Research Paper

An Extension of the Systems Modeling Language for Physical Interaction and Signal Flow Simulation

Conrad Bock,[1],* Raphael Barbau,[2] Ion Matei,[3] and Mehdi Dadfarnia[1]

[1]U.S. National Institute of Standards and Technology, 100 Bureau Dr, Stop 8263, Gaithersburg, MD 20899
[2]Engisis, LLC, 10411 Motor City Dr, Ste 750, Bethesda, MD 20817
[3]Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304

Received 30 May 2013; Revised 17 January 2017; Accepted 13 March 2017, after one or more revisions
Published online in Wiley Online Library (wileyonlinelibrary.com).
DOI 10.1002/sys.21380

ABSTRACT

Computer-interpretable representations of system structure and behavior are at the center of developing today's complex systems. Systems engineers create and review these representations using graphical modeling languages that capture requirements, designs, and tests (such as the Systems Modeling Language, SysML). However, these languages must be used in conjunction with analysis tools, in particular, with simulators for physical interaction and numeric signal flow based on ordinary and algebraic differential equation solvers. These kind of simulation tools are often used separately from system modeling tools, leading to inconsistencies that require additional work to eliminate, preventing multidisciplinary concerns from being reflected in the overall system design. As a result, there is an increasing need for integrating physical interaction and signal flow simulation tools and languages into system modeling under a single framework. In this article, we first present an abstraction of the constructs and semantics these simulation tools and languages have in common, based on earlier reviews. Then, we compare SysML to our simulation abstraction to find the parts of SysML closest to simulation modeling, and to identify simulation concepts missing from SysML. This leads to extensions of SysML to bridge the gaps, illustrated with an example application. Next, we address issues in translating extended SysML models to common simulation tools and languages, including the differences between them. Finally, we validate the approach by applying the extension to an example SysML model, automating the translations in software, and showing that the results execute the same way on different simulation platforms. © 2017 Wiley Periodicals, Inc. Syst Eng 20: 395–431, 2017

Key words: SysML; Analysis; Lumped parameter; Modelica; Simulink/Simscape

1. INTRODUCTION

System modeling helps system engineers coordinate the work of multiple other engineering disciplines (mechanical, material, electrical, software, and so on), many of which use simulation models to specify system structures and predict their behavior [van den Bosch and van den Klauw, 1994; Fritzon, 2011]. System modeling also specifies systems structures and behaviors, but is often done separately from simulation. This forces engineers to specify portions of their systems in simulation that are often already available in system models, and vice versa, leading to inconsistencies that require additional work to eliminate. System and simulation models

*Author to whom all correspondence should be addressed (e-mail: conrad.bock@nist.gov).

must be integrated to reduce these inefficiencies and ensure multidisciplinary concerns are reflected in the overall system design.

Some simulation tools present graphical interfaces for linking system components specified by ordinary and algebraic differential equations (derivatives of functions of one variable), which are applicable to a wide range of physical interactions between components (such as mechanical, electrical, and so on) as well as communication of numeric signals [Controllab Products, 2015; The MathWorks, Inc, 2016a; Open Source Modelica Consortium, 2016]. These linked components are referred to as *physical interaction and signal flow* models in this article (also known as lumped parameter, one-dimensional, or network models). Tools for this kind of simulation generate additional equations from links between components, solve them, and report the results as graphs of system property values over time. Simulation in the rest of this article refers to this particular kind of simulation.

Behind their graphical interfaces, physical interaction and signal flow simulation tools treat all engineering disciplines the same way. They achieve this through commonalities in the underlying physics, which are all based on exchange of physical substances in terms of their conserved characteristics (electric charge, momentum, entropy, and so on), without regard to the particular kind of substance [Paynter 1960; Cellier, Elmqvist, and Otter, 1999]. The mathematics of lumped parameter systems is the same across engineering disciplines, and the equations can be solved by the same algorithms. Simulation tools also leverage similarities between numeric signal flow and potential for physical flow, which have the same values on both ends of links between system components. Simplified physical equations can be used for signal flow and solved by simplified algorithms.

Graphical interfaces presented by these kind of simulators express concepts similar to the Systems Modeling Language (SysML), an extension of the Unified Modeling Language (UML) for systems engineering [Object Management Group, 2017a; Object Management Group, 2015a].[1] Simulation tools and SysML show system components and their interconnections, and how physical substances and information flow between components. The graphics of these simulators use symbols and images specific to each engineering discipline, while SysML uses symbols that are not discipline specific. SysML and these simulators have underlying textual languages to record models in computer-processable file formats. Simulators translate graphical models into file-based formats, which are transformed into equations for solution by numerical analysis. SysML-based tools use their file-based formats as input to other kinds of analysis and verification, such as checking completeness of designs against requirements.

Despite the similarities in modeling approaches and architecture of SysML modeling tools and physical interaction and signal flow simulators, these simulators are typically used separately from SysML tools, leading to the problems described earlier. Others have noticed these similarities, proposing integrations between SysML and one simulation tool or language (*simulation platform*), as described in Section 2. This forces engineers to respecify simulation-specific information for each platform they use, even though these platforms have many more modeling capabilities in common than they have differences. Single platform integrations also tend to surface the platform to systems models without adaptation or simplification, making them more complicated than necessary for systems engineering.

Our approach to integrating SysML and physical interaction and signal flow simulation is to identify modeling capabilities in common between widely used simulation platforms, compare these with SysML, and extend SysML with only the simulation modeling capabilities that SysML does not already have. This enables redundant elements in systems models and simulation platforms to be specified once, rather than manually recoded for each platform, simplifying translation and synchronization between SysML and multiple simulation platforms. This provides more efficient integration of systems engineering models and processes with physical interaction and signal flow simulation than platform-specific approaches [Dadfarnia, Bock, and Barbau, 2016].

The structure of this article is as follows. Section 2 covers related work, showing it does not address platform-independent physical interaction and signal flow simulation modeling in SysML. Section 3 gives an overview of this kind of simulation, presenting an abstraction of concepts that widely used platforms have in common, based on earlier reviews. Section 4 compares this abstraction with SysML concepts, identifying that are equivalent or close to equivalent. Based on this comparison, Section 5 presents an extension of SysML that reuses SysML concepts where they are equivalent to simulation concepts, and extends them where they are close, to match the simulation abstraction developed in Section 3. Section 6 gives an example of modeling a system using the extension. Section 7 addresses issues in translating between extended SysML models and simulation platforms, including differences between them, illustrated with the example in Section 6. Section 7 also describes a publicly available implementation of the SysML extension and platform translations, using it for validation on the example. Section 8 summarizes the article and discusses future work.[2]

2. RELATED WORK

Integration efforts for multiple simulation platforms typically only address interactions occurring while simulations are carried out, rather than during development of simulation models, as in this article. For example, High-Level Architecture, Simulation Modeling Platform, and Functional Mockup Interface define interfaces and services that enable simulators of many kinds to be operated with others at the same time [IEEE Standards Association, 2010; European Cooperation for Space Standardization, 2011; Modelica Association, 2014a]. These standards give simulators access to each other's variable values during simulation, notifications of events as they occur, and to some other aspects of simulation, such as time step size. They are not concerned with interoperation of simulation modeling platforms or integration with systems engineering.[3]

Integration of SysML parametric diagrams and general equation solvers, such as MATLAB® [The MathWorks, Inc, 2016b], are available in commercial tools. SysML parametrics capture reuse of equations within a system. For example, using the equation F=ma in parametric diagrams specifies which properties of systems or their components correspond to F, m, and a. Parametric diagrams and their tools do not focus on particular kinds of equations, such as differential equations, and consequently are cumbersome to use for physical interaction and signal flow, because they cannot provide abstractions specific to these applications. The same applies to efforts integrating other systems modeling languages specifically with MATLAB [Dori, Renick, and Wengrowicz, 2016].

Some UML-based integration with physical interaction simulation depend on bond graphs [Paynter, 1960], which model energy flows between components independently of whatever is carrying the energy [Secchi, Fantuzzi, and Bonfe, 2005; Turki, Thierry, and Sghaier, 2005]. Popular physical interaction simulation platforms, such as Simscape™ (an extension of Simulink® and MATLAB) and the Modelica® language [Modelica Association, 2014b; The MathWorks, Inc, 2016a; The MathWorks, Inc, 2016c], are adapted from bond graphs [Cellier et al., 1999], and much more widely used industrially. Due to bond graphs' low level of practical usage, they are not considered in this article. The same applies to other efforts based on less widely used simulation languages [Berkenkotter et al., 2006].

Despite the benefits of incorporating common physical interaction and signal flow concepts into system models, most integration efforts between SysML and physical interaction and signal flow simulation platforms focus on one platform, usually Modelica or Simulink. Some authors propose integrating SysML specifically with Simulink either by extending SysML for transformation to Simulink [Kawahara et al., 2009; Liu and Cao, 2010; Snyder, Bocktaels, and Feigenbaum, 2010; Reichwein, 2011; Rahman and Mizukawa, 2013], or with SysML modeling patterns that reproduce Simulink semantics [Bock, 2006; Sjostedt et al., 2008]. Others propose SysML extensions specifically for transformation to Modelica [Nytsch-Geusen, 2007; Schamai et al., 2009; Vasaiely, 2009; Rahman and Mizukawa, 2013], one standardized by the Object Management Group [Paredis et al., 2010; Object Management Group, 2012], or attempt to find SysML modeling patterns that reproduce Modelica semantics [Sjostedt et al., 2007].

One attempt to extend SysML for multiple simulation platforms addresses only one signal flow tool (Simulink), starts its analysis with particular simulation platforms, rather than creating an abstraction of them first, leading to common modeling capabilities only within extensions for specific engineering disciplines [Cao et al., 2013]. In some cases, this results in terminology derived from simulation tools, rather than systems modeling, even though the extension is for the most widely used systems engineering modeling language. This and some of other integration efforts cited above use SysML version 1.2, whereas SysML version 1.3 contains significant upgrades to ports and flow properties that affect integration with simulators.

3. PHYSICAL INTERACTION AND SIGNAL FLOW SIMULATION

This section gives an overview of physical interaction and signal flow simulation (hereafter called *simulation*) through an abstraction of simulation concepts that widely used platforms have in common, based on earlier reviews [Matei and Bock, 2012a] [Matei and Bock, 2012b] (see Section 7 about differences between platforms). The abstraction is compared to SysML in Section 1 to identify SysML concepts most appropriate for supporting this kind of simulation. Sections 3.1 and 3.2 cover structural and behavioral modeling concepts, respectively.

