OpenBuildingControl

—

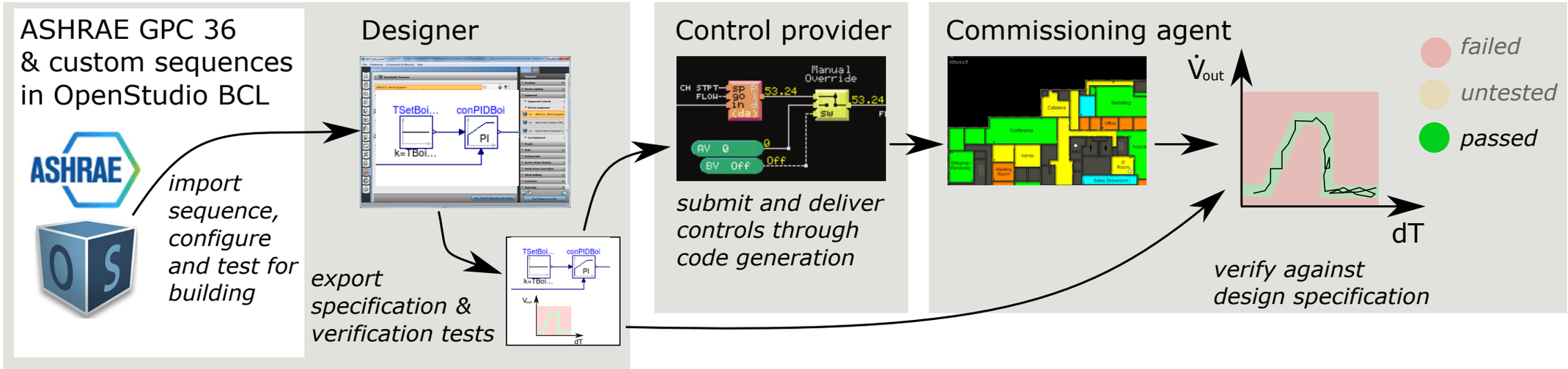
First specification and example

Michael Wetter

February 2, 2017

Lawrence Berkeley National Laboratory

OpenBuildingControl: Design and implement control sequences error-free and at lower cost to owner



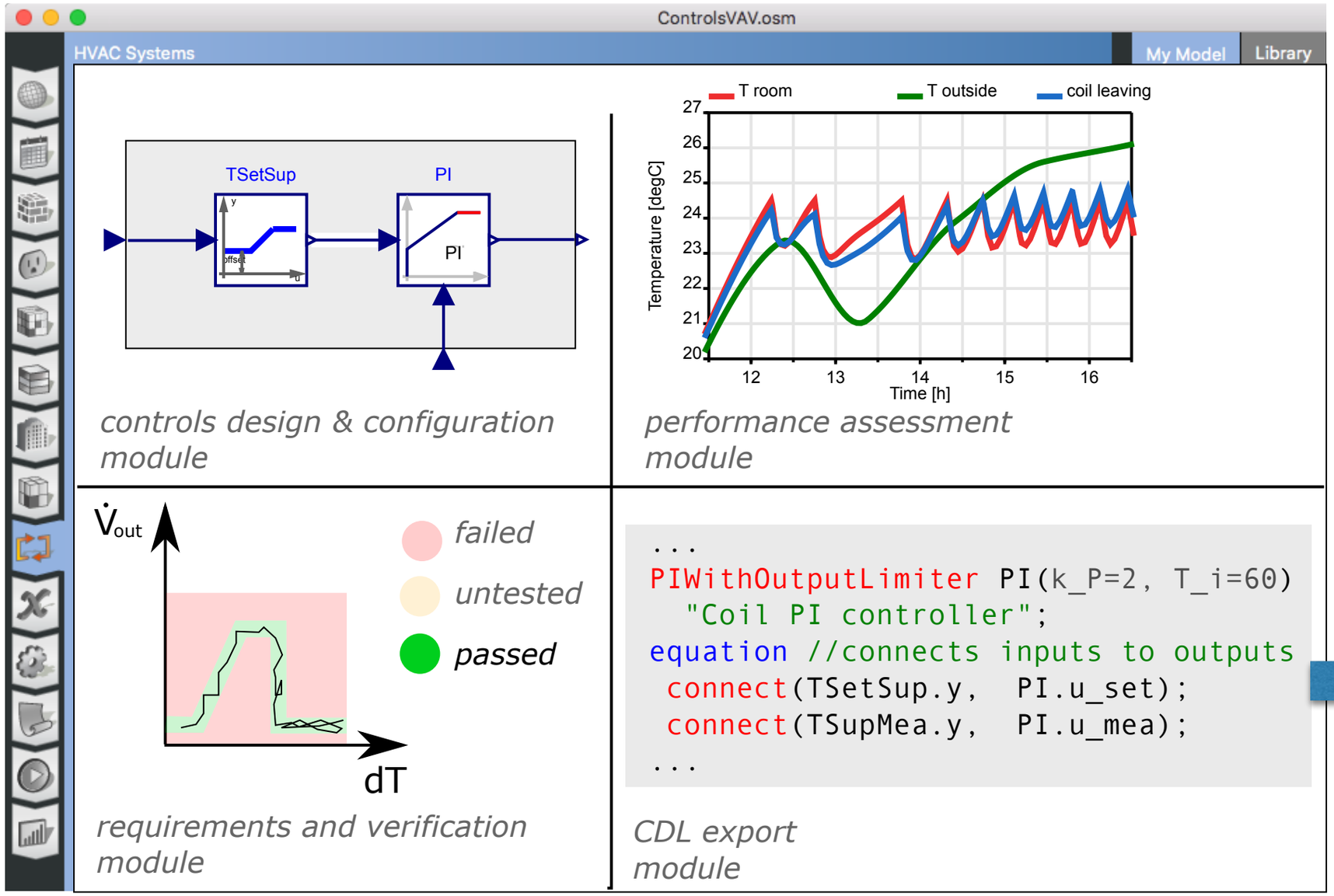
Codify best practice | Design | Implement | Verify against original design intent

BACnet standardizes communication.

