OpenBuildingControl

Control Description Language Specification developed in OpenBuildingControl

—

Michael Wetter
Paul Ehrlich
Jianjun Hu
Antoine Gautier
Milica Grahovac

October 28, 2020

Lawrence Berkeley National Laboratory

Overview

Objectives

- Understand current
  - specification,
  - implementation,
  - tools, and
  - use of CDL.
- How does it relate to Modelica, and what are the Modelica, FMI, eFMI, SSP Standards?

Outline

- Introduction
- Development process
- CDL specification
- Current use
- Foundational standards
- Demo

Introduction

A brief history of CDL

First specified in fall 2016 in OpenBuildingControl Phase I.

Spring 2017: First released through the Modelica Buildings Library.

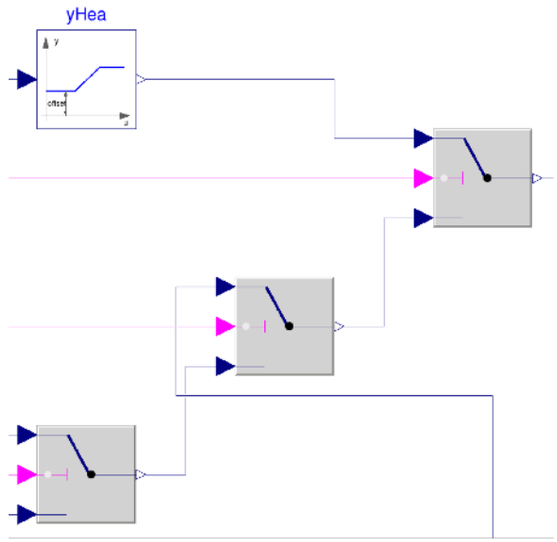
Since then in use in OpenBuildingControl projects to

- implement control sequences,
- compare their closed loop performance, and
- export to json, Word, HTML and ALC EIKON (prototype).

Fall 2020: Signed contract to develop a control sequence selection and configuration tool, aka “Linkage”

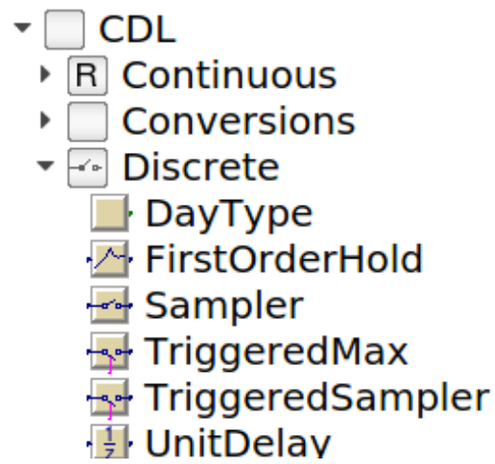
What is the Control Description Language?

A **declarative language** for expressing block-diagrams for controls

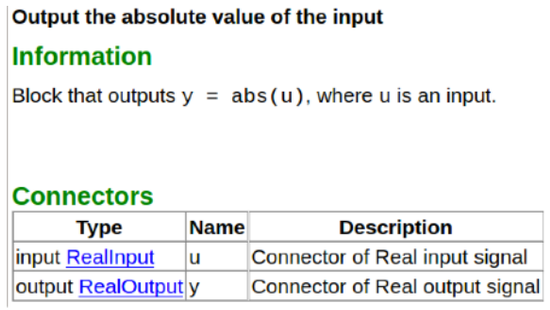


A **library with elementary input/output blocks.**
Their mathematical functionality [but not their implementation] needs to be provided by CDL-compliant control providers.

Example: CDL has an adder with inputs `u1` and `u2`, gains `k1` and `k2`, and output
`y = k1*u1 + k2*u2`.



A **syntax for documentation** and **tagging.**



Output the absolute value of the input

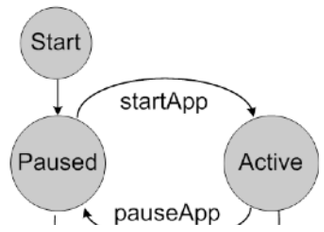
Information

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.



What is CDL not?

Not a programming language.

- -> its behavior is expressed mathematically, how to execute it is intentionally not specified.

Not a software tool implementation.

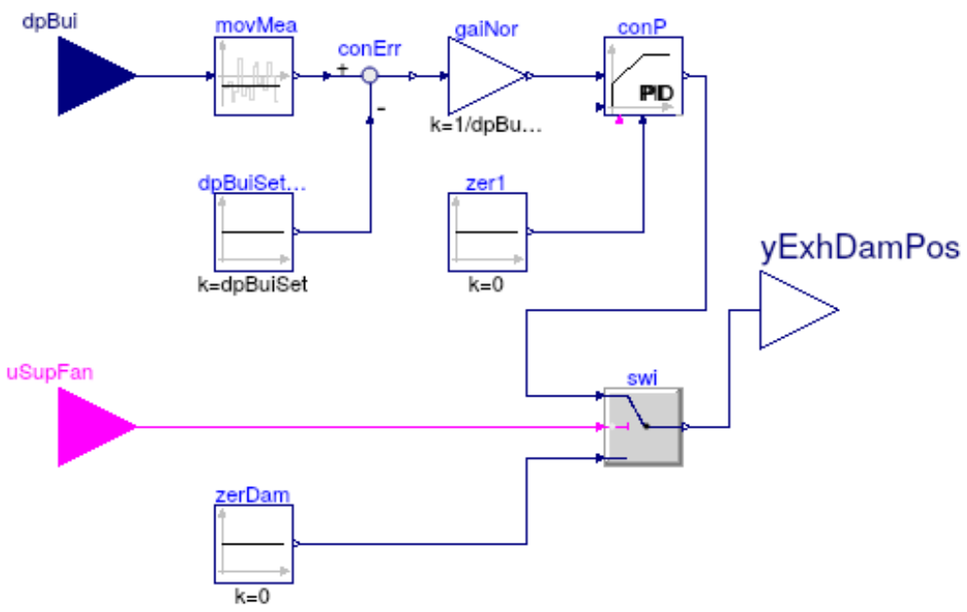
- It is declarative.
- How to edit, debug, show documentation etc. is up to tools that implement CDL.

Not a control sequence; what you do with the language is up to the user.

Not a communication specification. This is done through, e.g., BACnet.

Introductory example of CDL

Typical user view:



Information

Control sequence for actuated exhaust damper `yExhDam` without fans. It is implemented according to ASHRAE Guidline 35 (G36), PART5.N.8. (for multi zone VAV AHU), PART5.P.6 and PART3.2B.3 (for single zone VAV AHU).

Multi zone VAV AHU: Control of actuated exhaust dampers without fans (PART5.N.8)

1. The exhaust damper is enabled when the associated supply fan is proven on `uFan = true`, and disabled otherwise.
2. When enabled, a P-only control loop modulates the exhaust damper to maintain a building static pressure of `dpBui`, which is by default *12* Pa (*0.05* inchWC).
3. When `uFan = false`, the damper is closed.