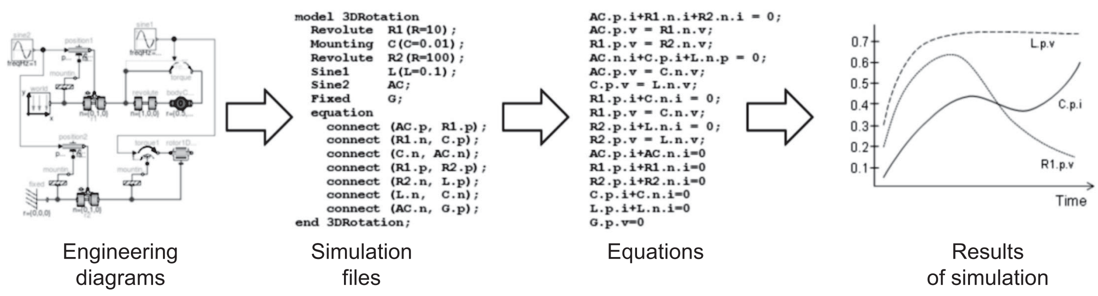
The simulators covered in this section use numerical analysis techniques to solve differential equations [Iserles, 2008], but present equation variables to engineers as if they were properties of system components, rather than showing the variables only in the context of equations. This gives engineers the benefit of automated equation solvers through the more familiar concepts of systems, components, and properties. These concepts are presented in *simulation models*, either graphically using symbols and images from various engineering disciplines, or textually, in simulation languages stored as computer-processable and human-readable files. These files are translated into equations, solved by numerical integration, with results reported back on the values of variables over time. This process is illustrated in Figure 1.

3.1. System Structure in Simulation Modeling

An earlier review of simulation platforms identified similarities in the concepts they use for modeling system structure [Matei and Bock, 2012a]:

- *Components* are system elements that process and exchange physical substances in terms of their conserved characteristics (electric charge, momentum, entropy, and so on) or numeric information (signals) with each other.
- *Ports* are elements of components enabling exchange with other components.
- *Links* are system elements that connect ports across which exchange occurs between components.
- *Properties* of components and ports quantify the processing and exchange of physical substances or numeric information.
- *Subsystems* are components composed of other components, which may be other subsystems. *Atomic components* are not composed of other components.
- *Models* are top-level subsystems or components simulated separately from any other subsystems or components that might use them.

Terms for the concepts above differ between simulation platforms, but the meaning is the same. Simulation platforms sometimes refer to components as blocks, ports as pins or connectors, links as lines, and properties as variables or parameters. The terms above are part of an abstraction over simulation platforms that enable comparison to SysML in Section 4, development of a simulator-independent extension


```
model 3DRotation
  Revolute  R1(R=10);
  Mounting  C(C=0.01);
  Revolute  R2(R=100);
  Sine1     L(L=0.1);
  Sine2     AC;
  Fixed     G;
  equation
    connect (AC.p, R1.p);
    connect (R1.n, C.p);
    connect (C.n, AC.n);
    connect (R1.p, R2.p);
    connect (R2.n, L.p);
    connect (L.n,  C.n);
    connect (AC.n, G.p);
end 3DRotation;
```

```
AC.p.i+R1.n.i+R2.n.i = 0;
AC.p.v = R1.n.v;
R1.p.v = R2.n.v;
AC.n.i+C.p.i+L.n.p = 0;
AC.p.v = C.n.v;
C.p.v = L.n.v;
R1.p.i+C.n.i = 0;
R1.p.v = C.n.v;
R2.p.i+L.n.i = 0;
R2.p.v = L.n.v;
AC.p.i+AC.n.i=0
R1.p.i+R1.n.i=0
R2.p.i+R2.n.i=0
C.p.i+C.n.i=0
L.p.i+L.n.i=0
G.p.v=0
```

Figure 1. Simulation tool process.

of SysML in Section 5, and translation from SysML to simulation platforms in Section 7.

Simulation modeling tools typically provide a graphical interface showing components and subsystems as symbols or images specific to each engineering discipline, with links shown as lines between these symbols, and ports sometimes shown as smaller symbols and images on component symbols. Properties are typically shown in dialogs or other interfaces accessible from component and port symbols, rather than directly on diagrams. Some simulation tools enable modelers to specify system structures in text files, and those with graphical interfaces will generate these text files automatically. Example file formats from some simulation tools are shown in Section 7.

3.2. System Behavior in Simulation Modeling

Behavior in simulation refers to changes in component and system property values over time. Behavior of simulation models is derived from behavior of their components and links between ports on components. Earlier reviews of simulation platforms compared the concepts they use for modeling behavior, as described in this section [Matei and Bock, 2012a] [Matei and Bock, 2012b]. Section 3.2.1 covers how simulation models set limits on changes in property values. Section 3.2.2 covers the behavior of links between ports, while Section 3.2.3 covers component behavior. Section 3.2.4 describes the combination of these into behavior of the overall simulation model.

3.2.1. Property Values

System behavior involves components that process and exchange physical substances and numeric information (signals), but simulating this behavior only predicts variations of component and system property values over time (see Section 3.1 about properties). This is because property values are the only things that change during simulation (system structure cannot change).

Properties in simulation models can place two kinds of restrictions on how their values change [Matei and Bock, 2012a] [Matei and Bock, 2012b]. The first of these is whether their values change at all during simulation:

- *Constant* properties have values that do not change during each "run" of a simulator, but might change between simulation runs, such as the capacity of a pump.
- *Variable* properties have values that might change during a single simulation run, such as the rate at which a fluid is coming out of a pump at a particular time.

Terms for the concepts above differ between simulation platforms, but the meaning is the same. Simulation platforms sometimes refer to constants as parameters.[4] The terms above are part of an abstraction over various simulation platforms that enables comparison to SysML in Section 4, and development of a simulator-independent extension of SysML in Section 7.

The second restriction is on variables, which can change in two ways during simulation:

- *Continuous* variables have values that are close to their values at nearby times in the past and future.[5]
- *Discrete* variables have values that are the same as their values at nearby times in either the past or the future, or both.[6]

Informally defined, continuous variables vary smoothly over time, including the possibility of remaining constant, while discrete variables are always constant for a period of time, then change instantaneously to a possibly very different value for another period of time. Variables being continuous and discrete do not imply any restriction on the range of their values, only the way in which those values change over time. For example, the top line in Figure 2 shows a variable changing continuously, while the rest show discrete variables. Discrete variables can be further restricted to change values only at regular intervals (*change cycle*). The second line in Figure 2 has a change cycle lasting as long as the first line segment on the left. The rest of the segments on that line are either of the same length or multiples of the first segment's length (discrete variables do not need to change at every interval). The third line is the same, except the change cycle is smaller, lasting as long as the second line segment on the left.

Terms for the concepts above differ between simulation platforms, but the meaning is the same. Simulation platforms sometimes refer to change cycles as sample times, as in sensors that output measures of continuously varying physical properties at regular intervals. The terms above are part of an abstraction over various simulation platforms that enables comparison to SysML in Section 4, and development of a simulator-independent extension of SysML in Section 7.

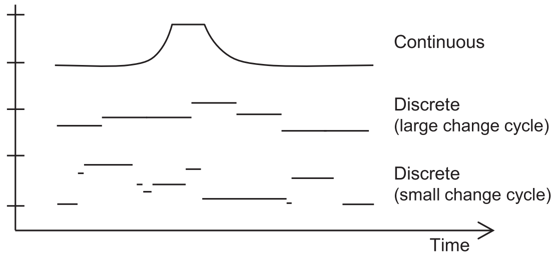


Figure 2. Continuous and discrete variables.

3.2.2. Component Interaction

Simulation models distinguish component interactions based whether the things being exchanged are physical or informational [Matei and Bock, 2012b]:

- *Physical interaction*: Components exchange various kinds of physical substances in terms of their conserved characteristics, such as electric charge, momentum, and entropy.
- *Signal flow*: Components exchange numeric information.

These differ in that physical substances are conserved, while information is not, and movement of physical substances affects the mover (bidirectional flow), while sending of information does not affect the sender (unidirectional flow). The latter difference is reflected in whether components specify inputs and outputs, see Signal Flow subsection.

Terms for the concepts above differ between simulation platforms, but the meaning is the same. Simulation platforms sometimes refer to signal flow modeling as causal and physical interaction modeling as acausal, even though the systems being modeled never have an effect without an earlier cause (a fundamental law of physics). The terms above are part of an abstraction over various simulation platforms that enables comparison to SysML in Section 4, and development of a simulator-independent extension of SysML in Section 7.

Physical interaction and signal flow are covered in the Physical Interaction and Signal Flow subsections, respectively, with equations for both given in the Physical Interaction subsection.

Physical Interaction. Physical interactions between system components are bidirectional, because physical actions of components on others might cause reactions back onto the acting components. For example, components attempting to output water to other components might be forced to accept water if the other components put the water under greater pressure. Physical modeling does not specify inputs and outputs for components, since any output can also be an input, due to reactions back into the output component (compare to sending information between components, see the Signal Flow subsection).

Simulation treats movement of physical substances as uninterrupted movement of their conserved characteristics, such as electric charge, momentum, and entropy, rather than individual movement of the substances themselves, such as electrons, steel, and steam. This makes rates of flow independent of the time interval over which flow rates are calculated, which could cause flow rates to vary discontinuously between objects. For simplicity, physical substances in the rest of this article refer to their conserved physical characteristics, rather than the substances themselves.

Simulation platforms generate and solve the same differential equations for flow of conserved physical substances by treating them all as carrying energy [Paynter, 1960] [Cellier, Elmqvist, and Otter, 1999]. The mathematics of energy exchange is the same regardless of the kind of substances carrying it, and the equations can be solved by the same algorithms. Rate of energy exchange (*power*) is equal to the product of the following two properties of flowing substances:[7]

- *Flow rate*: The amount of substance per time moving in or out ports, such as current (electric charge per time), force and torque (linear and angular momentum per time), and entropy flow rate (entropy per time).
- *Potential to flow*: An impetus for substances to move in or out of ports, such as voltage for electric charge, linear and angular velocity for momentum, and temperature for entropy.

The potential for substances to flow can only be realized (as nonzero flow rates) when:

- potentials in the system differ, either between ports of a component, or between a port and an internal component (potentials cannot differ across links, see below).
- components allow substances to flow, see resistance to flow in Section 3.2.3.

For example, fluid can only flow through a pipe (treated as a component) when pressures on the ends of a pipe (treated as ports) are different, and this difference in potential becomes actual flow depending on how much the pipe resists it.

Terms for flow rate and potential differ between physical disciplines, as illustrated in Table I, but the concepts are the same from the point of view of simulation [Raven, 1995] [Cellier et al., 1999].[8] Sometimes potential to flow is referred to as effort, even though forces and torques are not potentials. The terms above are part of an abstraction that enables comparison to SysML in Section 4, and development of a simulator-independent extension of SysML presented in Section 7.

Links between system components in simulation models pass along properties of physical flows above in a very restricted way, because links do not correspond to physical things, they just represent mathematical equations between variables in separate components. Links do not create, destroy, transform, store, resist, or take up time when physical substances "flow" across them (compare to components in Section 3.2.3). This is reflected in restrictions on flow rates and potentials across links, with flow rates on link ends adding to zero, and potentials on link ends always being equal, as illustrated in Figure 3 (compare potentials to information that can be sent to multiple destinations without being divided up across them, see the Signal Flow subsection). The bidirectional arrows represent physical substances going into and out of ports on components. A single flow on the left is split into two on the right. Flow rates are taken as positive in one direction and negative in the other, causing flow rates on the ports to sum up to zero, as indicated by the first equation at the

Table I. Commonality Across Physical Domains (Adapted from [Raven, 1995])

Domain	What is flowing	Flow rate	Potential to flow
Electrical	Charge	Current	Voltage
Mechanical, translational	Linear momentum	Force	Linear velocity
Mechanical, angular	Angular momentum	Torque	Angular velocity
Hydraulic	Volume	Volume flow rate	Pressure
Thermal	Entropy	Entropy flow rate	Temperature

bottom of the figure. This reflects conservation of substances flowing between components, which requires changes in flow rates at each port to balance changes in flow rates at the other ports. Potentials are the same on all ports, as indicated by the second equation. This reflects lack of resistance between components, which requires changes in potential at each port to be matched by the same changes in the other ports.

