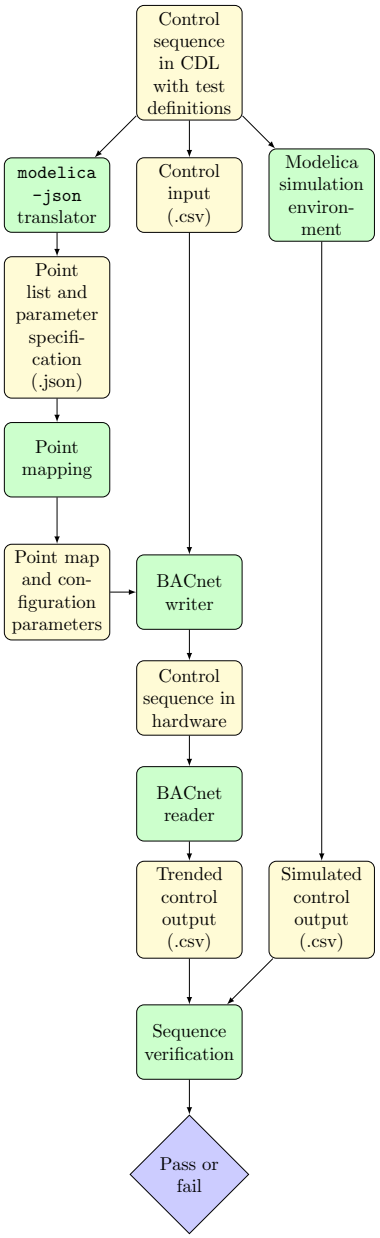


Figure 4: Different files (yellow) and processes (green) involved in the sequence verification tool.

be trended. Then, the CDL reference model of that sequence can be simulated, using the trended control input time series as input. The CDL reference model then simulates the expected control output time series. The verification step then checks whether the trended control output is within the user-specified tolerance of the expected, simulated control output.

Ongoing and future work

The following work, through collaborations among National Labs, academia and industry, is all ongoing in support of a model-based, digitized workflow for the development and deployment of high performance control sequences for grid-interactive, efficient buildings. CDL and its JSON representation are at the time of this writing reviewed by the ASHRAE

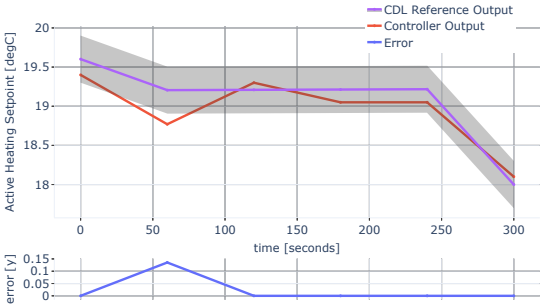


Figure 5: An illustration of the output chart generated by the sequence verification tool for a zone heating set point. The shaded region is the region of acceptance, and the blue trajectory shows the error between reference and actual control output.

Standard 231P committee whose purpose is to standardize a control description language for mechanical systems, active facades and lighting. Other activities under development include

1. libraries of control sequences in CDL for distribution with the Modelica Buildings Library,
2. a next-generation simulation engine, Spawn of EnergyPlus that will enable integrated building energy and control design and deployment workflows (Wetter et al., 2020),
3. a control sequence selection and configuration tool called `linkage` that will export control sequences in CDL, a Spawn energy model and documentation that support the construction process,
4. tools for export of semantic models from Modelica to Brick, and
5. tools for formal verification of control sequences.

Conclusions

By using a declarative modeling language, in our case a small subset of Modelica called Control Description Language, we developed and demonstrated prototype tools for digitizing the control delivery process. This process has benefits for many stakeholders: Control specialists can implement libraries of customizable control sequences. Energy modelers can use sequences from these libraries, customize them to their building, and then test, improve and compare the closed loop performance of different control sequences using energy simulation. Next, they can export a digital specification of the sequence, including their natural language description, for bidding, implementation and commissioning. Control providers can build tools for semi-automated bidding and for import of the sequences into their control product platform. Commissioning agents can use the digital control logic specification for formal verification of as-installed sequences. Building analytics and FDD providers can use the semantic model to bootstrap their software deployment.