Optional developer and power user view:

```
block ExhaustDamper
  "Control of actuated exhaust air dampers without fans"

  parameter Real dpBuiSet(
    final unit="Pa",
    final quantity="PressureDifference",
    max=30) = 12
    "Building static pressure difference setpoint";

...

  Buildings.Controls.OBC.CDL.Interfaces.RealInput dpBui(
    final unit="Pa",
    displayUnit="Pa")
    "Building static pressure difference"
    ∃ ;
...
  Buildings.Controls.OBC.CDL.Interfaces.RealOutput yExhDamPos(
     final unit="1",
     min=0,
     max=1)
    "Exhaust damper control signal (0: closed, 1: open)"
    ∃ ;

  Buildings.Controls.OBC.CDL.Continuous.MovingMean movMea(
    delta=300)
    "Average building static pressure measurement"
    ∃ ;
...
equation
  connect(dpBui, movMea.u)
    ∃ ;
...
∃
end ExhaustDamper;
```

CDL Development Process

Various use cases have been developed with industry partners

- 4.1. Controls Design
  - 4.1.1. Loading a Standard Sequence from a Library
  - 4.1.2. Customizing a Control Sequence for an HVAC System
  - 4.1.3. Customizing and Configuring a Control Sequence for a Single-Zone VAV System
  - 4.1.4. Customizing and Configuring a Control Sequence for a Multizone VAV System
  - 4.1.5. Performance Assessment of a Control Sequence
  - 4.1.6. Defining Integration with non-HVAC Systems such as Lighting, Façade and Presence Detection
- 4.2. Bidding and BAS Implementation
  - 4.2.1. Generate Control Point Schedule from Sequences
- 4.3. Commissioning, Operation, and Maintenance
  - 4.3.1. Conducting Verification Test of a VAV Cooling-Only Terminal Unit
  - 4.3.2. As-Built Sequence Generator

Source: https://obc.lbl.gov/specification/useCases.html

Requirements have been developed with industry partners, and are all realized in CDL

1. The CDL shall be **declarative**.
2. CDL shall be able to express control sequences and their **linkage** to an object model which represents the plant.
3. CDL shall represent control sequences as a set of **blocks** (see Section 7.5) with inputs and outputs through which blocks can be connected.
4. It shall be possible to **compose blocks** hierarchically to form new blocks.
5. The **elementary building blocks** [such as a gain] are defined through their input, outputs, parameters, and their response to given outputs. The actual implementation is not part of the standard [as this is language dependent].
6. Each block shall have **tags** that provide information about its general function/ application [e.g. this is an AHU control block] and its specific application [e.g. this particular block controls AHU 2].
7. It shall be possible to identify whether a block represents a **physical** sensor/actuator, or a **logical** signal source/ sink. [Needed for pricing.]

Source: http://obc.lbl.gov/specification/requirements.html#cdl

Requirements have been developed with industry partners, and are all realized in CDL

8. Blocks and their inputs and outputs shall be allowed to contain **metadata.** The metadata shall identify expected characteristics, including but not limited to the following.

For inputs and outputs:

- units,
- a quantity [such as “return air temperature” or “heating requests” or “cooling requests”],
- analog or digital input or output, and
- for physical sensors or data input, the application (e.g. return air temperature, supply air temperature).

For blocks:

- an equipment tag [e.g., air handler control],
- a location [e.g., 1st-floor-office-south], and
- if they represent a sensor or actuator, whether they are a physical device or a software point. [For physical sensors, the signal is read by a sensor element, which converts the physical signal into a software point.]

9. It shall be possible to **translate** control sequences that are expressed in the CDL to implementation of major control vendors.

Source: http://obc.lbl.gov/specification/requirements.html#cdl

Requirements have been developed with industry partners, and are all realized in CDL

10. It shall be possible to **render** CDL-compliant control sequences in a visual editor and in a textual editor.

11. CDL shall be a **proper subset of Modelica 3.3** [Mod12]. [Section Control Description Language specifies what subset shall be supported. This will allow visualizing, editing and simulating CDL with Modelica tools rather than requiring a separate tool. It will also simplify the integration of CDL with the design and verification tools, since they use Modelica.]

12. It shall be possible to **simulate CDL-compliant control sequences in an open-source, freely available Modelica environment.**

13. It shall be possible to simulate CDL-compliant control sequences in the **Spawn of EnergyPlus.**

14. The object model must be **rigorous**, **extensible** and **flexible.**

Source: http://obc.lbl.gov/specification/requirements.html#cdl

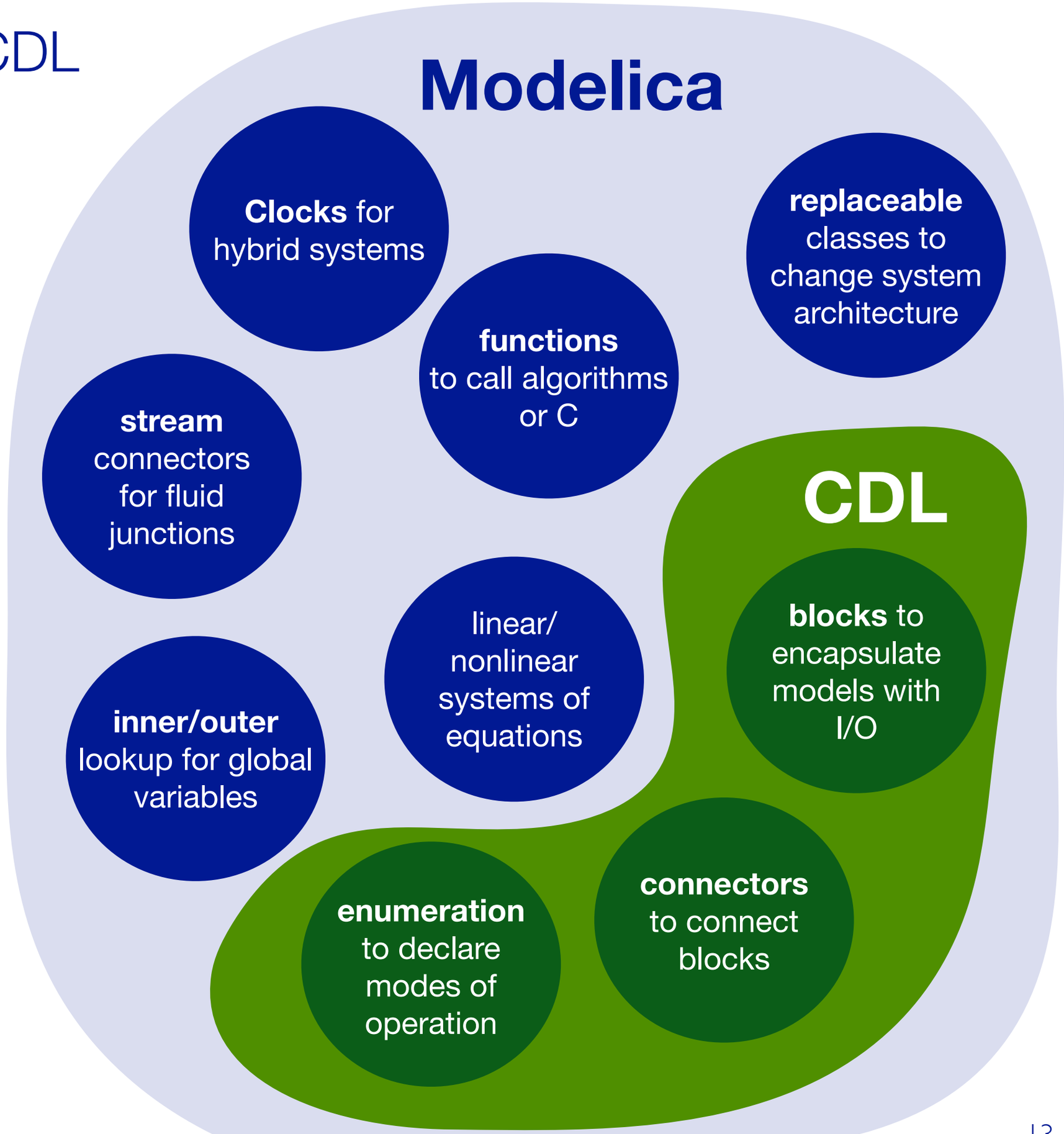
How we specified CDL

Conform to the Modelica Standard 3.3,
but remove everything that is not needed to practically declare control logic and their English language documentation.

Keep it simple & easy to parse.

… and allow reuse of technology from the Modelica ecosystem.

Reviewed by advisory panel, through peer-review process, and used by various project partners from industry.