OpenBuildingControl will standardize
- basic functional building blocks that are used to compose sequences and tests,
- expressing control sequences,
- expressing functional verification tests,

for bidding, automatic implementation and automated functional testing.

OpenBuildingControl: Design and implement control sequences error-free and at lower cost to owner



- Point list
- Bidding documents
- Operator manual
- …

Use cases and requirements

Use cases and requirements

Use case

- various use cases posted, but need additional use case provided by the team & TAG
- See http://obc.lbl.gov/specification/useCases.html

Requirements

- Started listing some requirements, needs further discussion (and expansion)
- See http://obc.lbl.gov/specification/requirements.html

All files can be edited on https://github.com/lbl-srg/obc/tree/master/specification/source

Control Description Language

What is CDL?

A language used to specify control sequences and verification tests.

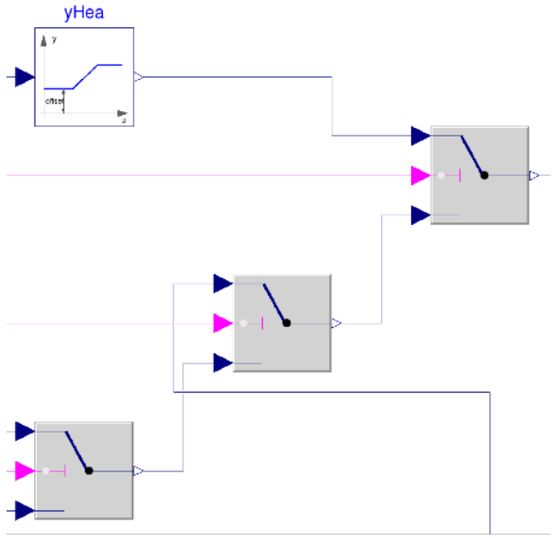
Not a control sequence.

Control sequences are specified, in a declarative way, *using* CDL.

What is CDL?

A declarative language for expressing block-diagrams for controls (and requirements)

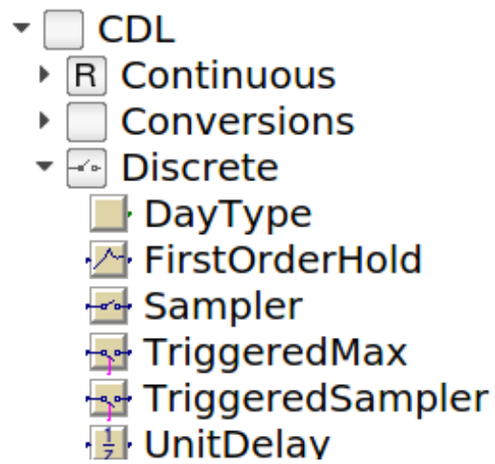
A graphical language for rendering these diagrams.



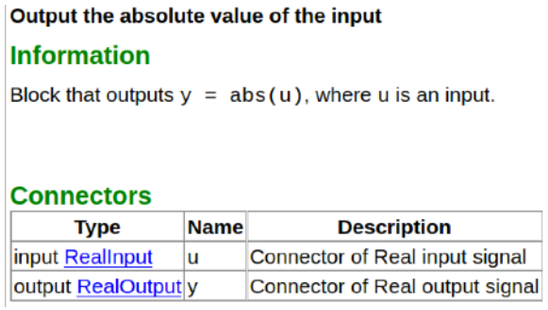
A library with elementary input/output blocks that should be supported [through a translator] by CDL-compliant control providers

Example: CDL has an adder with inputs `u1` and `u2`, gains `k1` and `k2`, and output

`y = k1*u1 + k2*u2`.



A syntax for documenting the control blocks and diagrams.



Output the absolute value of the input

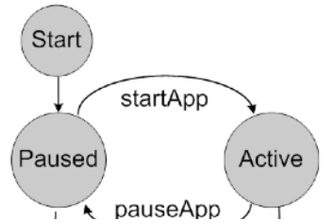
Information

Block that outputs `y = abs(u)`, where u is an input.

Connectors

Type	Name	Description
input RealInput	u	Connector of Real input signal
output RealOutput	y	Connector of Real output signal

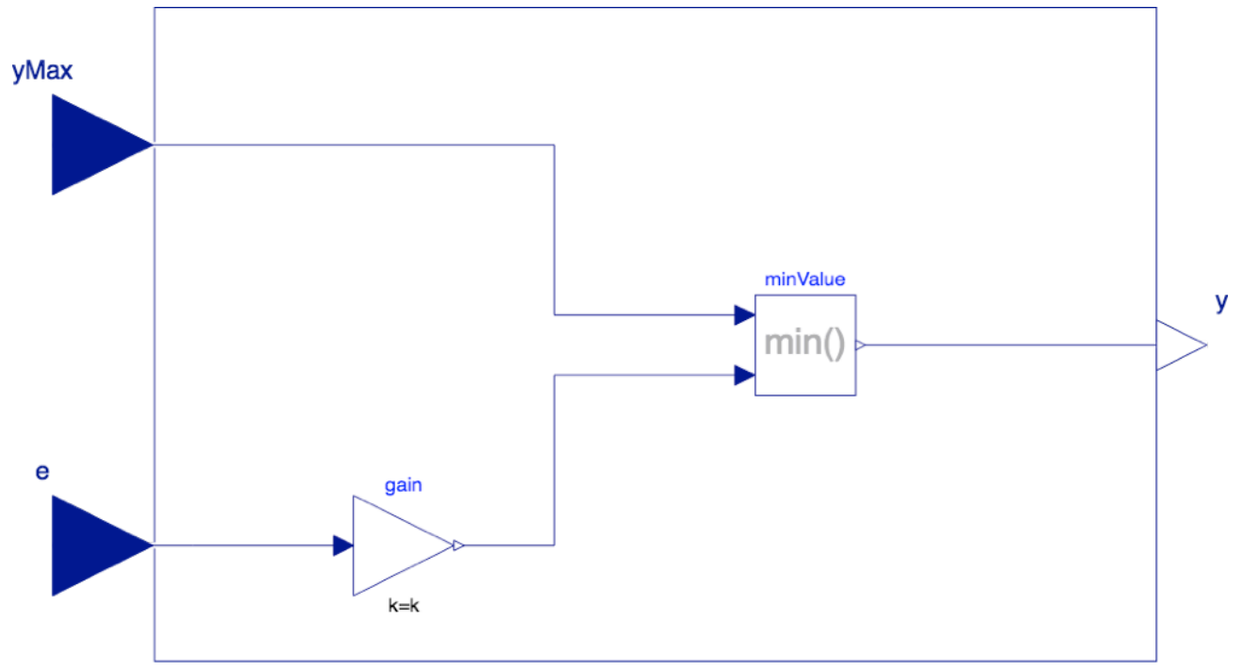
A model of computation that describes the interaction among the blocks.