Simulation models distinguish variable properties according to the two properties of flowing substances above:

- *Conserved* variables represent flow rates.
- *Nonconserved* variables represent potentials to flow.

Terms for the concepts above differ between simulation platforms, but the meaning is the same. Simulation platforms sometimes refer to conserved variables as flow, balancing, or through variables, and nonconserved variables as across variables. The terms above are part of an abstraction over various simulation platforms that enable comparison to SysML in Section 4, and development of a simulator-independent extension of SysML in Section 7.

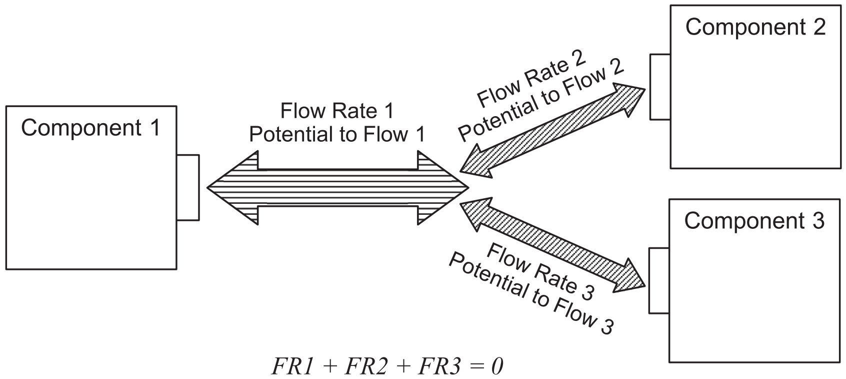
Signal Flow. Signal flow between system components is unidirectional, because sending (numeric) information only affects the receiver, with no reaction back to the sender as in physical interactions. This is typical in information movement and manipulation, as needed in control engineering and signal processing. Signal flow modeling specifies inputs and outputs for components, with signals moving from outputs of one component to inputs of others or back to its own inputs (compare to physical interaction between components, see the Physical Interaction subsection).

Simulation models distinguish ports by the direction of flows they support:

- *Input* ports accept only incoming flows.
- *Output* ports provide only outgoing flows.
- *Bidirectional* ports accept incoming flows and provide outgoing flows (for physical interactions, see the Physical Interaction subsection).

Signals behave as if they were potentials to flow (nonconserved variables), because they move along links to multiple components without change, like information sent to multiple destinations, as illustrated by Figure 3 in the Physical Interaction subsection.[9] In this sense, signal flow is a special case of physical interaction that is concerned only with potentials. Signal flow ultimately occurs through the medium of physical substances (electric charge, momentum, fluid, entropy, and so on), but these substances can have any flow rate, including zero, and still carry signals, allowing flow rate to be ignored in signal flow simulation. For example, sound passes along pipes full of water even when no water moves through the pipes, and voltages pass along wires between operational amplifiers with almost no current.[10] Modeling signals over physical media is useful for modern systems, which usually include physical processes monitored and controlled by software [Dorf and Bishop, 2016].

The terms above for input and output ports are the same across simulation platforms, though both are sometimes informally called causal ports (see Section 3.2.2 about this terminology). Simulation platforms usually indicate bidirectional ports by the absence of indication that they are input or output, though they are sometimes informally called acausal



$$FR1 + FR2 + FR3 = 0$$

$$PF1 = PF2 = PF3$$

Figure 3. Flow rate and potential to flow between components.

ports. The terms above are part of an abstraction over various simulation platforms that enables comparison to SysML in Section 4, and development of a simulator-independent extension of SysML in Section 7.

Links that have an input port on one end must have an output port on the other, and vice versa, which also means links that have a bidirectional port on one end must have a bidirectional port on the other. However, the same component can have all three kinds of port in systems that combine signal flow and physical interaction, as in electronic control of physical devices. Another restriction is multiple output ports cannot link to the same input port in signal flow. This is because signals have one numeric value each, and merging multiple outputs into the same input would require different numeric output values to be combined into one numeric input.[11] Physical systems that implement signal flow do not behave this way (see the Physical Interaction subsection) and simulators do not support it.

3.2.3. Component Behavior

Simulation models specify behavior with equations containing variables named according to properties in the models. Equations in a component or system can only refer to properties of itself, its ports, and properties of its subcomponents and their ports (to any depth). Equations in a component cannot refer to properties of the components containing it, or to external components linked to its ports. This encourages equations to be defined without dependence on how components are reused in other components or systems.

Equations in components specify what happens to physical substances and signals available at their ports (compare to links in the Physical Interaction subsection of 3.2.2, which have only predefined equations). Component equations can specify how flow rate, potential, and signal values are related across ports and with component properties, according to the intended processes, such as:

- Passing physical substances and signals between ports. Equations can relate the values of flow properties of physical substances at one port to those of the same substances at another port. For example, the volume flow rate and pressure of a fluid at one end (port) of a pipe component can be related to those properties at the other end. Flow rates have opposite signs at each end and pressure is lower at the end to which fluid is moving, due to resistance in the pipe.[12] Equations can relate signal values coming into one port to those going out of another. For example, an amplifier accepts a varying numeric signal and emits a signal that varies the same way, but more widely.[13]
- Transforming physical substances and signals from port to port. Equations can relate the values of flow properties of physical substances at one port to those of different substances at another port. For example, the torque and angular velocity of a gear can be related to force and linear velocity of a rack, transforming angular to linear momentum. A component for this transformation has ports for connecting to a gear and rack, and equations specifying that the power of the momentums (products of their flow rates and potentials, see the Physical Interaction subsection of 3.2.2) will be equal for ideal gears. Equations can relate numeric signal values coming into one port to Boolean values going out of another. For example, a signal processor can accept a varying numeric signal and emit true or false values depending on whether the input is above some threshold.
- Creating and destroying physical substances. Equations for transformations as in the previous bullet involve creation and destruction of physical substances, such as creating linear momentum. This can involve destruction of physical substances also, such as differences in angular and linear momentum due to gear friction. Signal transformation could be considered creation and destruction in the sense of information lost by filtering or added by sensing.
- Storing physical substances. Equations can specify that physical substances flowing into a component are kept there until a later time. For example, the flow rate of fluid at a valve on a tank can be mathematically integrated to calculate the total amount of fluid in the tank. Signals can be integrated, though this is not the same as storing signals, because some information about past variations is lost.

Equations refer to properties of components that are derived from port properties or track internal states. The properties can be constants and variables, but not flow rates or potentials, at least not in the same sense as ports. For example, when a component handles physical substances, it is useful for the component to have variables for

- Potential difference between ports, because it is potential differences that cause substances to flow (see the Physical Interaction subsection of 3.2.2). This variable differs from potentials at ports, which are only for single points in the system, and cannot cause flows by themselves.
- Rate at which substances flow through the component. This variable differs from flow rates at ports, because it reflects whether substances are created, destroyed, transformed, stored, or just pass through the component (compare to flows across links, which must conserve physical substances, see Figure 3 in the Physical Interaction subsection of 3.2.2). It will have the same value as the flow rate of one of the ports only when the component passes the substance between ports without changing anything in between, except possibly decreased potential due to resistance. Otherwise it will reflect differences in flow rates between ports.

Component variables also track internals states, for example, how much of a physical substance is stored at a particular time (and similarly for signal integrators), or the temperature of a component. Constants are needed for properties that do not vary during simulation, such as resistance or transformation ratios between substances.

3.2.4. System Behavior

Simulation tools translate their graphical models into equations, then use numerical analysis techniques to calculate the values of variables over time. Variable properties are translated into mathematical variables in equations, omitting the relationship of variables to components or other simulation model elements, or to physical substances (see the introduction to Section 3). Numerical analysis techniques are only concerned with mathematical relationships between numeric values, and do not directly address engineering and even physical aspects of the systems being modeled. The equations do not depend on the engineering disciplines involved, though simpler equations can be used when only signals are exchanged between components. The equations do not depend on the symbols or images used in graphical interfaces.

Equations generated from simulation models include those specified by the modeler for components, as well as others generated automatically from links between components, see Sections 3.2.3 and the Physical Interaction subsection of 3.2.2, respectively. Simulation tools generate two kinds of equations from links between components:

- *Ordinary differential/difference equations* (ODEs) have a single derivative or difference by itself on one side, for a function of one variable with respect to time.
- *Differential/difference algebraic equations* (DAEs) can have multiple derivatives or differences on both sides, for functions of one variable with respect to time, and include unary operators on derivatives or differences, such as exponentiation.

ODEs are generated for links between output and input ports, while DAEs are generated for links between bidirectional ports (signal flows and physical interactions, respectively, see the Signal Flow and Physical Interaction subsections of 3.2.2).

4. SysML COMPARED TO SIMULATION MODELS

This section compares SysML to simulation modeling to find concepts in SysML that are equivalent or close to those in simulation. The comparison used to guide development of a simulator-independent extension of SysML in Section 5. A brief outline of SysML is given in this section, with more detail in an extended example in Section 6, but familiarity with SysML is assumed [Object Management Group 2017a] [Object Management Group 2015a] [Friedenthal, Moore, and Steiner, 2014] [Holt and Perry, 2013].

Table II. Simulation and SysML Terms

Simulation Concepts		SysML
Models		Blocks with internal block diagram, treated separately from other blocks they might be components of
Components	Atomic	Blocks without internal block diagram
	Subsystems	Blocks with internal block diagram
Links		Connectors
Ports		Ports with flow properties
Equations		Constraint blocks

SysML includes diagrams for structure and behavior, as well as diagrams for requirements and parametric relationships, which can be considered to be structure or behavior depending on their content. Package diagrams are used to organize system models, and are considered a kind of structure diagram that applies to models rather than systems. Figure 4 summarizes the SysML diagram taxonomy (arrows point to broader categories). Bold outlines indicating diagrams that include extensions to UML.

The most basic structural elements of SysML center around *blocks*, which represent kinds of systems, subsystems, components, and parts (systems and their elements). Blocks appear primarily in these kinds of diagrams:

- *Block definition* diagrams show kinds of systems and their elements, as well as their relationships, including classification (taxonomies) and containment, and their quantitative properties.
- *Internal block* diagrams describe the interconnection of components and parts used within a single system or component.
- *Parametric* diagrams are a kind of internal block diagram that specify or apply equations involving properties of subsystems, components, and parts.