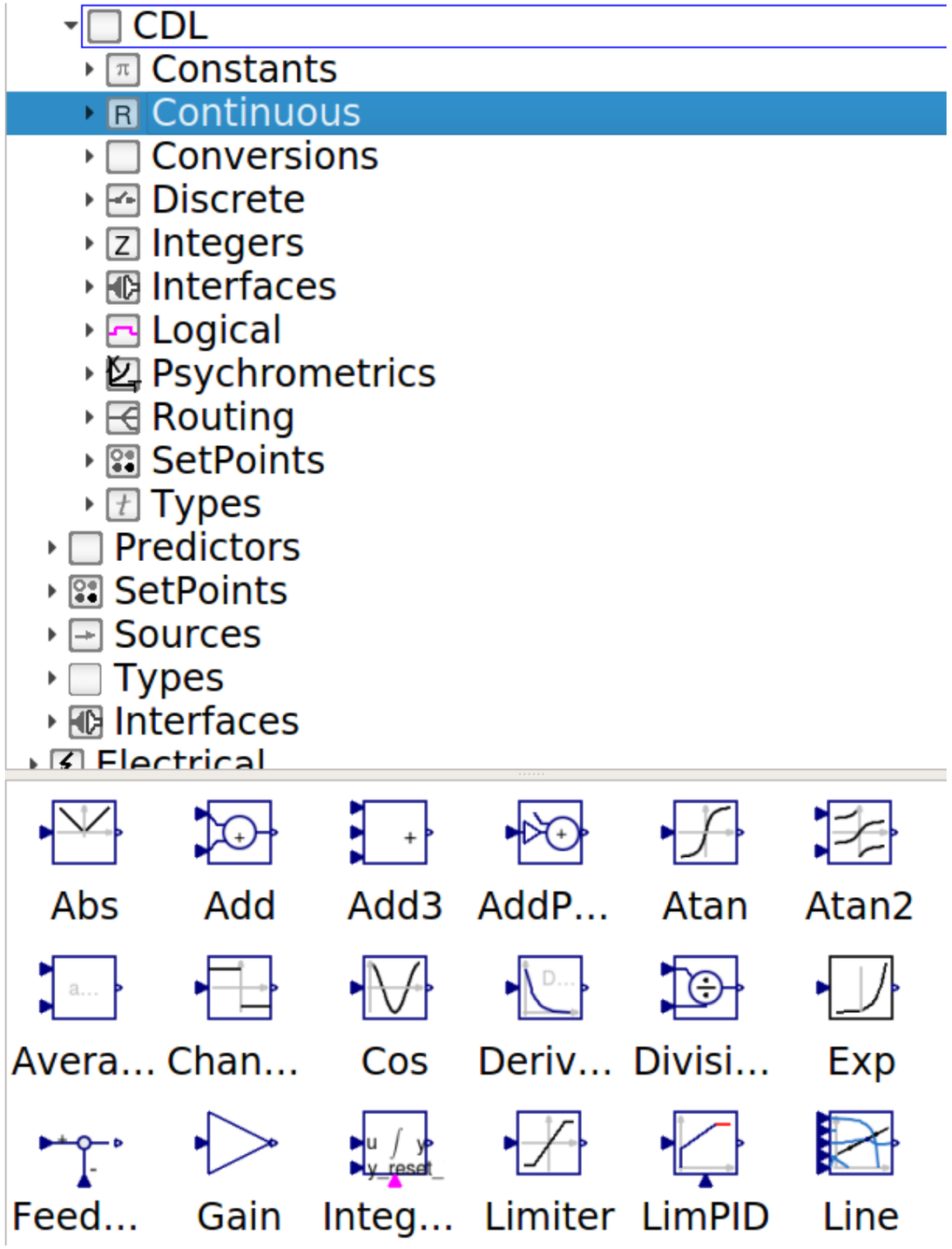
CDL Specification

CDL Specification

Available at https://obc.lbl.gov/specification/cdl.html, and linked from the slides that follow.

Convention: Text in […] is informative.

Basic elementary blocks are defined in a library that is immutable to the users (fixed by the specification).



See https://obc.lbl.gov/specification/cdl.html#elementary-building-blocks

Basic elementary blocks are defined in a library that is immutable to the users (fixed by the specification).

Behavior is expressed mathematically,

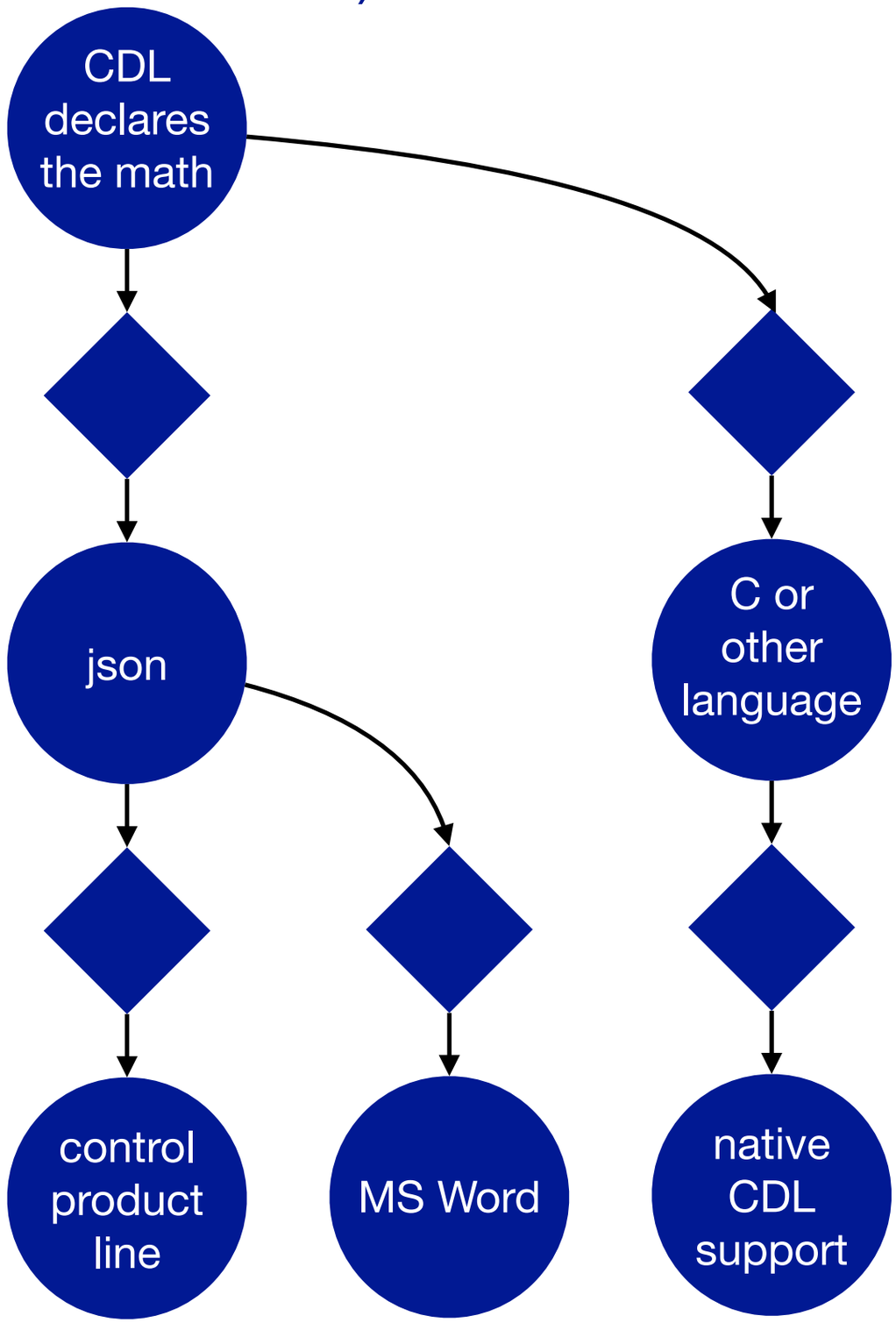
$$(p, t, u(t), x(t)) \mapsto y(t).$$

Software implementation is not part of the specification, as it should not be part of the standard.

Note: This implies that

- the implementation can be graphical or text-based.
- CDL-compliant sequences can be executed as functions (C, Java, Python, Julia, …)
- The controller product line does not need to know anything about CDL or Modelica.

Control providers who support CDL need to be able to implement the same functionality as is defined by the elementary CDL blocks.



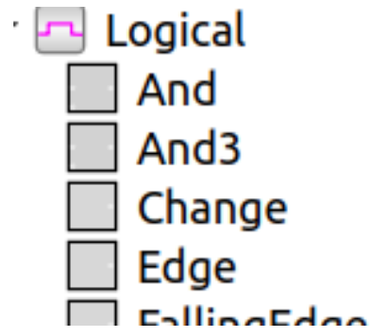
See https://obc.lbl.gov/specification/cdl.html#elementary-building-blocks

Basic elementary blocks are defined in a library that is immutable to the users (fixed by the specification).