Developed first version of specification for review and further implementation

Proposed

- Syntax
- Permissible data types
- Encapsulation of functionality
- Instantiation
- Connectors
- Connections
- Annotations
- Composite blocks
- Model of computations

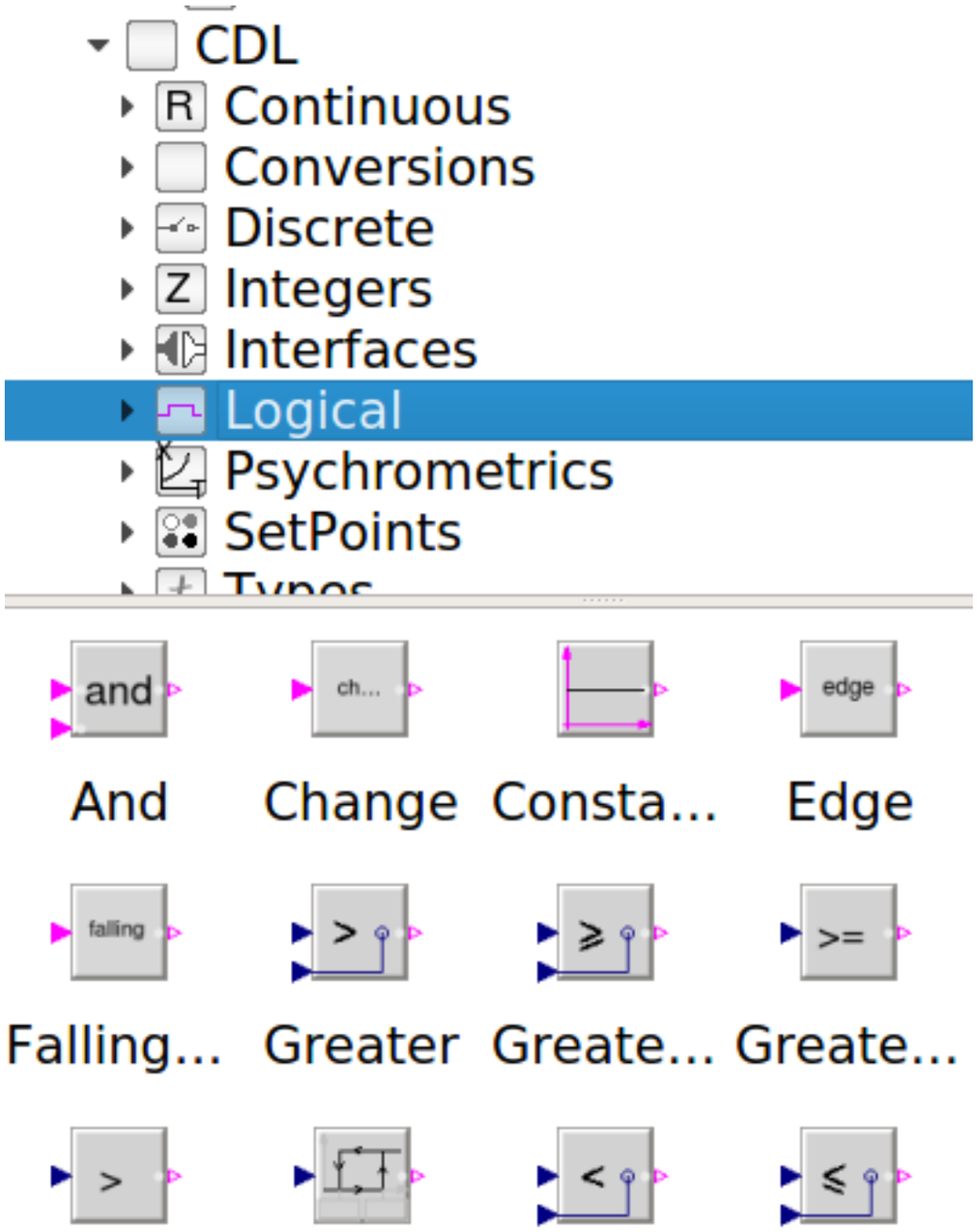


See specification for details: http://obc.lbl.gov/specification/cdl.html

Developed first version of CDL library

Created library with basic CDL blocks.

Need input from TAG to review, add new blocks as needed and remove what should not be in CDL.



Browse CDL library at

http://obc.lbl.gov/specification/cdl/latest/help/CDL.html

Sequence Specification

Example: VAV Temperature and Fan Speed Set Points

ASHRAE Guideline 36

BSR/ASHRAE Guideline 36P, *High Performance Sequences of Operation for HVAC Systems*
First Public Review Draft

air is cool, while avoiding excessive fan energy use and utilizing the cooling coil when outdoor air is warm.

For this to work, it is essential that both SATsp and SATsp-C are controlled off the same physical SAT sensor.

It is also critical that the minimum value of the setpoint that controls the economizer (SATsp) is lower than the minimum value of the setpoint that controls the chilled water valve (SATsp-C). Otherwise a brief temperature excursion due to the cooling coil will lead to short cycling of the economizer and subsequent unnecessary energy use by the cooling coil.