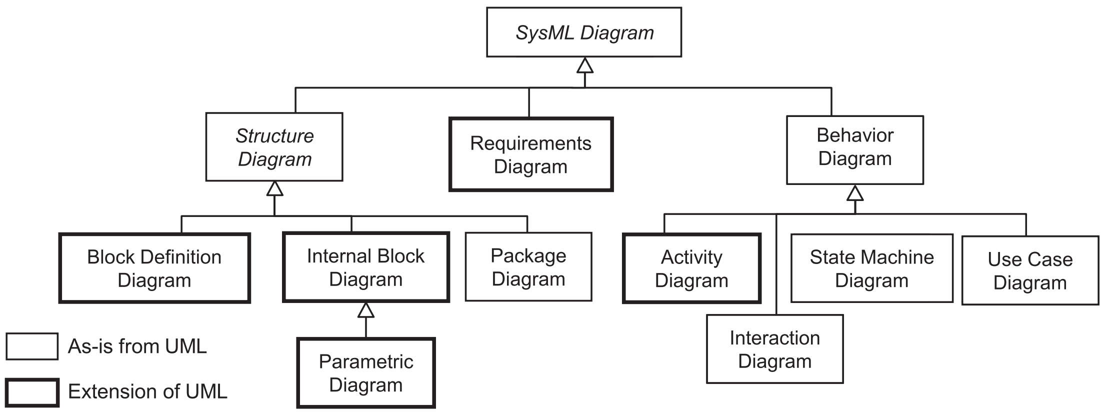


Figure 4. SysML diagrams.

System behavior in SysML is specified in these kind of diagrams:

- *Use case* diagrams are high-level descriptions of systems interacting with the environment in which they are used.
- *Interaction* diagrams specify flow of items between systems and their environments, and between subsystems, components, and parts.
- *State machine* diagrams specify how systems or their elements react to external stimuli.
- *Activity* diagrams specify actions taken by systems or their elements, sequences of those actions, and flow of items (physical substances or information) between them.

Among SysML structural diagrams, internal block diagrams naturally correspond to simulation models, because both

- describe systems and their elements (blocks in SysML), as well as links and directional and bidirectional flows between them (connectors and ports with flow properties in SysML, ports with variables in simulation).
- can use equalities to define behavior of components (constraint blocks in SysML, equations in simulation).
- support reuse of the same element by multiple diagrams, with reused elements defined once elsewhere (in block definition diagrams and simulation libraries).

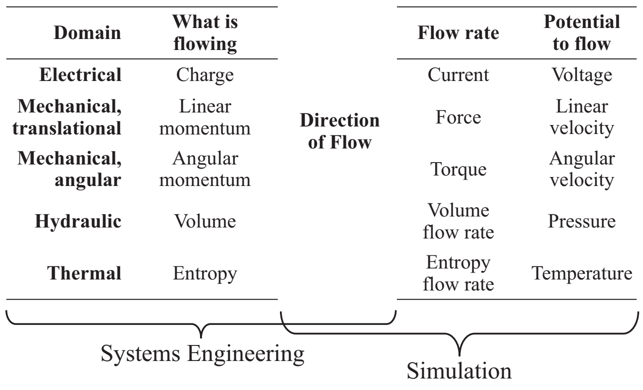
Table II gives correspondences between the abstract structural simulation concepts in Section 3.1 and SysML terms.

Parametric diagrams are internal structure diagrams useful in simulation modeling when applying reusable equations (constraint blocks) to components. Parametric diagrams describe equalities (binding connectors) between properties of systems, subsystems, components, parts, and reusable constraint blocks expressing equations. These are useful for properties of the same system or element, but when used for equations between components, such as flow rate conservation in Figure 3 in the Physical Interaction subsection of 3.2.2, the constraints would appear for every interaction between components, cluttering diagrams with equations that can be assumed to apply even when they are not shown, as in typical simulation models.

SysML structural diagrams and simulation models have a number of similarities and differences:

1. SysML value properties do not indicate whether their values are constant or variable,[14] continuous or discrete, and conserved or nonconserved (in the simulation sense of these terms, see the Physical Interaction subsection of 3.2.2).
2. SysML flow properties specify direction of flow, as simulation ports do, but flow properties specify the kinds of things that flow, while simulation ports do not. Flow properties do not specify flow rates and potentials as simulation ports do. This is illustrated in Figure 5.
3. SysML connectors provide equality semantics for links between nonconserved variables (bindings), but not conservation semantics for links between conserved variables.

The second comparison above regards flow modeling, as illustrated in Figure 5. The figure breaks up Table I in



Domain	What is flowing		Flow rate	Potential to flow
Electrical	Charge	**Direction of Flow**	Current	Voltage
Mechanical, translational	Linear momentum		Force	Linear velocity
Mechanical, angular	Angular momentum		Torque	Angular velocity
Hydraulic	Volume		Volume flow rate	Pressure
Thermal	Entropy		Entropy flow rate	Temperature

Systems Engineering
Simulation

Figure 5. Overlapping concerns about flows in systems engineering and simulation.

the Physical Interaction subsection of 3.2.2 to show which properties of physical flow are of concern to system modeling and simulation separately, and adds flow direction as an element of concern to both. System engineering identifies the direction flow to indicate whether physical or informational disciplines will be involved (see Section 3.2.2), and identifies the kind of physical items flowing between components (charge, momentum, fluid, and so on) to indicate the physical engineering disciplines needed (electric, mechanical, fluid, and so on). Simulators require the direction of flow to predict the evolution of flow rates and potentials over time, but are not concerned with the kind of things flowing (see the Physical Interaction subsection of 3.2.2).

Among SysML behavior diagrams, all but activity diagrams are unsuitable for physical interaction and signal flow simulation modeling:

- Use cases diagrams only describe hierarchies of interactions, with no specification of the flows between system components.
- Interactions diagrams cannot specify continuous flows, because each flow is a separate element in the model, shown graphically as a line between interacting elements.
- State machine diagrams do not explicitly specify flows between subsystems, components, and parts, because they are designed to specify sending and receiving for each subsystem, component, or part separately, rather than the links between them.

In terms of execution, activity diagrams are very similar to signal flow simulation models, but less so to physical interaction simulation. Activities and signal flow simulation both express flows between outputs and inputs of elements that specify detailed behavior (actions in activities, components in simulation models). Activity diagrams support continuous and discrete flows, as simulation models do, because there is no limit to how frequently actions can exchange items or how rapidly they execute [Bock, 2006]. Activity diagrams support bidirectional flows at their boundaries, but not when reused in other activities, making physical interaction modeling cumbersome.

Regarding terminology, activity diagrams are concerned with actions, while typical engineering applications are concerned with systems, components, and parts (structure vs. behavior). Though this does not affect execution, it is important in choosing the portions of SysML that are most likely to be adopted by engineers. Some signal flow applications are purely behavioral, as in signal processing, but even in these areas, engineers often imagine physical components performing the processing.

Despite the semantic (execution) similarity of activity diagrams and signal-flow simulation models, the extension described in Section 5 is based on structural diagrams in SysML, because they are more easily adapted to physical interaction simulation, and engineers are more likely to see them as fitting their needs, due to terminology and typical application of the diagrams. Unification is possible between structural and behavior diagrams in SysML [Bock and Odell, 2011], but this is not standardized yet.

The comparison of SysML and simulation models in this section suggests that SysML internal block diagrams are the closest to simulation models, but differ in their use of properties. Internal block diagrams and simulation models both show interconnected system components between which flows occur, with system components defined separately for reuse in multiple diagrams and models. SysML properties support flow modeling. These overlap simulation models by indicating direction of flow, but differ by giving the kind of thing flowing and omitting flow rate and potential variables needed in simulation. Section 5 fills the gaps between SysML internal block diagrams and simulation modeling to support integration.

5. SysML EXTENSION FOR SIMULATION

This section uses the comparison of SysML and simulation modeling in Section 4 to guide development of SysML extensions. The extensions reuse SysML concepts where they are equivalent to those in the simulation abstraction developed in Section 3, and extends them to match the abstraction where they are close. The extensions are modeled using UML's *profiling* facility, the same one SysML uses to extend UML. The constructs in profiles used in the simulation extension are:

- Stereotypes, to add modeling capabilities and constrain usage.
- Model libraries, to provide reusable model elements.
- Textual languages, for defining expressions and constraints.

The extension for simulation defines stereotypes based on SysML blocks and properties (appearing in SysML block definition and internal block diagrams, see Section 4), reusable applications of these stereotypes in model libraries, and an equation language for constraints, shown in Sections 5.1, 5.2, and 5.3, respectively.

5.1. Stereotypes

The first two stereotypes distinguish constants and variables, as shown on the left in Figure 6, which can be used to describe flows between components or behavior within components (see Sections 3.2.1 through 3.2.3). The SimConstant stereotype is applied to SysML properties with values that remain constant during each simulation execution, while SimVariable is applied to properties with values that might change during a simulation. Simulation variables characterize how their values change:

- isContinuous: If true, the variable's value can only change continuously, otherwise the value can only change discretely, as defined in Section 3.2.3. The default is continuous.
- changeCycle: Specifies the time cycle at which a discrete variable (isContinuous=false) might change values. During each cycle, the value must be constant. The value might change at the end of each cycle, but not necessarily. In the case of continuous variables, changeCycle is zero, which means the variable value might change at any time. The default is zero.
- isConserved: For specifying flows between components only, see below.

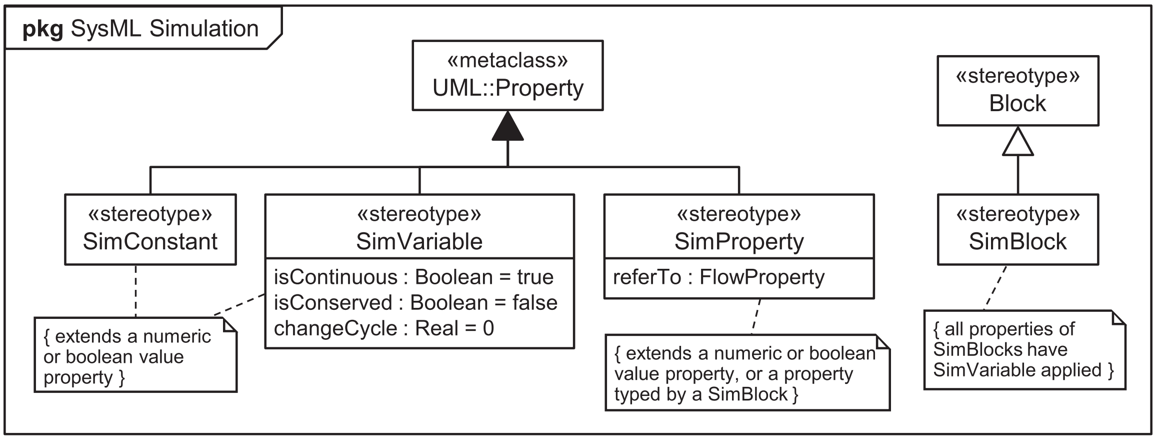

pkg SysML Simulation
«metaclass» UML::Property
«stereotype» Block
«stereotype» SimConstant
«stereotype» SimVariable
isContinuous : Boolean = true
isConserved : Boolean = false
changeCycle : Real = 0
«stereotype» SimProperty
referTo : FlowProperty
«stereotype» SimBlock
{ extends a numeric or boolean value property }
{ extends a numeric or boolean value property, or a property typed by a SimBlock }
{ all properties of SimBlocks have SimVariable applied }

Figure 6. SysML stereotypes for physical interaction and signal flow simulation.

The characteristics above are partly dependent on each other and on the property to which the stereotype is applied. Values of simulation variables must be numbers or Booleans (Real, Integer, or Boolean in SysML, or specializations of them). Continuous variables must have real number values.