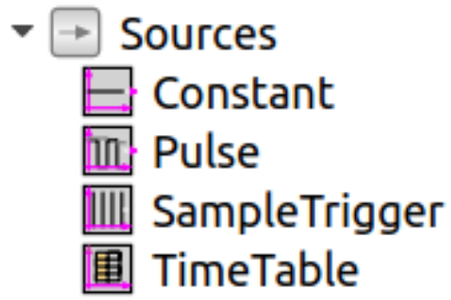
Basic blocks are organized in packages

- Constants
- Continuous
- Conversions
- Discrete
- Integers
- Logical
- Psychrometrics
- Routing
- SetPoints
- Utilities
- Types
- Interfaces

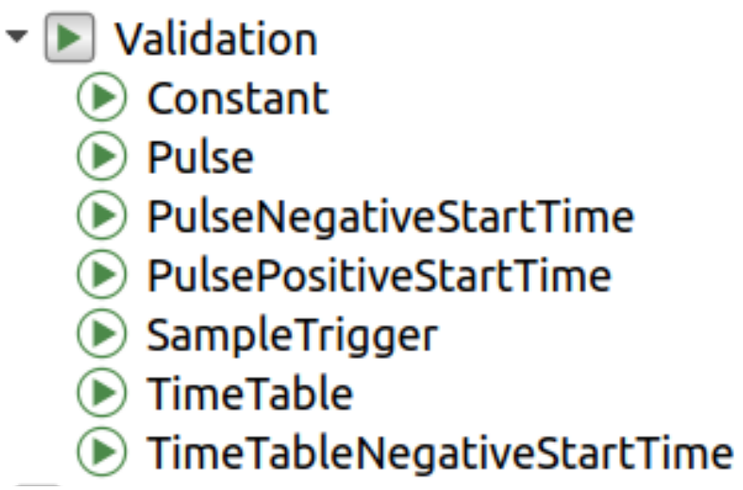
`blocks` are in `packages`:



Signal sources are in `Sources` sub package:



Blocks for validation and CI tests are in `Validation` package

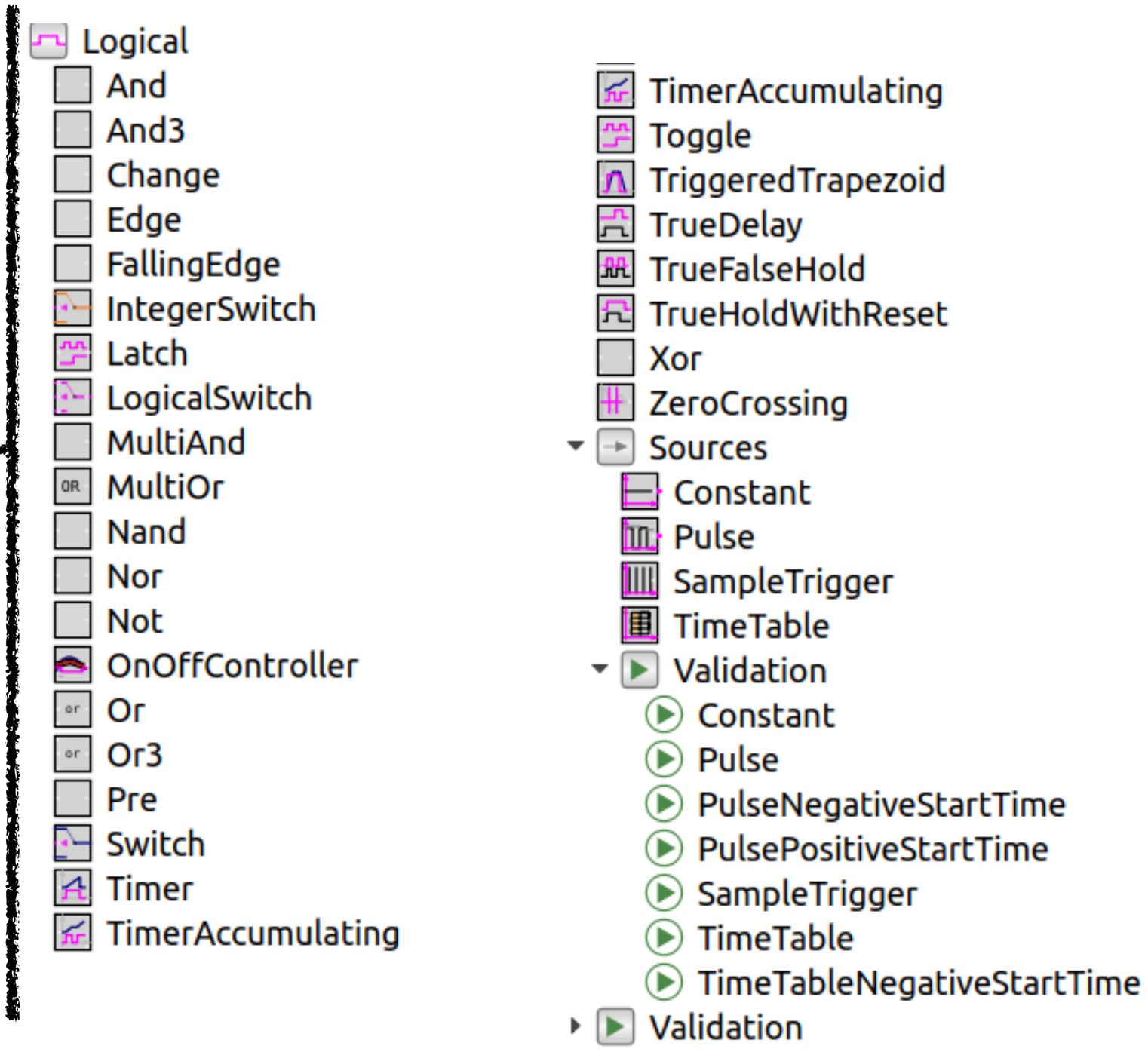


See https://obc.lbl.gov/specification/cdl.html#elementary-building-blocks

Basic elementary blocks are defined in a library that is immutable to the users (fixed by the specification).

Basic blocks are organized in packages

- Constants
- Continuous
- Conversions
- Discrete
- Integers
- Logical
- Psychrometrics
- Routing
- SetPoints
- Utilities
- Types
- Interfaces



See https://obc.lbl.gov/specification/cdl.html#elementary-building-blocks

Demo of some elementary blocks.

Review of elementary blocks

Elementary control blocks have been reviewed with industry, used to implement many sequences from Guideline 36, RP 1711 and other sources, *but*

We want 231P committee to review and decide

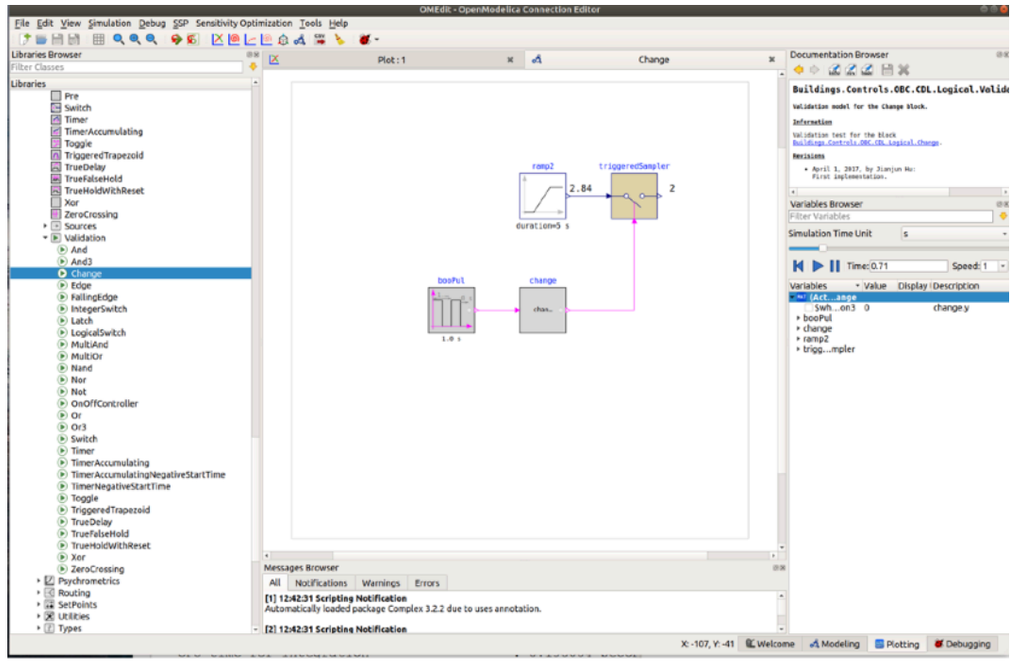
- are these the right blocks,
  - which ones are missing?
  - which ones are not needed?
  - which ones do you see problematic when mapping to an equivalent in your product line?