2. Supply Fan Speed Control and Supply Air Temperature Setpoint Reset

   a. The supply fan shall run whenever the unit is in any mode other than Unoccupied Mode.

   b. Provide a ramp function to prevent changes in fan speed of more than 10% per minute.

   c. Minimum, medium and maximum fan speeds shall be as follows:

      1) Maximum cooling fan speed (MaxCoolSpeed), maximum heating fan speed (MaxHeatSpeed) and minimum fan speed (MinSpeed) setpoints shall be per 3.2B.1.

      2) Medium fan speed (MedSpeed) shall be reset linearly based on outdoor air temperature between the following endpoints:

         a) When the outdoor air temperature equals the zone temperature +1°F, Medspeed shall be MinSpeed.

         b) When the outdoor air temperature is 5.6°C (10°F) below the zone temperature, Medspeed shall be equal to MaxCoolSpeed.

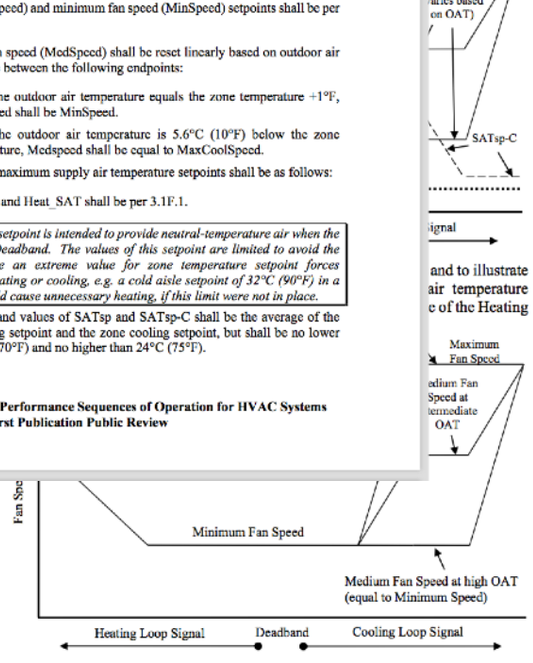
   d. Minimum and maximum supply air temperature setpoints shall be as follows:

      1) Cool_SAT and Heat_SAT shall be per 3.1F.1.

The Deadband setpoint is intended to provide neutral-temperature air when the Zone State is Deadband. The values of this setpoint are limited to avoid the situation where an extreme value for zone temperature setpoint forces unnecessary heating or cooling, e.g. a cold aisle setpoint of 32°C (90°F) in a datacenter could cause unnecessary heating, if this limit were not in place.

      2) The Deadband values of SATsp and SATsp-C shall be the average of the zone heating setpoint and the zone cooling setpoint, but shall be no lower than 21°C (70°F) and no higher than 24°C (75°F).

ASHRAE Guideline 36: High Performance Sequences of Operation for HVAC Systems
First Publication Public Review

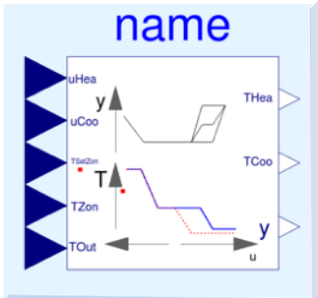


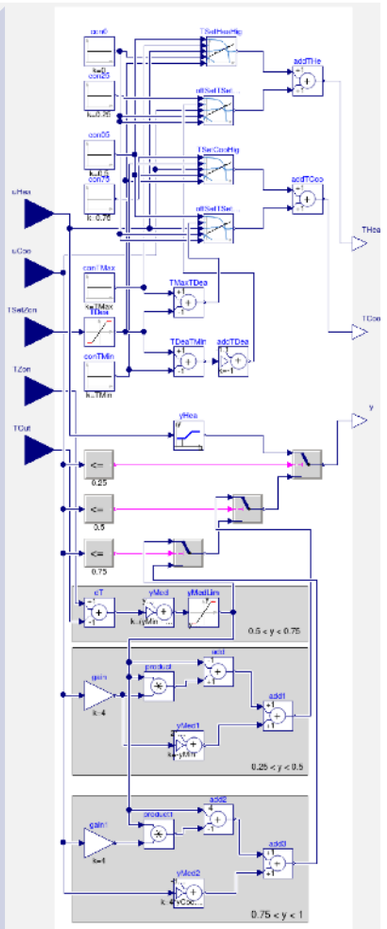
ASHRAE Guideline 36: High Performance Sequences of Operation for HVAC Systems
First Publication Public Review

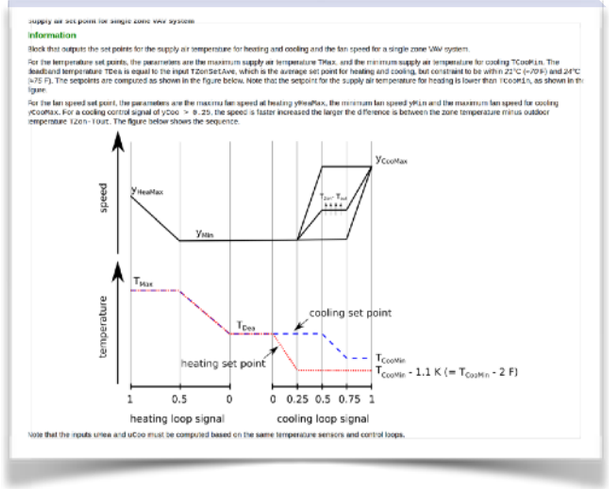
value to Cool_SAT minus 1.1°C (2°F), while SATsp-C is the Deadband value.