The rest of the stereotypes are only for specifying flows between components. These have two kinds of characteristics:

1. Time-independent: specified by SysML flow properties, in particular, the kinds of things flowing and their possible directions. These aspects of flows do not change during simulation, as those in the next item do.
2. Time-dependent: specified by SysML properties that have the SimProperty stereotype applied. Simulation properties can be typed in two ways, depending on whether the flow is physical or signal:
   a. For physical interaction, simulation properties are typed by blocks with the SimBlock stereotype applied, to specify flow rates and potentials (see the Physical Interaction subsection of 3.2.2). All properties of simulation blocks have SimVariable applied, and must be real-valued and continuous. The variables must be defined in pairs of conserved and nonconserved. When isConserved is true, values are conserved across flows between components (flow rate), otherwise the values are the same (potential to flow), as defined in the Physical Interaction subsection of 3.2.2. The default is nonconserved.
   b. For signal flow, simulation properties have numeric or Boolean values (see the Signal Flow subsection of 3.2.2).[15] To specify that a signal value changes discretely, including cyclically, the SimVariable stereotype is applied to the simulation property (SysML model elements can have more than one stereotype applied at the same time). Otherwise, the signal is assumed to be continuous or discrete based on the simulation property type (real signals being continuous, integral and Boolean signals being discrete).

Simulation properties bring the two kinds of information above together by referring to flow properties (via referTo in Fig. 6). The flow property's type specifies the kind of things flowing, which are not Booleans or numbers. The simulation property referring to it gives (a) real numbers characterizing the flow of those physical things via simulation variables on simulation blocks (for physical interaction) or (b) a number or Boolean that is flowing as the value of the simulation property (for signal flow). The flow property must be defined on the same block (port type) as the simulation property, or a more general block.

Some shorthand phrasing is used in the rest of this article for brevity:

- Simulation variables on a simulation block typing a simulation property (or the simulation property itself for signal flow) are called *flow variables* of the flow property referred to by the simulation property.
- Compatible flow properties on the types of connected ports are said to be connected.[16]

Additional modeling constraints apply when simulation properties are on blocks used and connected in internal block diagrams:

1. Flow variables of connected flow properties must have the same names (unless they are simulation properties) and the same types (*matching* flow variables).
2. Connected flow properties must have opposite direction or both must be bidirectional.[17]
3. Flow properties with in direction can be connected to no more than one other flow property.[18]

Connectors between flow properties are given the semantics of simulation links (see Fig. 3 in the Physical Interaction subsection of 3.2.2), restricting values of flow variables. In the context of each instance of a block containing a connector [Bock, 2004], and for each collection of connected flow properties, the values of matching:

1. conserved flow variables sum to zero.
2. nonconserved (including signal) flow variables are equal.

In the first semantic rule above, all connectors to the same flow property are viewed as a single multiway connection, and all the matching conserved flow variables together obey the rule.

Simulation variables on the same simulation block are typically typed in nonoverlapping ways to avoid redundancy. For example, if one simulation variable is typed by voltage, there will not be another simulation variable on the same simulation block typed by voltage, because they would have the same value for the flows they describe. Ports in simulation platforms sometimes have multiple variables with the same type, where each variable is interpreted as a different flow, and typically obeying different equations. This would be modeled with multiple simulation blocks, one for each variable of the same type, and each simulation block would describe a separate flow by referring to separate flow properties. The flow properties could be on the same block used to type ports, enabling ports in the extended SysML model to be in one-to-one correspondence with ports in simulation platforms.

Correspondences and differences between the SysML extension stereotypes above and simulation platforms are discussed in Section 7.3.

5.2. Model Libraries

Constructing SysML models for simulation is easier when commonly needed simulation elements are predefined in *model libraries*. Model libraries are included in the SysML extension for component interaction and component behavior (see Sections 3.2.2 and 3.2.3) in common between the most widely used platform libraries (Simulink/Simscape and Modelica). The two parts of the extension library (interaction and behavior) each divide into elements for physical and signal modeling.

Model libraries for component interaction are shown in Figures 7 and 8. Features of the blocks are segmented into compartments labeled according to the kind of feature in them, such as flow properties or simulation variables. Val-

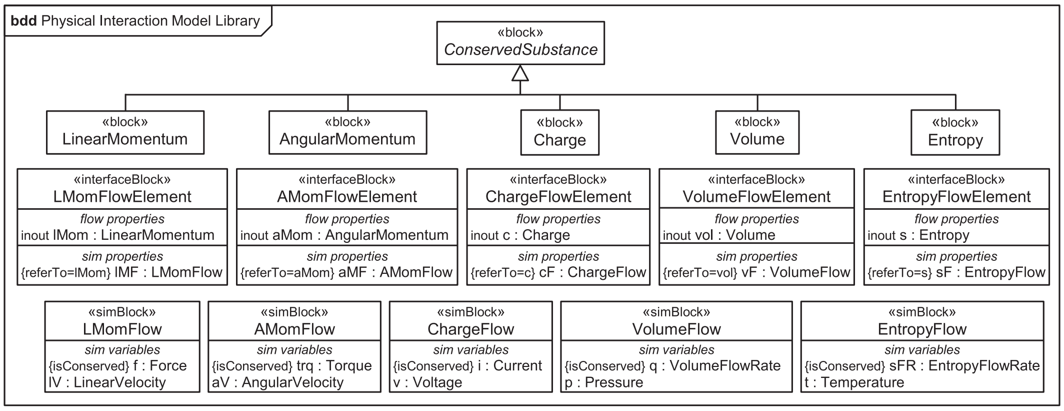

bdd Physical Interaction Model Library
«block» ConservedSubstance
«block» LinearMomentum
«block» AngularMomentum
«block» Charge
«block» Volume
«block» Entropy
«interfaceBlock» LMomFlowElement; flow properties: inout lMom : LinearMomentum; sim properties: {referTo=lMom} lMF : LMomFlow
«interfaceBlock» AMomFlowElement; flow properties: inout aMom : AngularMomentum; sim properties: {referTo=aMom} aMF : AMomFlow
«interfaceBlock» ChargeFlowElement; flow properties: inout c : Charge; sim properties: {referTo=c} cF : ChargeFlow
«interfaceBlock» VolumeFlowElement; flow properties: inout vol : Volume; sim properties: {referTo=vol} vF : VolumeFlow
«interfaceBlock» EntropyFlowElement; flow properties: inout s : Entropy; sim properties: {referTo=s} sF : EntropyFlow
«simBlock» LMomFlow; sim variables: {isConserved} f : Force; lV : LinearVelocity
«simBlock» AMomFlow; sim variables: {isConserved} trq : Torque; aV : AngularVelocity
«simBlock» ChargeFlow; sim variables: {isConserved} i : Current; v : Voltage
«simBlock» VolumeFlow; sim variables: {isConserved} q : VolumeFlowRate; p : Pressure
«simBlock» EntropyFlow; sim variables: {isConserved} sFR : EntropyFlowRate; t : Temperature

Figure 7. Physical interaction library.

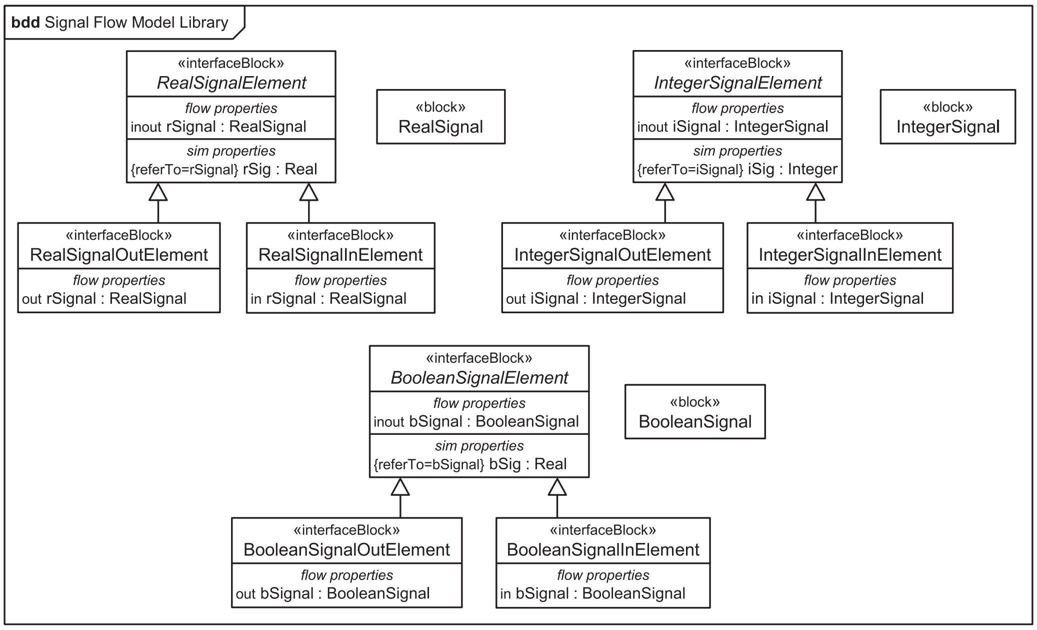

bdd Signal Flow Model Library
«interfaceBlock» RealSignalElement; flow properties: inout rSignal : RealSignal; sim properties: {referTo=rSignal} rSig : Real
«block» RealSignal
«interfaceBlock» IntegerSignalElement; flow properties: inout iSignal : IntegerSignal; sim properties: {referTo=iSignal} iSig : Integer
«block» IntegerSignal
«interfaceBlock» RealSignalOutElement; flow properties: out rSignal : RealSignal
«interfaceBlock» RealSignalInElement; flow properties: in rSignal : RealSignal
«interfaceBlock» IntegerSignalOutElement; flow properties: out iSignal : IntegerSignal
«interfaceBlock» IntegerSignalInElement; flow properties: in iSignal : IntegerSignal
«interfaceBlock» BooleanSignalElement; flow properties: inout bSignal : BooleanSignal; sim properties: {referTo=bSignal} bSig : Real
«block» BooleanSignal
«interfaceBlock» BooleanSignalOutElement; flow properties: out bSignal : BooleanSignal
«interfaceBlock» BooleanSignalInElement; flow properties: in bSignal : BooleanSignal

Figure 8. Signal flow library.

ues of stereotype properties such as isConserved and referTo appear in curly braces before the names of the properties they apply to (Boolean stereotype properties are only shown when their value differs from the default). These libraries contain SysML interface blocks, which are blocks that do not define behavior. They can be used as components or ports in models that do not specify behavior or can be specialized by regular blocks to add behavior in components or ports. The two libraries are for:

- Physical interaction (Fig. 7): Each kind of conserved substance has two related elements, one providing flow and simulation properties for that substance (in middle of the diagram vertically), and the other providing flow rate and potential properties on simulation blocks typing the simulation properties (at the bottom of the diagram). Similar elements can be defined for other conserved substances and reused as necessary, see Section 8. Units are needed in this model, but are beyond the scope of this article. To validate the SysML extension, Figure 7 uses specialized real number value types named according to the kind of physical quantities being specified, such as force or angular velocity (see Section 7.3).
- Signal flow (Fig. 8): These are numeric or Boolean, and flow in or out of components or ports. Signal flow properties rSignal, iSignal, and bSignal are introduced in the general blocks RealSignalElement, IntegerSignalElement, and BooleanSignalElement, respectively, but their direction is disambiguated in specialized blocks using property redefinition (a way of restricting properties inherited from more general blocks). These library elements are useful when signal values have no application-specific meaning, enabling the names (RealSignal, iSig, and so on) to be reused (compare to the example in Section 6.3).[19]