See https://obc.lbl.gov/specification/cdl.html#elementary-building-blocks

Review of elementary blocks

To review blocks, use any of these:

- html at https://simulationresearch.lbl.gov/modelica/releases/dev-2020-10-27/help/Buildings_Controls_OBC_CDL.html
- Modelica environments:
  Download https://github.com/lbl-srg/modelica-buildings/archive/master.zip
  and browse Buildings.Controls.OBC.CDL using either:
  - Dymola demo version:
    https://www.3ds.com/products-services/catia/products/dymola/trial-version/
  - OpenModelica's OMEdit:
    https://openmodelica.org.
    Note: To correctly render icons,
    open OMEdit, select "Tools -> Options -> Simulation" and uncheck box "Enable new frontend use in the OMC API (faster GUI response)".
    To be fixed in https://trac.openmodelica.org/OpenModelica/ticket/5631



Basic types

CDL supports

- Real
- Integer
- Boolean
- String
- Enumeration, can be used, e.g., to introduce modes as

```
type SystemMode = enumeration(
  Heating “Heating mode”,
  Cooling “Cooling mode”,
  Off     “System off”)
  “Allowed modes of operation”;
```

Types variability can be

- parameter (user-changeable)
  `parameter Real k(min=0) = 1 “Proportional gain";`
- constant (fixed during translation)
  `constant Real pi = 3.14159;`

Arrays (used sparingly as many BAS don’t support arrays).

- Arrays can be flattened during translation, e.g.,
  `parameter Real[2] x = {0, 1};`
  can be converted to
  `x_1 = 0` and `x_2 = 1`.
- Array size is fixed during program execution, allowing
  - predictable memory requirement
  - statically guaranteed array access

See https://obc.lbl.gov/specification/cdl.html#permissible-data-types

Instantiation declares constants, parameters and blocks by reference

Parameter assignments, e.g.,

```
CDL.Continuous.Gain gai(k=-1) "Constant gain of -1";
```

- Can involve simple calculation (can optionally be evaluated during translation).

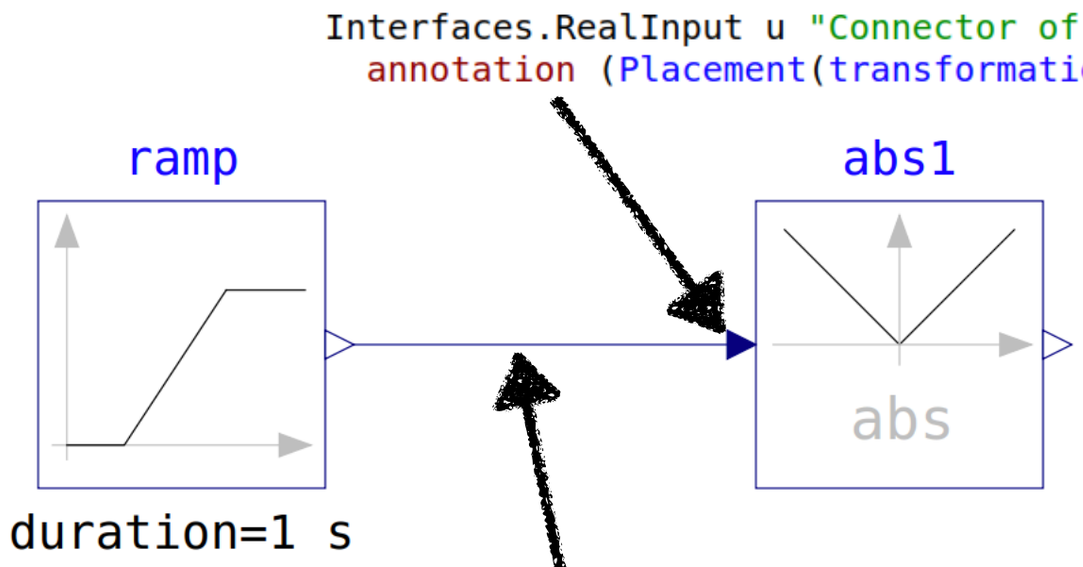
Instances can, *optionally*, be conditionally removed during translation:

```
parameter Boolean have_occSen=false
  "Set to true if zones have occupancy sensor";

CDL.Interfaces.IntegerInput nOcc if have_occSen
  "Number of occupants"
    annotation (__cdl(default = 0));

CDL.Continuous.Gain gai(
  k = VOutPerPer_flow) if have_occSen
    "Outdoor air per person";
equation
connect(nOcc, gai.u);
```

If you want to keep this input during the translation, then use this value for the unconnected input.

See https://obc.lbl.gov/specification/cdl.html#instantiation

Connectors connect inputs to outputs




```
within Buildings.Controls.OBC.CDL.Continuous.Validation;
model Abs "Validation model for the absolute block"

  Buildings.Controls.OBC.CDL.Continuous.Abs abs1
    "Block that outputs the absolute value of the input"
    annotation (Placement(transformation(extent={{-10,-10},{10,10}})));

  Buildings.Controls.OBC.CDL.Continuous.Sources.Ramp ramp(
    height=2,
    duration=1,
    offset=-1) "Block that generates ramp signal"
    annotation (Placement(transformation(extent={{-60,-10},{-40,10}})));

equation
  connect(ramp.y, abs1.u)
    annotation (Line(points={{-38,0},{-38,0},{-12,0}}, color={0,0,127}));

  annotation ( ...);

end Abs;
```

Contract: All inputs must be connected to exactly one output (single assignment rule).

Semantics: `ramp.y = abs1.u` at any time.