ASHRAE Guideline 36: High Performance Sequences of Operation for HVAC Systems
First Publication Public Review

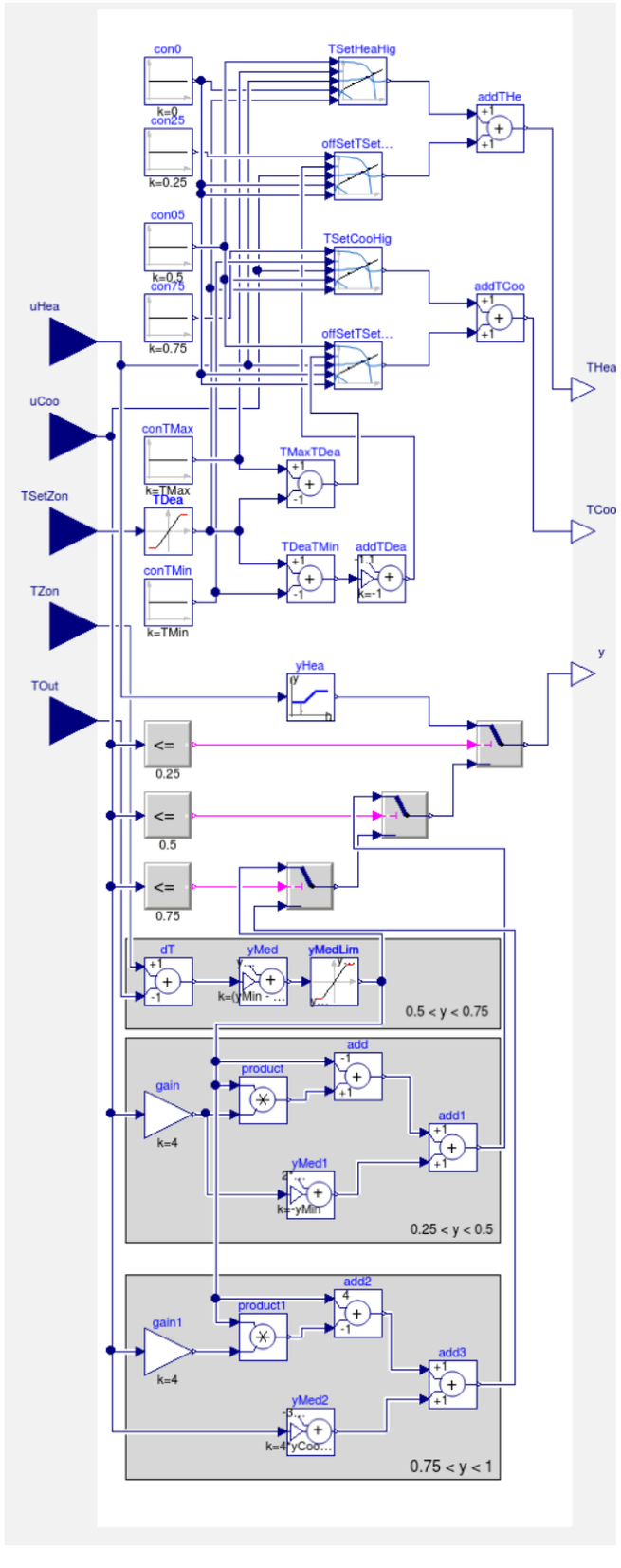
Implementation using CDL

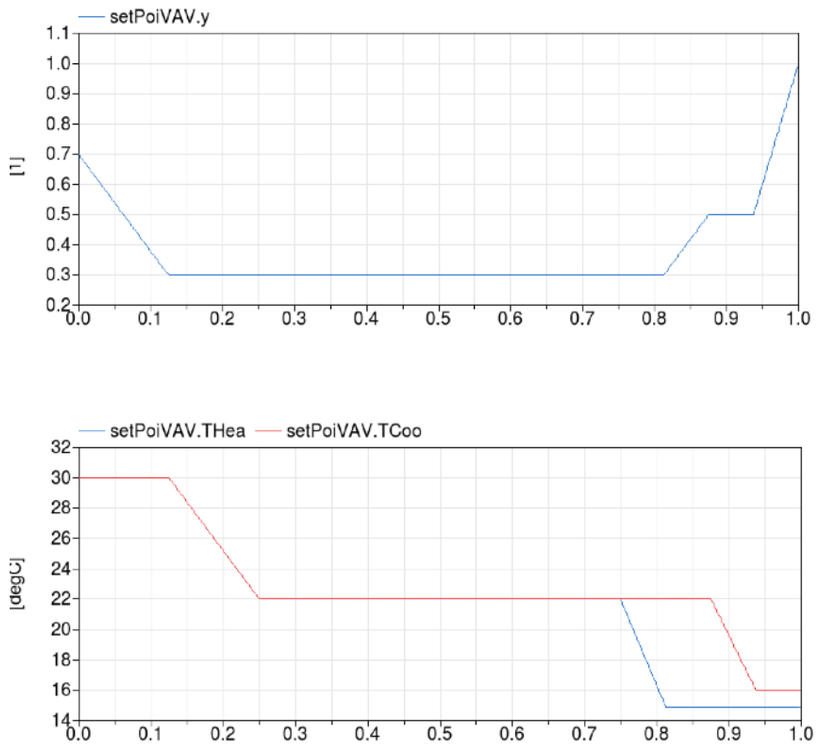


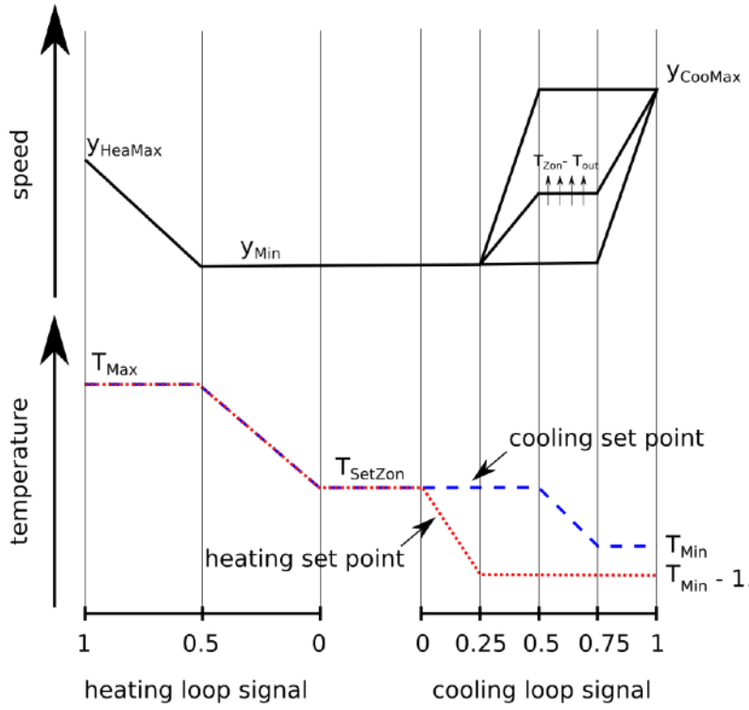




Example: VAV Temperature and Fan Speed Set Points

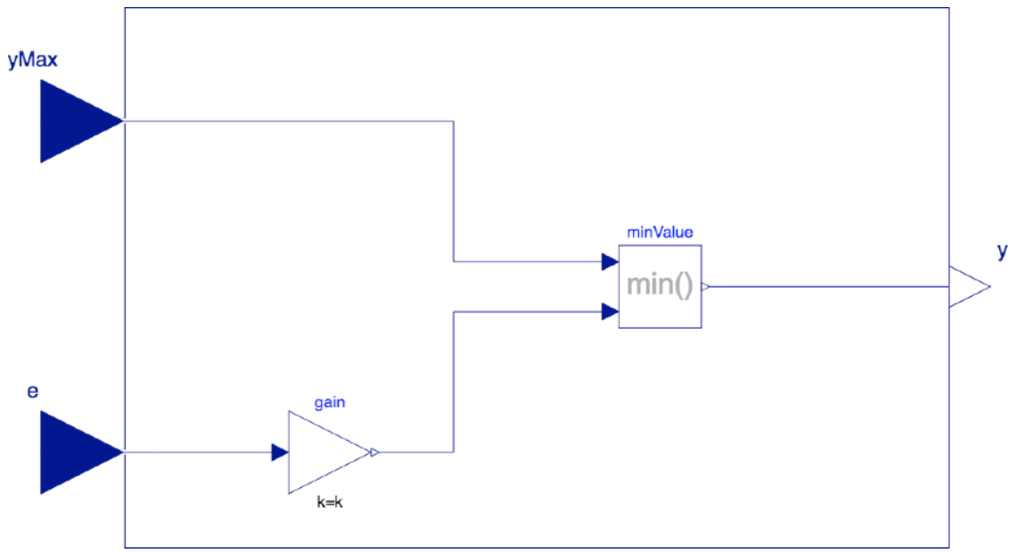






Sequence is implemented on development branch on branch 609_cdl on [Buildings library github repository](#).

Custom sequences can be specified using blocks from CDL, pre-configure ASHRAE G36 sequences (and any custom-library that is based on CDL)




```
model CustomPWithLimiter
  "Custom implementation of a P controller with variable output
limiter"

  parameter Real k "Constant gain";

  Interfaces.RealInput yMax "Maximum value of output signal”;