A small portion of the model libraries for component behavior are shown in Figure 9. The ones along the top are for

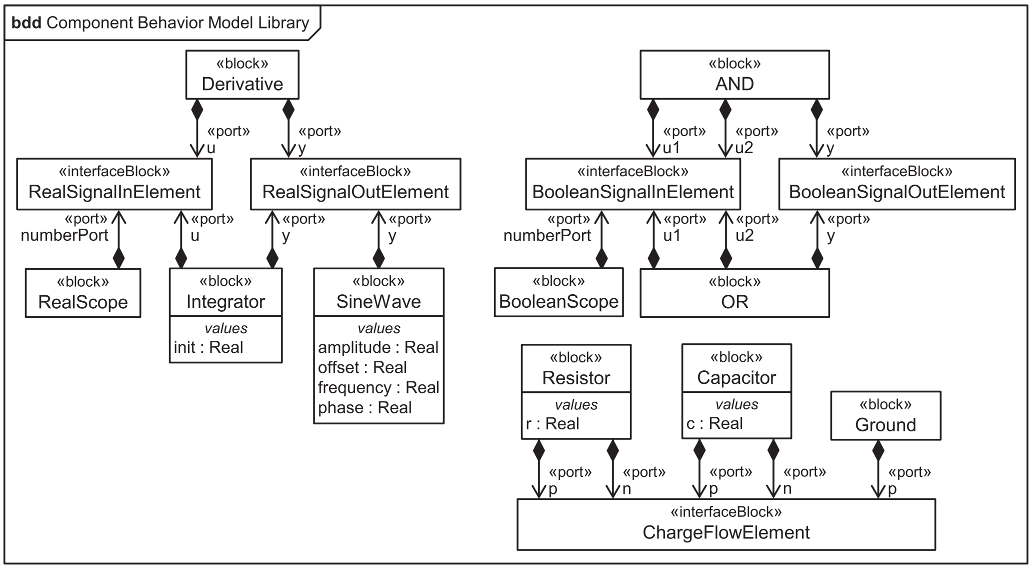


Figure 9. Component behavior library (signal and physical).

signals, while the ones on the lower right are for physical behavior. On the upper left are real number signal components, to calculate derivatives and integrals, generate sine waves, and display results. On the upper right are Boolean signal components, for logical operations and displaying results. The lower right shows three electrical components. Their behavior is defined in the corresponding simulation platform libraries, see Section 7.3. Model libraries in the SysML extension provide many other components for common mathematical operations, discrete and nonlinear behavior, sources and sinks, routing, and electrical behavior, which are publicly available [Barbau and Bock, 2017].

Correspondences and differences between the SysML extension libraries above and simulation platform libraries are discussed in Section 7.3.

5.3. Equation Language

SysML constraints can be written in any textual language, but SysML does not define one.[20] Fortunately, the equation languages for most widely used simulation platforms (Simulink, Simscape, and Modelica) have many constructs in common and simple correspondences for others. This yields a useful equation language for SysML models that can be translated to simulation tools typically used by engineers. The language includes:

- A subset of Modelica's equation grammar that includes only these nonterminals and the terminals they depend on: `simple_expression, expression, if_equation, name, function_call_args, logical_expression, logical_term, logical_factor, relation, arithmetic_expression, rel_op, add_op, term, mul_op, factor, primary, component_reference, output_expression_list, expression_list, function_arguments, array_subscripts, function_argument, for_indices, named_arguments, named_argument, subscript`.
- Predefined mathematical functions: `abs, sign, sqrt, div, mod, rem, ceil, floor, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, log, log10, exp, der`.

Differences between the Modelica subset above and the equation language for Simulink and Simscape (MATLAB) are discussed in Section 7.4.

6. EXAMPLE USING THE SIMULATION EXTENSION

This section applies the SysML extension in Section 5 to an example combining physical interaction and signal flow. Section 6.1 describes the problem and models its system structure and behavioral relationships. Section 6.2 models some of the physical interactions used in Section 6.1, and Section 6.3 models some of the signals flows.

6.1. System Structure and Behavior

The example is illustrated in Figure 10, an automobile cruise control system, including its vehicle, the operating environment of the vehicle, and the physical and informational processes involved (*total system*). Cruise controllers attempt to keep the speed of their vehicles constant despite environmental disturbances that might affect it, such as changes in the slope of the road, rolling resistance of the tires, and wind effects. Cruise controllers respond to these environmental disturbances by adjusting the power delivered by the vehicle's engine, to maintain the desired speed initially set by the driver.

Figure 10 shows physical interactions with solid, bidirectional arrows between system components, and signal flows with dashed, unidirectional arrows. The driver gives the desired speed to the cruise controller, modeled as a signal flow,

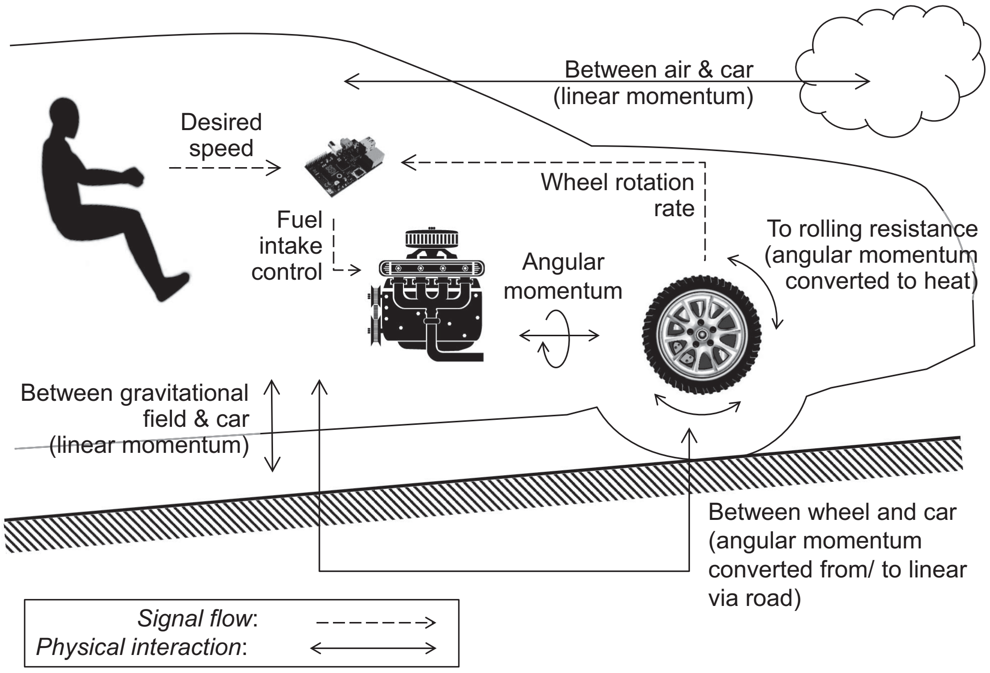


Figure 10. Total system for a cruise controller.

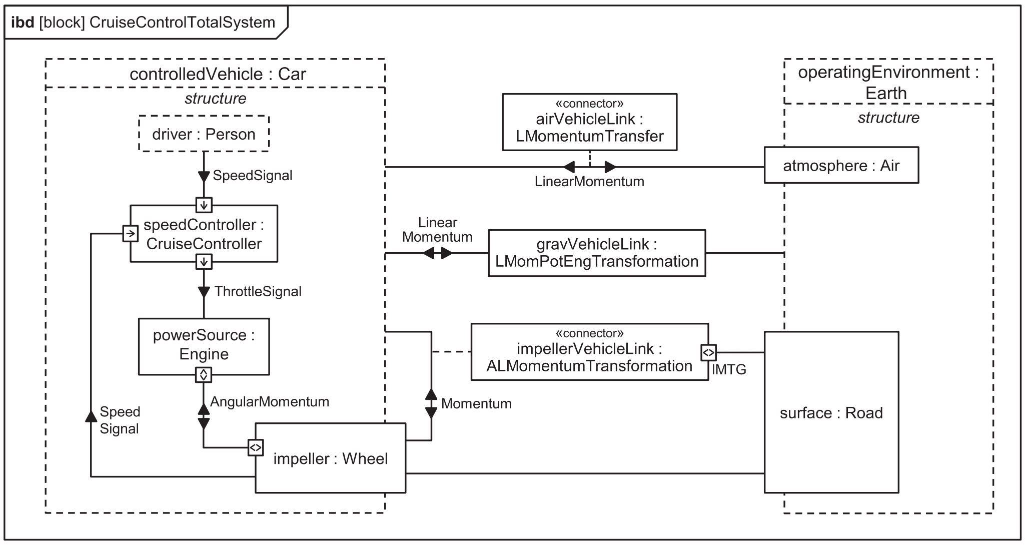

ibd [block] CruiseControlTotalSystem
controlledVehicle : Car
structure
driver : Person
SpeedSignal
speedController : CruiseController
ThrottleSignal
powerSource : Engine
AngularMomentum
Speed Signal
impeller : Wheel
«connector» airVehicleLink : LMomentumTransfer
LinearMomentum
Linear Momentum
gravVehicleLink : LMomPotEngTransformation
«connector» impellerVehicleLink : ALMomentumTransformation
IMTG
Momentum
operatingEnvironment : Earth
structure
atmosphere : Air
surface : Road

Figure 11. Internal block diagram for Figure 10.

while a wheel sends a signal giving the current speed of the vehicle. The cruise controller uses the desired and current speeds to determine how much fuel to inject into the engine subsystem and sends this as a signal to the engine. The engine interacts physically with the wheels, involving angular momentum as the conserved substance. The wheels and road transform angular momentum to and from the car's linear momentum, also a conserved substance. Ideally, this transformation would be complete, but the car interacts with the surrounding air, transferring momentum depending on their relative velocity, and it interacts with the earth gravitationally, reducing or increasing the car's momentum depending on the slope of the road. In addition, rolling resistance, caused by flexing of tires in the wheels, transforms some momentum to heat.

Figure 11 formalizes the components and relationships in Figure 10 as a SysML internal block diagram. Signals control the production of angular momentum by the engine, described more in Section 6.3. Angular momentum typically flows out the wheel and is transformed into linear momentum fed back into the car through interaction with the road. This appears in Figure 11 as a connector between wheel and car supported by an association block specifying the transformation. The car's linear momentum is also affected by gravitation and surrounding air, appearing in Figure 11 as additional connectors. More detail about Figure 11 is in Section 6.2.

6.2. Physical Interaction Definitions

Figure 12 models cars, wheels, and roads for the transformation between angular and linear momentum that happens across the connector between wheel and car in Figure 11. Figure 12 has some of the same properties (roles) and blocks (kinds of things playing the roles) as Figure 11, but they are notated differently because Figure 12 is a block definition diagram. Property names appear in Figure 11 to the left of colons in rectangle labels, and in Figure 12 as names on the ends of the associations, such as driver for the person controlling the car, or in block compartments, such as airVehicleLink for the momentum transfer between the vehicle and air around it. The kinds of things (blocks) playing these roles (typing these properties) are named to the right of colons in property labels in Figure 11 and in the blocks at the ends of associations in Figure 12.[21] Similarly, some connectors in Figure 11 are linked by dashed lines to properties typed by associations that are also blocks. These association blocks appear in Figure 12 linked by dashed lines to the associations they represent. This enables them to have their own properties and internal structure to model transformation of conserved substances.