See https://obc.lbl.gov/specification/cdl.html#connectors

Annotations

Used to specify items that do not affect the calculations, such as

- graphical rendering,
- tags,
- sequence documentation,
- revision notes,
- …

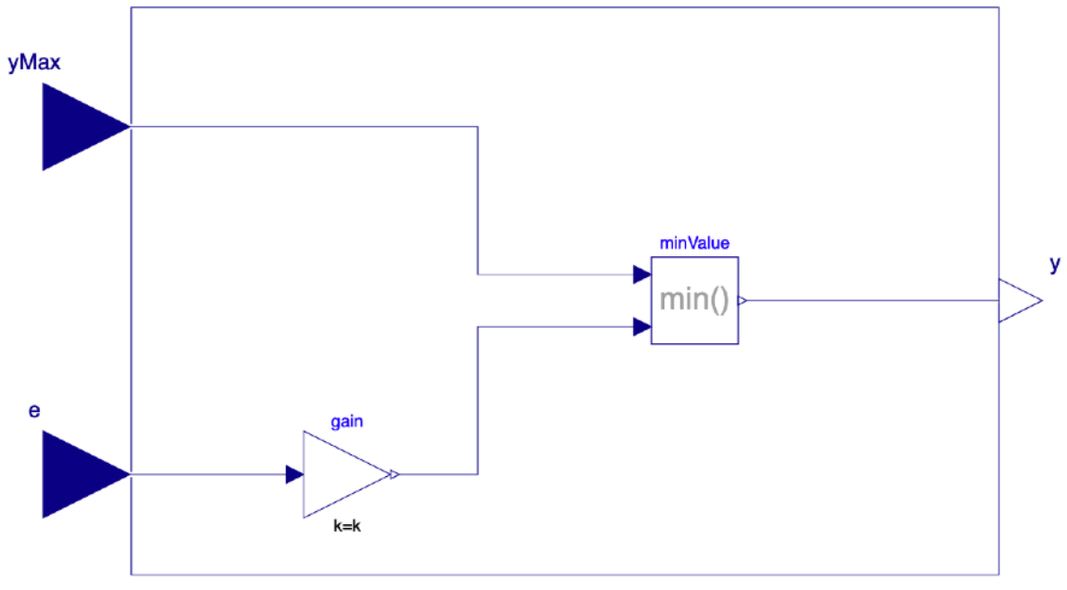
```
within Buildings.Controls.OBC.CDL.Continuous;
block Abs "Output the absolute value of the input"

  Interfaces.RealInput u "Connector of Real input signal"
    annotation (Placement(transformation(extent={{-140,-20},{-100,20}})));

  Interfaces.RealOutput y "Connector of Real output signal"
    annotation (Placement(transformation(extent={{100,-20},{140,20}})));

equation
  y = abs(u);

annotation (
    defaultComponentName="abs",
    Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{100,
          100}}), graphics={
        Text(
          lineColor={0,0,255},
          extent={{-150,110},{150,150}},
          textString="%name"),
        Rectangle(
        extent={{-100,-100},{100,100}},
        lineColor={0,0,127},
        fillColor={255,255,255},
        fillPattern=FillPattern.Solid),
        Polygon(
          points={{92,0},{70,8},{70,-8},{92,0}},
          lineColor={192,192,192},
          fillColor={192,192,192},
          fillPattern=FillPattern.Solid),
        Line(points={{-80,80},{0,0},{80,80}}),
        Line(points={{0,-14},{0,68}}, color={192,192,192}),
        Polygon(
          points={{0,90},{-8,68},{8,68},{0,90}},
          lineColor={192,192,192},
          fillColor={192,192,192},
          fillPattern=FillPattern.Solid),
        Text(
          extent={{-34,-28},{38,-76}},
          lineColor={192,192,192},
          textString="abs"),
        Line(points={{-88,0},{76,0}}, color={192,192,192}),
        Text(
          extent={{226,60},{106,10}},
          lineColor={0,0,0},
          textString=DynamicSelect("", String(y, leftjustified=false, significantDigits=3)))}),
    Documentation(info="<html>
<p>
Block that outputs <code>y = abs(u)</code>,
where
<code>u</code> is an input.
</p>
</html>", revisions="<html>
<ul>
<li>
March 2, 2020, by Michael Wetter:<br/>
Changed icon to display dynamically the output value.
</li>
<li>
January 3, 2017, by Michael Wetter:<br/>
First implementation, based on the implementation of the
Modelica Standard Library.
</li>
</ul>
</html>"));
end Abs;
```

See https://obc.lbl.gov/specification/cdl.html#annotations

Composite blocks compose sequences

Used to implement sequences.

Hierarchical composition allows refinement of sequence architecture.




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

  parameter Real k "Constant gain";

  CDL.Interfaces.RealInput yMax "Maximum value of output signal"
    annotation (Placement(transformation(extent={{-140,20},{-100,60}})));

  CDL.Interfaces.RealInput e "Control error"
    annotation (Placement(transformation(extent={{-140,-60},{-100,-20}})));

  CDL.Interfaces.RealOutput y "Control signal"
    annotation (Placement(transformation(extent={{100,-10},{120,10}})));

  CDL.Continuous.Gain gain(final k=k) "Constant gain"
    annotation (Placement(transformation(extent={{-60,-50},{-40,-30}})));

  CDL.Continuous.Min minValue "Outputs the minimum of its inputs"
    annotation (Placement(transformation(extent={{20,-10},{40,10}})));
equation
  connect(yMax, minValue.u1) annotation (
    Line(points={{-120,40},{-120,40},{-20,40},{-20, 6},{18,6}},
         color={0,0,127}));
  connect(e, gain.u) annotation (
    Line(points={{-120,-40},{-92,-40},{-62,-40}},
         color={0,0,127}));
  connect(gain.y, minValue.u2) annotation (
    Line(points={{-39,-40},{-20,-40},{-20,-6}, {18,-6}},
         color={0,0,127}));
  connect(minValue.y, y) annotation (
     Line(points={{41,0},{110,0}},
          color={0,0,127}));

  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;
```

See https://obc.lbl.gov/specification/cdl.html#composite-blocks

Model of Computation —
What does it mean to connect inputs and outputs?
What happens if time is advanced?
How do we ensure determinism and predictable timing?

From Modelica's model of computation:

- Values are **persistent** until changed explicitly.
- Values can be **accessed** at any time instant.
- Computation and communication at an event instant **does not take time**. [If computation or communication time has to be simulated, this property has to be explicitly modeled.]
- Every input connector shall be **connected** to exactly one output connector.

In addition, we further restrict:

- The dependency graph from inputs to outputs that directly depend on inputs shall be **directed** and **acyclic**.

  I.e., **no algebraic loops** are not allowed. (As this behavior is not predictable.)
- Values are always **present** (unlike some event-based systems.)

See https://obc.lbl.gov/specification/cdl.html#model-of-computation

Tagged properties

CDL is extensible to allow for example tags for semantic models:

```
annotation :
  annotation "(" [annotations ","]
     __cdl "(" [ __cdl_annotation ] ")" ["," annotations] ")"
```

where for Brick, `__cdl_annotation` is

```
brick_annotation:
   brick "(" RDF ")"
```

For example, for Brick:

```
annotation(__cdl(brick="soda_hall:flow_sensor_SODA1F1_VAV_AV a brick:Supply_Air_Flow_Sensor  ;
  bf:hasTag brick:Average ;
  bf:isLocatedIn soda_hall:floor_1 ."));
```

For example, for Haystack:

```
annotation(__cdl( haystack=
  "{\"id\"   : \"@whitehouse.ahu3.dat\",
    \"dis\" : \"White House AHU-3 DischargeAirTemp\",
    \"point\" : \"m:\",
    \"siteRef\" : \"@whitehouse\",
    \"equipRef\" : \"@whitehouse.ahu3\",
    \"discharge\" : \"m:\",
    \"air\"       : \"m:\",
    \"temp\"      : \"m:\",
    \"sensor\"    : \"m:\",
    \"kind\"       : \"Number\"
    \"unit\"       : \"degF\"}"));
```

See https://obc.lbl.gov/specification/cdl.html#tags

Current use of CDL

Tools and projects that use CDL as currently defined

Model authoring

- Linkage
  https://github.com/lbl-srg/linkage.js
  - LBL issued subcontract to develop control sequence selection and configuration tool tailored to HVAC industry
  - Based on CDL
  - Will output sequence description, point list, simulation models, etc.
- any Modelica-compliant tool (OpenModelica, IMPACT, Dymola, SimulationX, MapleSim, SystemModeler, SimCenter AmeSim, ANSYS Twin Builder, …)

Model translation

- Modelica-json
  https://github.com/lbl-srg/modelica-json
  - Parser from CDL to JSON, html, Word
  - Can output intermediate format, used for example to generate ALC EIKON code
  - Can generate point lists

Sequence implementations

- Modelica Buildings Library
  https://simulationresearch.lbl.gov/modelica/
  - Currently has ASHRAE Guideline 36 PR draft 1 models
  - Work in process for
    - ASHRAE Guideline 36 final release
    - Chiller plants from RP-1711
    - Boiler plants from RP-1711 (from PNNL)
    - Radiant systems (from Integral Group)
    - Natural ventilation (from Integral Group)
    - DOAS (from Facility Dynamics)
    - Packaged RTUs (from Facility Dynamics)

Verification and performance assessment

- Spawn of EnergyPlus
  https://lbl-srg.github.io/soep/
- Any tool that supports the Modelica or FMI standards, or
- through translation, other simulators

Foundational Standards

Foundational Standards – Modelica

https://modelica.org/

Open, industry-driven standard for multi-physics modeling

Developed since 1996

Large ecosystem of free and commercial libraries and tools

Large national and international collaborations on Modelica for building and district energy systems:

- IEA EBC Annex 60:
  42 institutes (2012-2017)
- IBPSA Project 1:
  >80 FTE, 29 institutes + 51 individual participants
  (2017-2022)
- Spawn of EnergyPlus:
  DOE funded with lab/industry to add Modelica-based control capabilities to EnergyPlus

Why standards?

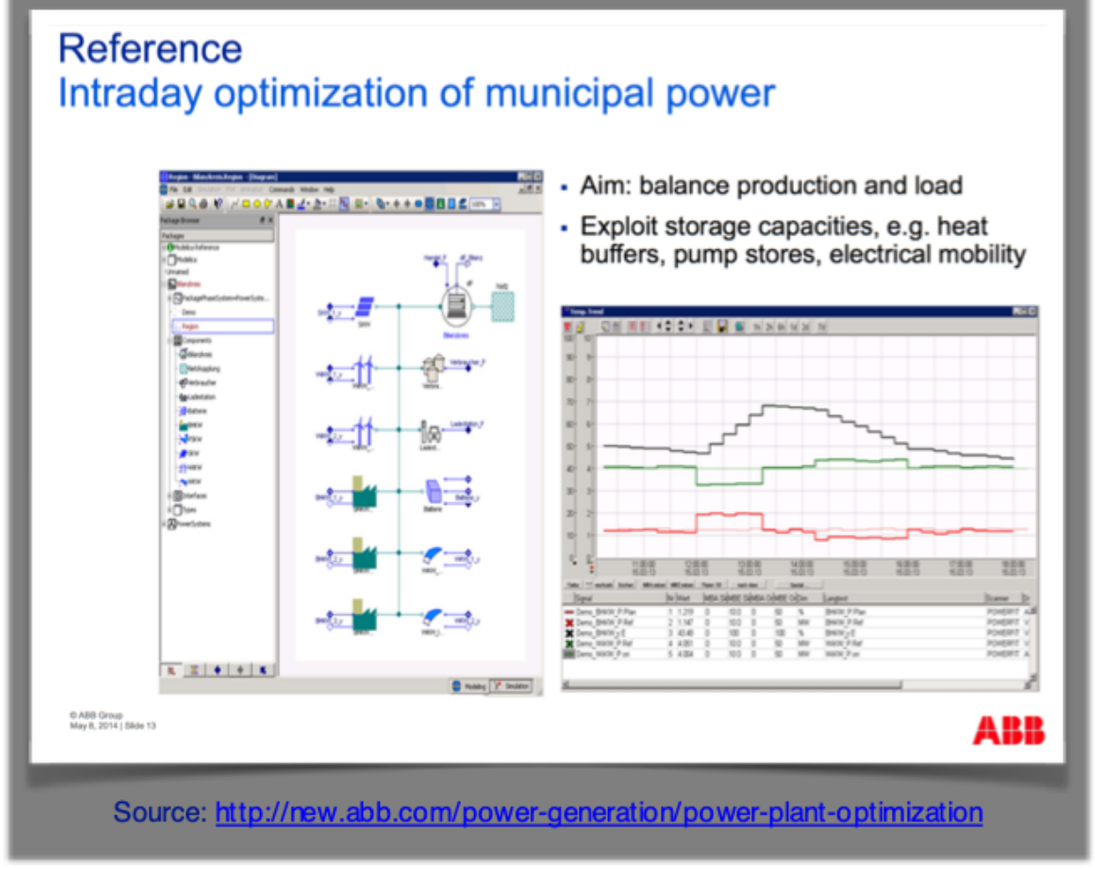
Leverages investments in related industries.

Provides well-tested APIs for software integration.

Provides to industry a stable basis for investment.

Avoids vendor lock-in.

7% of German power production is optimized based on Modelica



Source: http://new.abb.com/power-generation/power-plant-optimization

Related standards

https://www.fmi-standard.org/
API standard to exchange simulators or models

Developed within MODELISAR, an ITEA2 project to improve significantly the design of systems and of embedded software in vehicles.

ITEA project, 28 partners, 178 person years, 26 Mill. € budget, July 2008 - June 2011.

First version published in 2010.

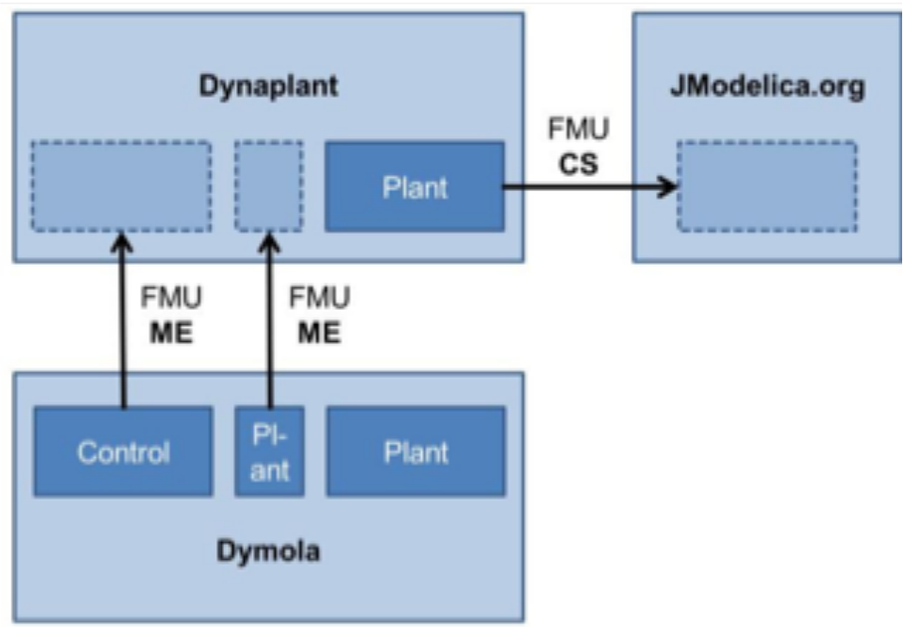
Supported by >100 tools.

https://ssp-standard.org/

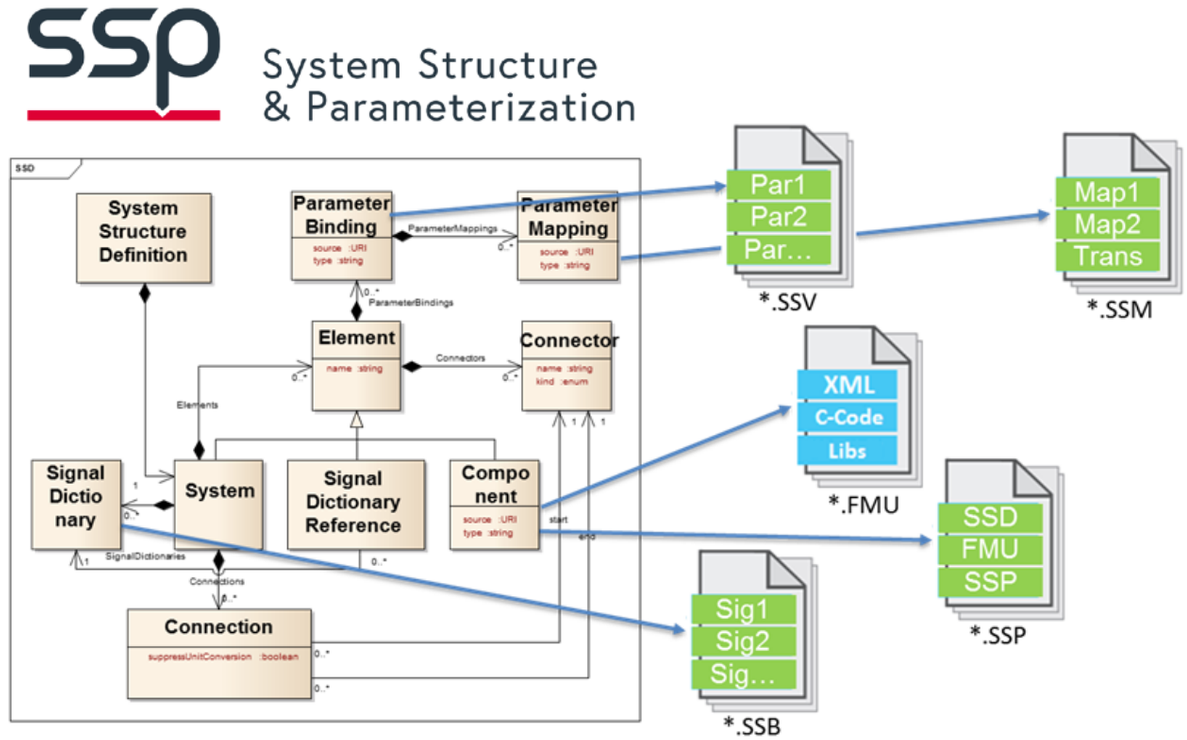
System Structure and Parameterization (SSP)

Tool independent standard to define complete systems consisting of one or more FMUs, including its parameterization that can be transferred between tools.

First version published in 2019.



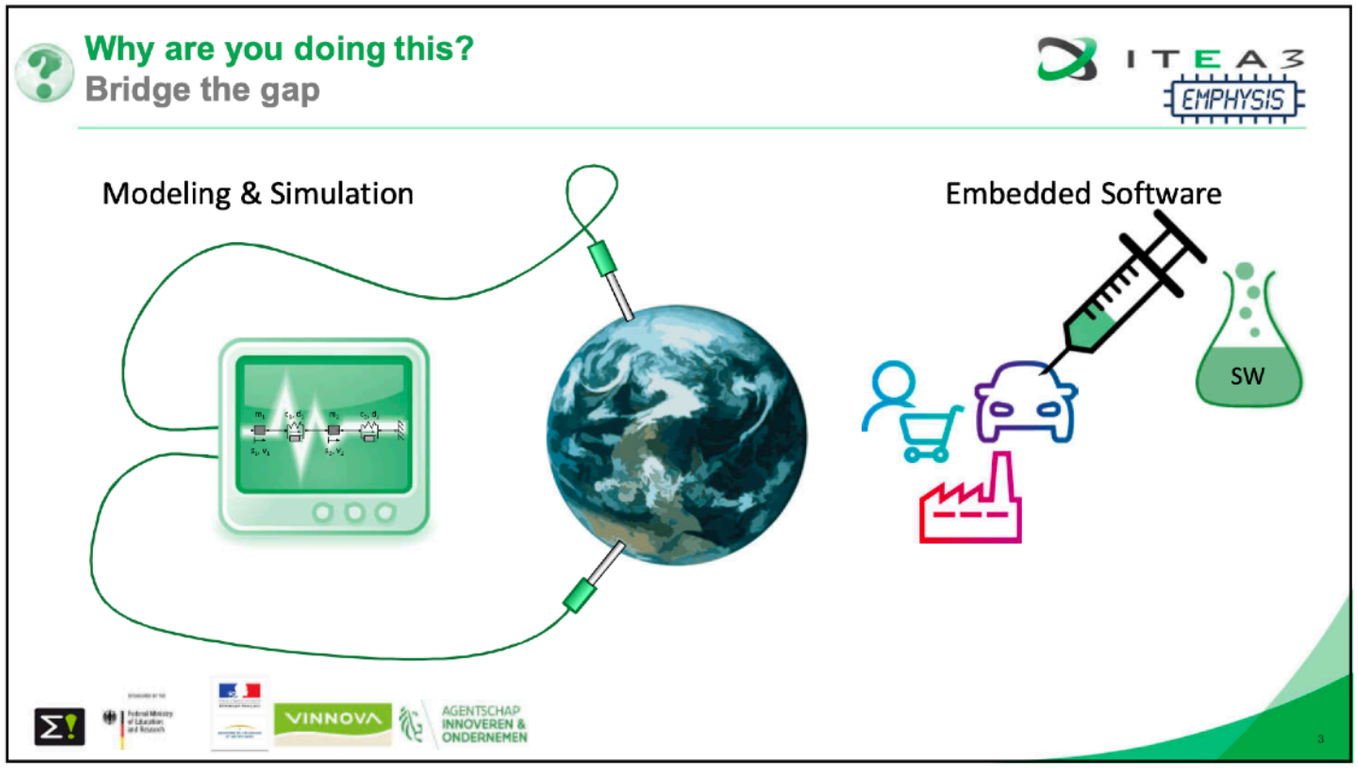
Powerplant simulation with Modelica (Dymola) coupled to in-house simulator (Dynaplant). Source: Siemens, doi:10.3384/ecp1511817

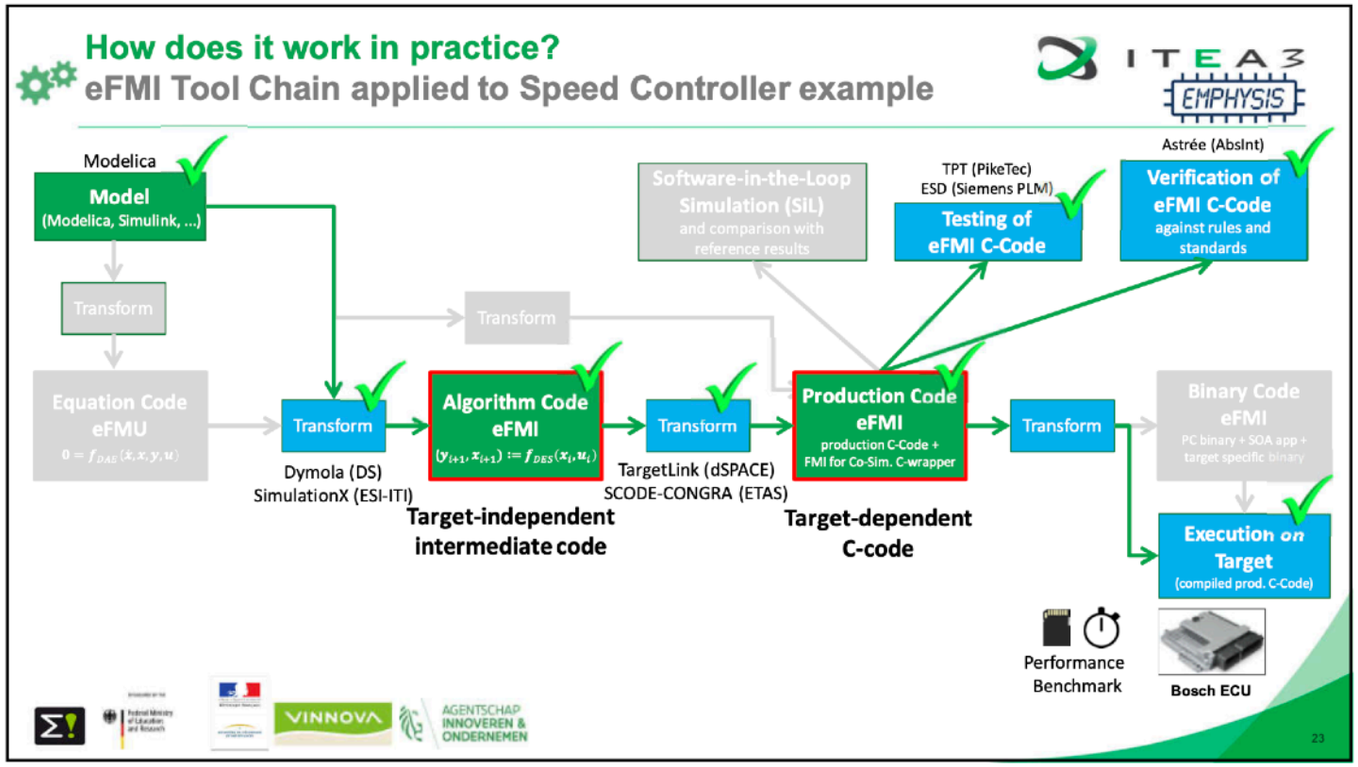


Related standards

eFMI
Expected early 2021.

https://modelica.github.io/Symposium2019/slides/jubilee-symposium-2019-slides-lenord.pdf





Why are these standards important?

Robust, rigorous, well-defined basis, no need to reinvent the wheel.

Provides stable basis for industry investment.

Avoids vendor lock-in and single dependency on a provider.

Ecosystem of developers and users provides tools for

- sequence authoring
- simulation for
  - control sequence development and testing and
  - performance assessment with energy model in the loop
- documentation
- code generation (for various platforms)
- translation to various other formats

Demo

Questions

Comments

Next steps