  Interfaces.RealInput e "Control error";

  Interfaces.RealOutput y "Control signal";

  Math.Gain gain(final k=k) "Constant gain";

  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 output <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;
```

Graphical annotations omitted.

CDL is used to implement open and proprietary sequences

The standard to be supported by vendors

- CDL
- ASHRAE
- G36

Sequences that come out of ASHRAE projects and can be shared with community.

Custom implementations that are built using the CDL language, and CDL blocks

- GSA
- ARUP
- ALC

GSA preferred sequences, made available through a CDL-complaint implementation.

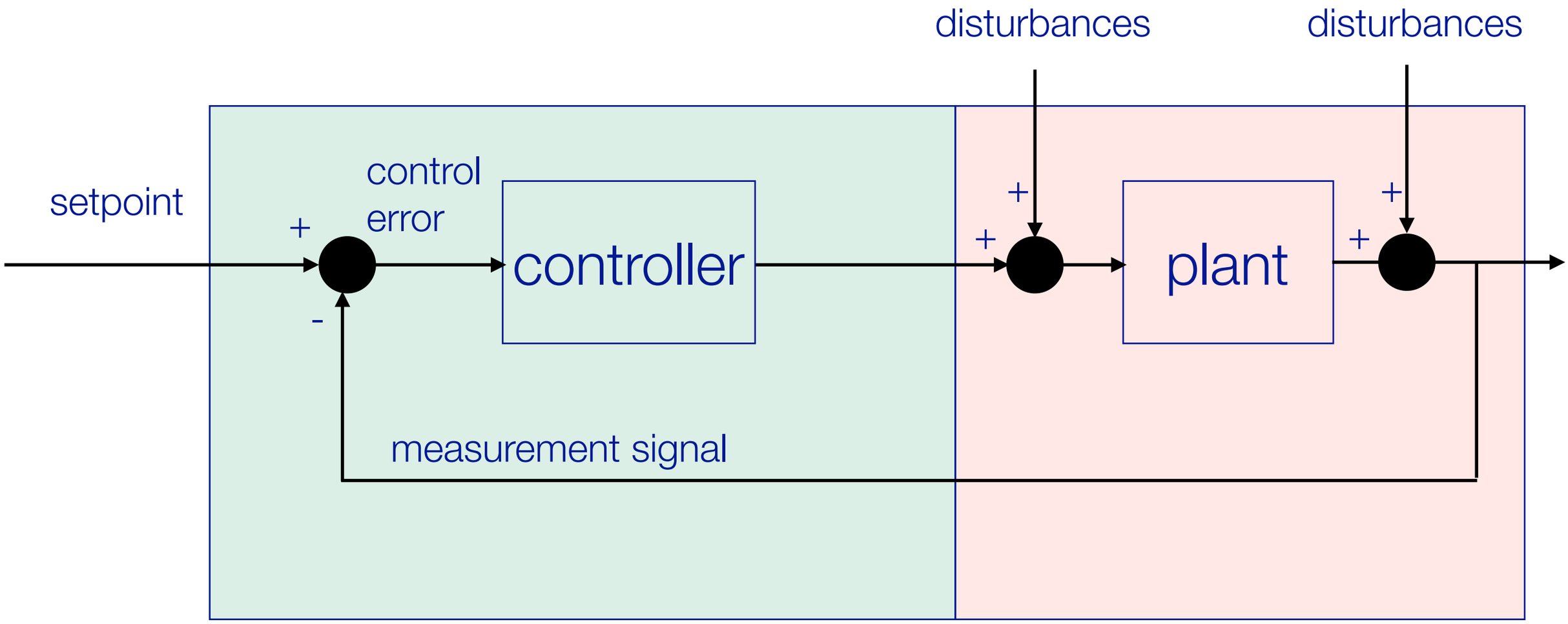
Design firms can share their own (proprietary) implementation across their offices.