Connectors in Figure 11 and associations in Figure 12 indicate the kind of things flowing across them with *item flows* (labeled filled triangles), sometimes between ports of components (small rectangles with arrows in Fig. 11 and names at the ends of associations in Fig. 12 indicated as ports). Connectors to the earth and road in Figure 11 reflect their involvement in converting between angular and linear momentum and gravitational potential energy, even though the earth and road are too large to accept or provide momentum.[22] In particular, connectors to the earth and road provide access to properties needed by equations specifying the interactions, such as the gravitation of the earth and slope of the road, and provide a reference for relative velocity, see Figure 14. Generalizations in Figure 12 (arrows headed by unfilled triangles pointing to the more general category) give wheels and cars the characteristics needed for momentum transformation by referring to blocks detailed in Figure 13, see next.

Figure 13 gives more detail on the transformation between angular and linear momentum in Figure 12. It shows features

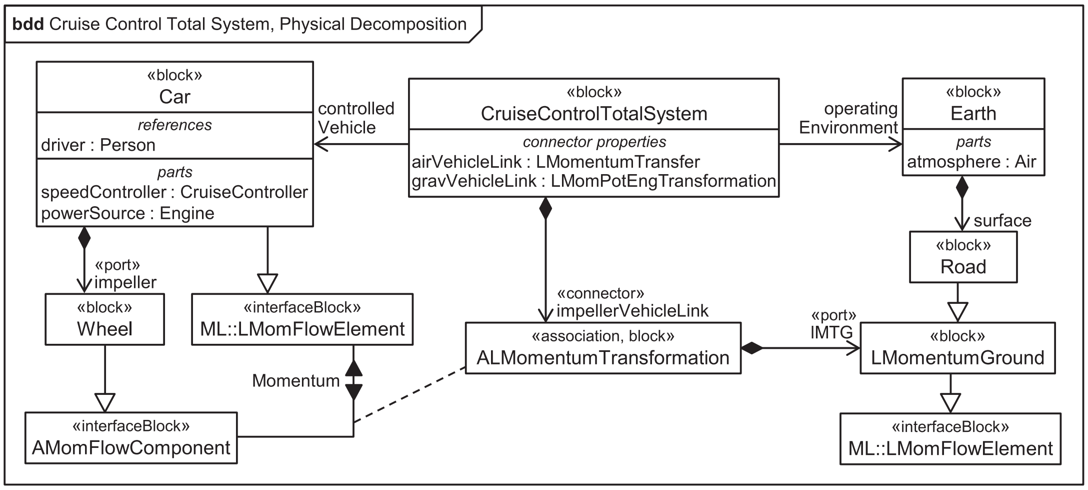


Figure 12. Block definition diagram for some physical elements used in Figure 11.

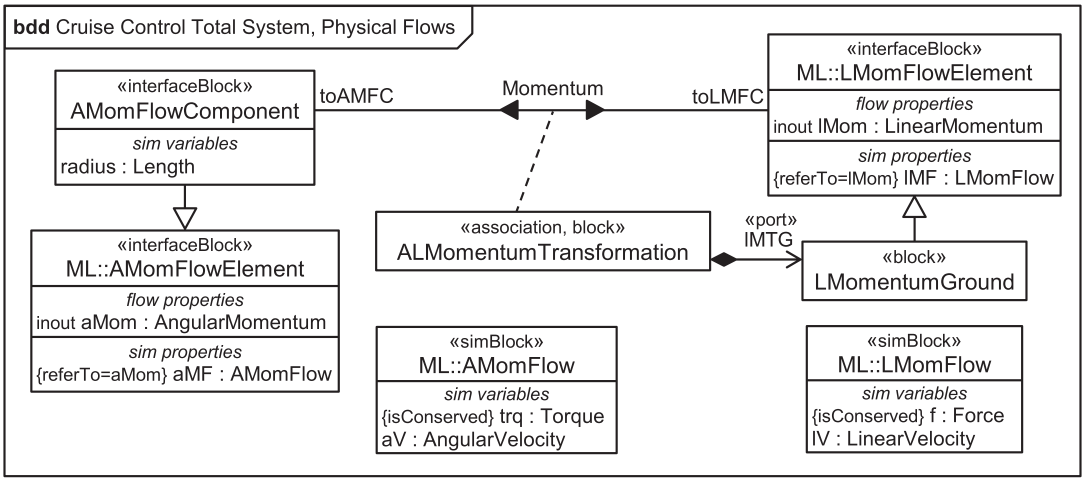


Figure 13. Simulation extension applied to physical interaction elements of Figure 12.

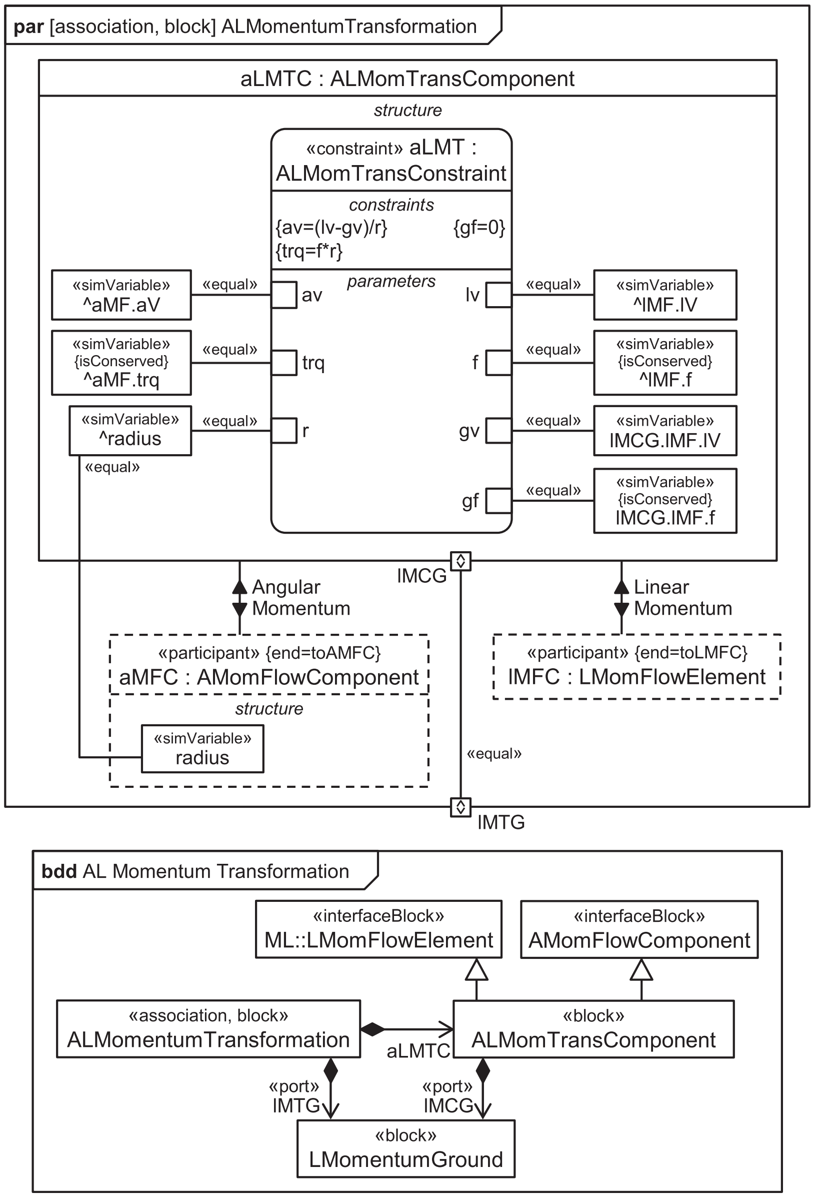

par [association, block] ALMomentumTransformation
aLMTC : ALMomTransComponent
structure
«constraint» aLMT : ALMomTransConstraint
constraints
{av=(lv-gv)/r}
{gf=0}
{trq=f*r}
parameters
av
trq
r
lv
f
gv
gf
«simVariable» ^aMF.aV
«simVariable» {isConserved} ^aMF.trq
«simVariable» ^radius
«simVariable» ^lMF.lV
«simVariable» {isConserved} ^lMF.f
«simVariable» lMCG.lMF.lV
«simVariable» {isConserved} lMCG.lMF.f
«equal»
Angular Momentum
lMCG
Linear Momentum
«participant» {end=toAMFC}
aMFC : AMomFlowComponent
structure
«simVariable» radius
«participant» {end=toLMFC}
lMFC : LMomFlowElement
lMTG
bdd AL Momentum Transformation
«interfaceBlock» ML::LMomFlowElement
«interfaceBlock» AMomFlowComponent
«association, block» ALMomentumTransformation
aLMTC
«block» ALMomTransComponent
«port» lMTG
«port» lMCG
«block» LMomentumGround

Figure 14. Constraints for transforming between angular and linear momentum.

of physical interaction model library elements (prefixed "ML::"), reused from Figure 7 in Section 5.2 (features from library elements can be omitted for brevity when they are reused, but are shown here for reference). The transformation association links the library's linear momentum interface block to a specialization of the library's angular momentum interface block (AMomFlowComponent). The specialized block introduces a simulation variable for radius, which is necessary to transform between angular and linear momentum.[23] Similarly, a block for physical objects too large to accept or provide linear momentum (LMomentumGround) specializes the library's linear momentum interface block. These specialized blocks inherit flow and simulation properties from the library, where the flow properties (aMom and lMom) are typed by the conserved substance flowing (AngularMomentum and LinearMomentum), and the simulation properties (aMF and lMF) give flow rate and potential simulation variables (trq, aV, and f, lV) via simulation blocks (AMomFlow and LMomFlow). The flow rate and potential simulation variables are constrained by equations transforming between angular and linear momentum (ALMomentumTransformation) in Figure 14, see next.