Government agencies and professional organizations can assess the closed loop control performance when developing new energy codes, and then provide a digital specification of the sequences to lower the barrier for implementation by control providers, mechanical engineers, simulation tool developers, and for code compliance checking.

While we demonstrated feasibility and benefits through our prototype implementation, future work needs to focus on rising the Technology Readiness Level needed for adoption by industry.

Acknowledgment

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231, the California Energy Commission's Electric Program Investment Charge (EPIC) Program, and Automated Logic, a Carrier company.

This work emerged from the IBPSA Project 1, an international project conducted under the umbrella of the International Building Performance Simulation Association (IBPSA). Project 1 will develop and demonstrate a BIM/GIS and Modelica Framework for building and community energy system design and operation.

References

ASHRAE (2006). *Sequences of Operation for Common HVAC Systems*. ISBN 1-931862-98-2.

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

Balaji, B., A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M. B. Kjærgaard, M. Srivastava, and K. Whitehouse (2016). Brick: Towards a unified metadata schema for buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, BuildSys '16, New York, NY, USA, pp. 41–50. Association for Computing Machinery.

Barwig, F. E., J. M. House, C. J. Klaassen, M. M. Ardehali, and T. F. Smith (2002, August). The national building controls information program. Summer Study on Energy Efficiency in Buildings, Pacific Grove, CA. ACEEE.

Drgoňa, J., J. Arroyo, I. Cupeiro Figueroa, D. Blum, K. Arendt, D. Kim, E. P. Ollé, J. Oravec, M. Wetter, D. L. Vrabie, and L. Helsen (2020). All you need to know about model predictive control for buildings. *Annual Reviews in Control 50*, 190–232. doi: 10.1016/j.arcontrol.2020.09.001.

PNNL (2017, May). *Impacts of Commercial Building Controls on Energy Savings and Peak Load Reduction*.

Fierro, G., A. K. Prakash, C. Mosiman, M. Pritoni, P. Raftery, M. Wetter, and D. E. Culler (2020). 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, New York, NY, USA, pp. 70–79. Association for Computing Machinery.

Fierro, G., M. Pritoni, M. Abdelbaky, D. Lengyel, J. Leyden, A. K. Prakash, P. Gupta, P. Raftery, T. Peffer, G. Thomson, and D. E. Culler (2019). Mortar: An open testbed for portable building analytics. *ACM Transactions on Sensor Networks 16*(1), 7:1–7:31. doi: 10.1145/3366375.

Fritzson, P., A. Pop, K. Abdelhak, A. Ashgar, B. Bachmann, W. Braun, D. Bouskela, R. Braun, L. Buffoni, F. Casella, R. Castro, R. Franke, D. Fritzson, M. Gebremedhin, A. Heuermann, B. Lie, A. Mengist, L. Mikelsons, K. Moudgalya, L. Ochel, A. Palanisamy, V. Ruge, W. Schamai, M. Sjölund, B. Thiele, J. Tinnerholm, and P. Östlund (2020). The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control 41*(4), 241–295. doi: 10.4173/mic.2020.4.1.

Parr, T. (2012, September). *The Definite ANTLR4 Reference*. Dallas, TX: The Pragmatic Programmers.

Wetter, M., K. Benne, A. Gautier, T. S. Nouidui, A. Ramle, A. Roth, H. Tummescheit, S. Mentzer, and C. Winther (2020, September). Lifting the garage door on Spawn, an open-source BEM-controls engine. In *Proc. of Building Performance Modeling Conference and SimBuild*, Chicago, IL, USA, pp. 518–525.

Wetter, M., A. Gautier, M. Grahovac, and J. Hu (2019, September). Verification of control sequences within OpenBuildingControl. In *16-th IBPSA Conference*, pp. 885–892. International Building Performance Simulation Association.

Wetter, M., M. Grahovac, and J. Hu (2018, August). Control description language. In *1st American Modelica Conference*, Cambridge, MA, USA.

Wetter, M., W. Zuo, T. S. Nouidui, and X. Pang (2014). Modelica Buildings library. *Journal of Building Performance Simulation 7*(4), 253–270. doi: 10.1080/19401493.2013.765506.