Control vendors can provide their own specialized sequences, either as open-source, or as compiled (proprietary) I/O blocks.

Companies are illustrative

Verification Test

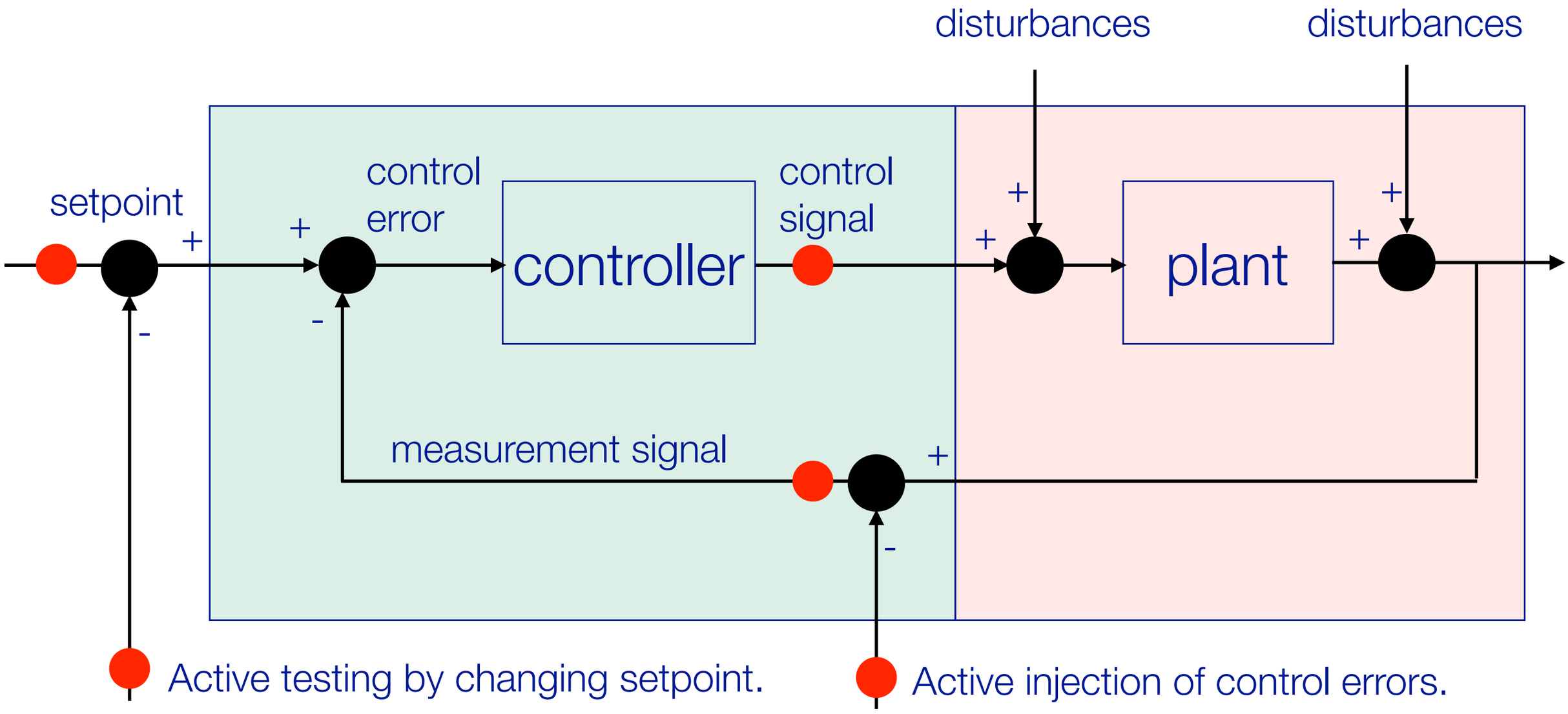
What should be verified?



Disturbances and plant are only approximately known, and hence should be excluded from the verification of the *control delivery*.

But they should be part of an end-to-end verification of the *building delivery*.

How should we verify?

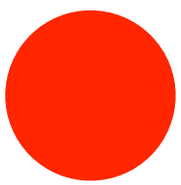


Active testing by changing setpoint.

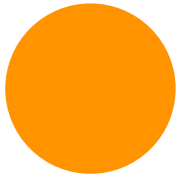
Active injection of control errors.

Red points indicate which signals to verify against a CDL generated response.

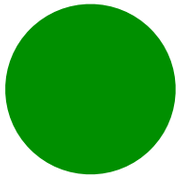
3-valued logic



Violated: Test condition is **violated** at least once.

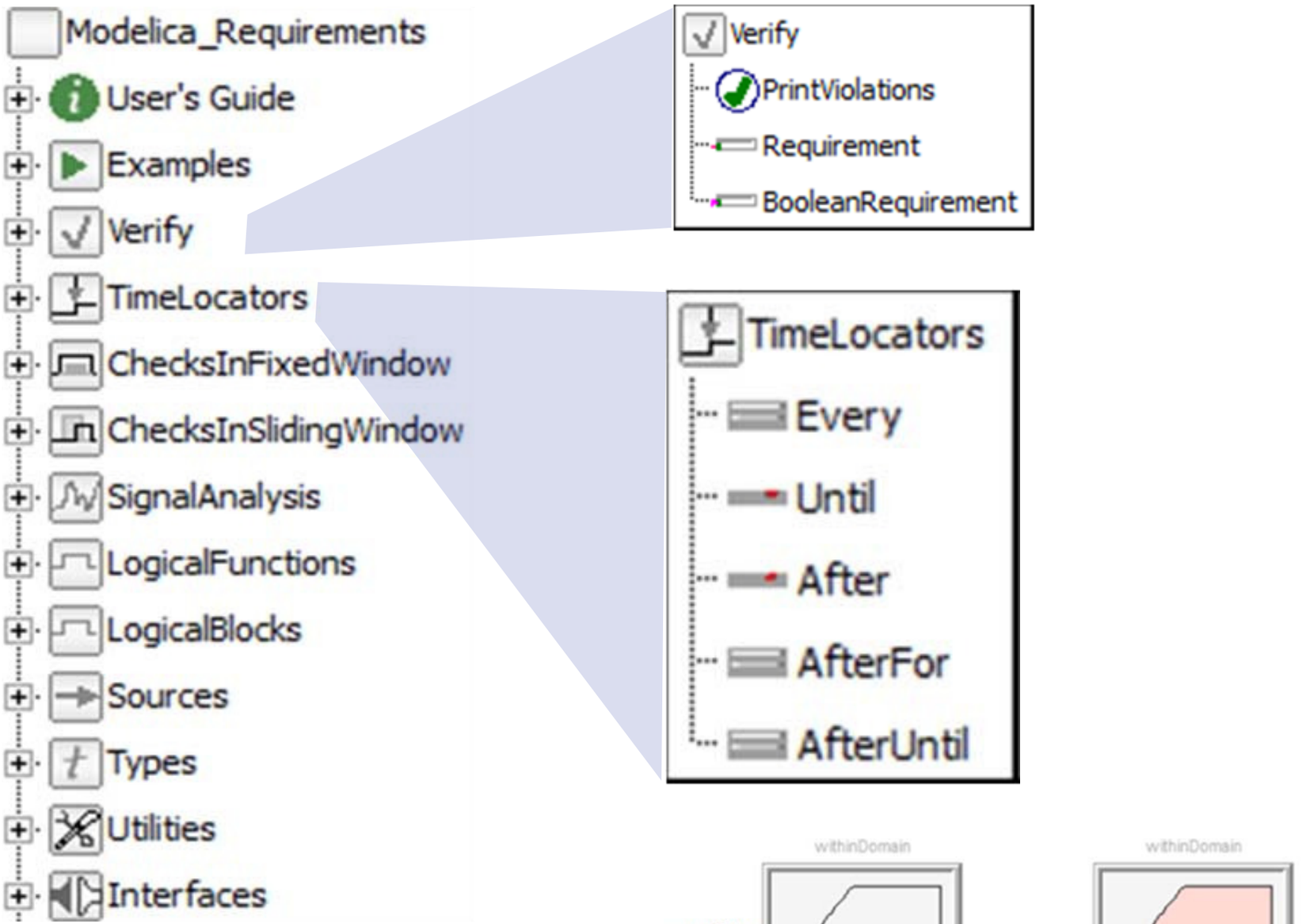


Untested: Test condition is **undecided** for the complete test period.



Satisfied: Test condition is **satisfied** at least once, and is never violated.

Implementation based on Modelica_Requirements



Images from [Otter et al. 2015]

Iteration over objects to check requirements

```
record PumpObservation
  "Observation signals needed for one pump”
  constant String name "Name of pump";
  Boolean cavitate "= true, if pump cavitates";
end PumpObservation;


PumpRequirements req(
  obs={PumpObservation(
    name = c.getInstanceName(),
    cavitate= c.p <= 1e4)
      for c in class HeatingPump})
```

Source: Adapted from Modelica Change Proposal MCP-0021 Component Iterators
Status: Under Evaluation 2015-12-08, version v2, #1848

Next

	Year 1												Q5
	Q1			Q2			Q3			Q4			
1 Specification			M1.1						M1.2				
2 Controls design tool													
2.1 Requirements and software architecture						M2.1							
2.2 Impl. of ctrl seq. (secondary sys.)											M2.2		
2.3 Impl. of ctrl seq. (primary sys., facade & lighting)													
2.4 Impl. of GUI													
2.5 CDL export to English language and a product line													
2.6 OpenStudio integration													
3 Functional verification tool													
3.1 Requirements and software architecture													
3.2 Impl. of hardware interface													
3.3 Impl. of verification test module													M3.1
3.4 Impl. of GUI													
4 Case studies													
5 Com. and market transformation plan													

Next

Need input and collaboration from the project team and TAG:

1. Add use cases and requirements to the templates (http://obc.lbl.gov/specification/).

2. Review the list of CDL blocks, and add/remove/revise blocks
    1. Current implementation documented at http://obc.lbl.gov/specification/cdl/latest/help/CDL.html
    2. For work in progress, see https://github.com/lbl-srg/modelica-buildings/issues?q=is%3Aissue+is%3Aopen+label%3AOpenBuildingControl

3. Work on functional verification tool specification
   (need to specify requirements and software architecture in months 1 & 2)

Resolve questions with project team and TAG:

4. How do CDL basic blocks compare to what manufacturers currently implement?

5. How to represent optimal start up and cool down for thermal zones? [And other proprietary implementations.]

6. What type(s) of PID controller should be in CDL (implementation of anti-windup, reset of state or output)
    - May need CDL, and extensions for particular vendor implementations, to be activated after vendor is selected.

7. Should signals that carry enumerations be supported by control vendors
   (e.g.., `enum dayType = {WeekDay , WeekEnd, Holiday}`).

8. What model of computations are used by the control vendors?