The values of properties in Figure 13 are determined by constraints, which are equations using property names as variables. SysML makes constraints reusable by enclosing them in constraint blocks, where parameters are the constrained properties. Figure 14 shows a constraint block (ALMomTransConstraint) used in a parametric diagram for the association block that transforms between angular and linear momentum in previous figures (block definition diagrams for constraint blocks are omitted for brevity). The constraint block usage (constraint property aLMT) has a compartment for equations (written in the language defined in Section 7.3), which are mathematical relationships between flow rates and potentials of angular and linear momentum (see Table I in the Physical Interaction subsection of 3.2.2). Components reuse constraints by linking their properties (some inherited as indicated with a caret) to parameters of constraint blocks (variables in the equations) with binding connectors to indicate the values of the properties are the same. In Figure 14, a component of the association block (ALMomTransComponent) binds its properties (mostly simulation variables on ports) to constraint parameters, while the association block binds one of them to a property of a participant (the radius of the wheel).[24]

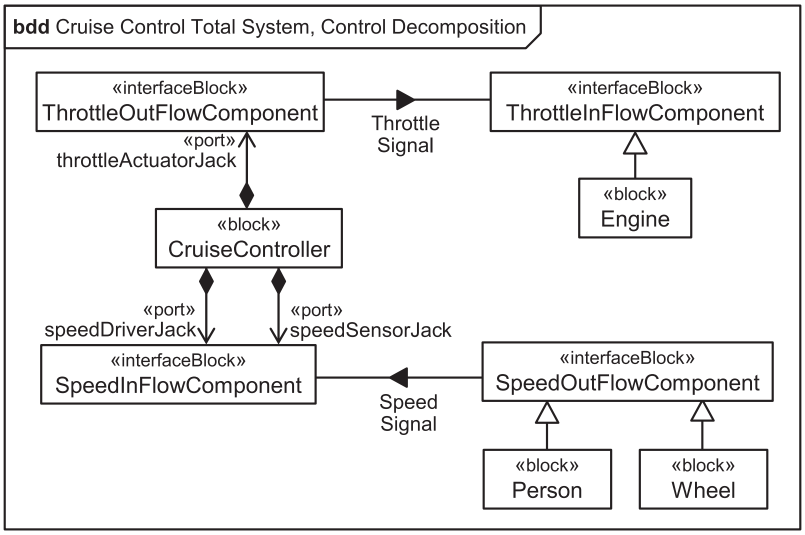

bdd Cruise Control Total System, Control Decomposition
«interfaceBlock» ThrottleOutFlowComponent
«interfaceBlock» ThrottleInFlowComponent
Throttle Signal
«port» throttleActuatorJack
«block» Engine
«block» CruiseController
«port» speedDriverJack
«port» speedSensorJack
«interfaceBlock» SpeedInFlowComponent
«interfaceBlock» SpeedOutFlowComponent
Speed Signal
«block» Person
«block» Wheel

Figure 15. Block definition diagram for some signal flow elements used in Figure 11.

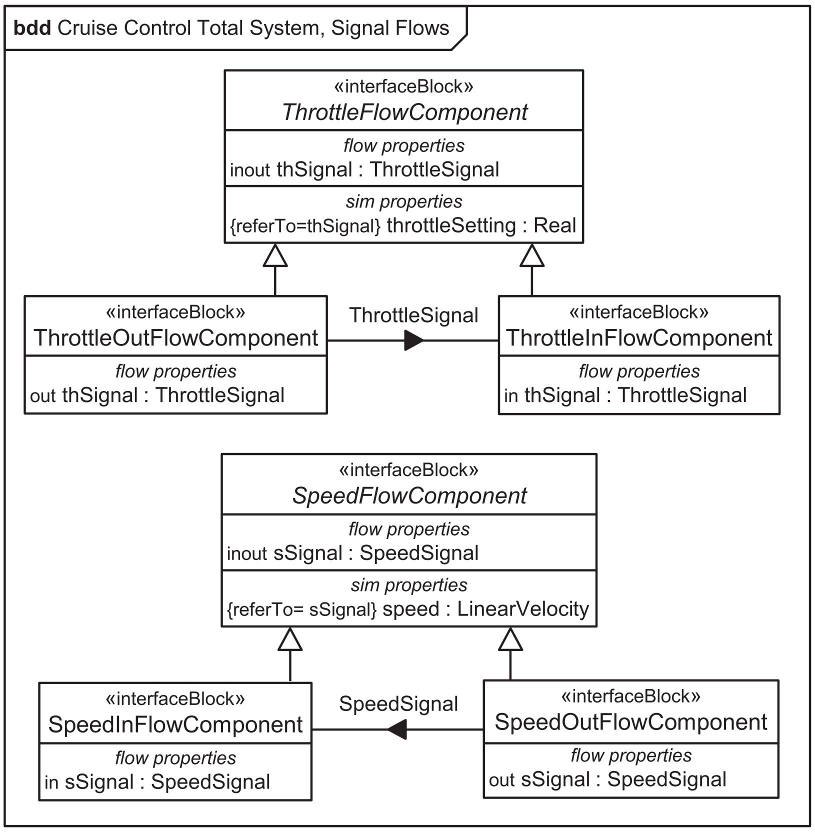

bdd Cruise Control Total System, Signal Flows
«interfaceBlock» ThrottleFlowComponent
flow properties
inout thSignal : ThrottleSignal
sim properties
{referTo=thSignal} throttleSetting : Real
«interfaceBlock» ThrottleOutFlowComponent
flow properties
out thSignal : ThrottleSignal
ThrottleSignal
«interfaceBlock» ThrottleInFlowComponent
flow properties
in thSignal : ThrottleSignal
«interfaceBlock» SpeedFlowComponent
flow properties
inout sSignal : SpeedSignal
sim properties
{referTo= sSignal} speed : LinearVelocity
«interfaceBlock» SpeedInFlowComponent
flow properties
in sSignal : SpeedSignal
SpeedSignal
«interfaceBlock» SpeedOutFlowComponent
flow properties
out sSignal : SpeedSignal

Figure 16. Simulation extension applied to signal flow elements of Figure 15.

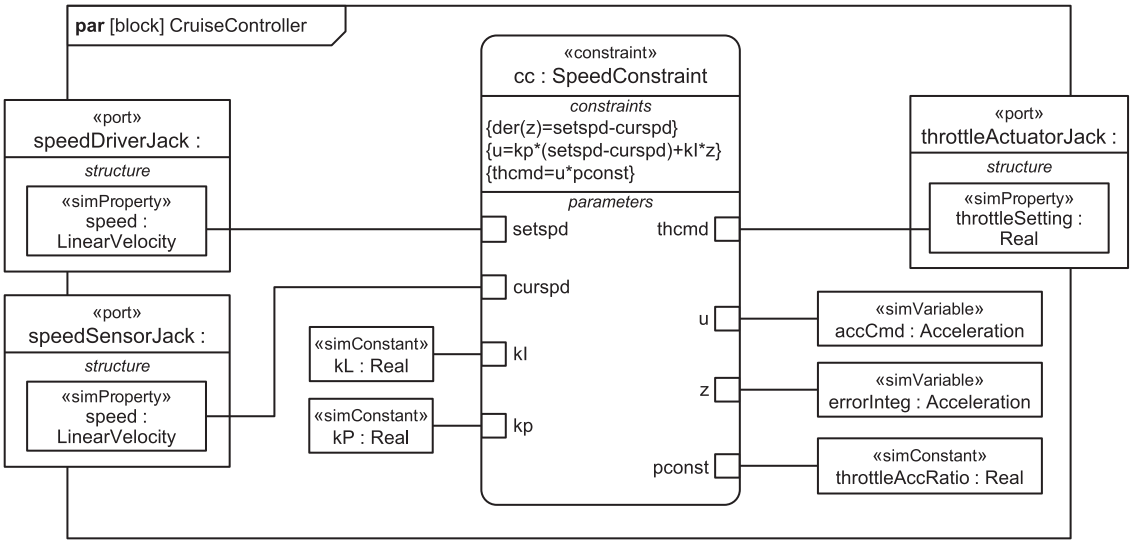


Figure 17. Parametric diagram for cruise controller in Figure 15.

6.3. Signal Flow Definitions

Figure 15 introduces signal flow to cruise controllers and components linked to it in Figure 11 (people, engines, and wheels), using blocks defined in Figure 16. The kinds of signals flowing in Figure 15 appear as types of flow properties thSignal and sSignal for throttle settings and speeds in Figure 16, respectively. These flow properties are referred to by simulation properties throttleSetting and speed, respectively, which will give numeric values sent during simulation (this is the same pattern as the signal flow library in Fig. 8 in Section 5.2, using names reflecting the application). Ports are needed in Figure 15 when the same signal is used for different purposes. For example, cruise controllers receive speed signals from their drivers and wheels, but the first is the goal speed, while the second is the current speed. Ports can be used even if a signal is used for only one purpose, such as the throttle setting sent from cruise controllers to engines, but generalization can also be used in this case. For example, engines receive throttle setting signals for one purpose, and are modeled as a kind of ThrottleInFlowComponent to reflect this.

Equations for the values of properties in Figure 16 appear in Figure 17 in a constraint block used in a parametric diagram for cruise controllers (the example is proportional-integral control [Dorf and Bishop, 2016]). Parameters of the constraint block are bound to simulation properties in the ports, giving numeric signal values, and to internal properties of the controller.

7. TRANSLATING BETWEEN SysML AND SIMULATION PLATFORMS

SysML models extended according to Section 5 can be translated to and from text files for simulation platforms, in particular the widely used Simulink, Simscape, and Modelica platforms. Issues arising in these translations are:

- Some SysML modeling capabilities are supported on only some of the simulation platforms above and sometimes only partially, which is addressed in Section 7.1, while some SysML capabilities are not supported on any of the platforms, addressed in Section 7.2.
- The model libraries in Section 5.2 have corresponding libraries on simulation platforms, many of which can be used in translation, but are different across platforms, as described in Section 7.2.
- The mathematical equation language in Section 5.3 is different in minor ways that require translation, as covered in Section 7.4.
- Input conditions must be provided for simulation, addressed in Section 7.5.

Section 7.6 describes an implementation of the SysML extension and Sections 7.1 through 7.4, validating these with the example in Section 6 and scenario in Section 7.5.

7.1. Platform Modeling

Simulation platforms use various terms corresponding to SysML extended according to Section 5, as summarized in Table III. Some differ between platforms and most are different than SysML. Some SysML and simulation terms are the same, but with different meanings. For example, SysML and Modelica have constructs called connectors, but in SysML this refers to links between components on internal block diagrams, while in Modelica it refers to elements on components that specify interactions with other components. Another example is simulation parameters, which are variables with values that do not change during simulation, but in SysML are properties of constraint blocks or inputs and outputs of behaviors. From a graphical perspective, diagrams in SysML and simulation tools have some constructs that appear the same visually, but have different meanings. For example, both have small rectangles or other shapes that appear on boundaries of components. In SysML, these are structural

Table III. Correspondences between Terms in Extended SysML and Example Simulation Languages (NA = not available)

Extended SysML	Modelica	Simulink/Simscape
Blocks without internal block diagrams	Models without connections	Block types/Components
Blocks with internal block diagrams	Models with connections	Systems/Components
Part properties	Model instances (informal)	Reference blocks/Component instances
Connectors	Connections	Lines/Connections
Generalization	Extension	NA/Subclassing
Redefinition	Redeclaration	NA
SimConstants	Parameters	Parameters
Ports with SimProperties referring to signal flow properties (in and out direction)	Input, output variables	Inport, outport/Input, output variables
Ports with SimProperties referring to physical (inout) flow properties	Ports (informal)	Connection ports/Nodes
Ports without SimProperties	NA	NA
Flow properties	NA	NA
SimBlocks	Connectors	NA/Domains
Conserved SimVariables (flow property direction inout)	Flow variables	NA/Balancing variables
Non-conserved SimVariables (flow property direction inout)	Variables	NA/Variables
ConstraintBlocks	NA	NA
ConstraintBlock usages	Equations	S-functions/Equations
Value types	Data types	Data types

elements (ports) involved in particular interactions with other components, while in simulation models they only describe flows, nothing structural.

Simulation platforms differ in support for an equivalent to SysML generalization. Modelica supports the equivalent of multiple generalizations for each component definition, as SysML does, while Simscape supports only a single generalization per component, and Simulink does not support any generalizations. When a generalization in a SysML model cannot be supported on a simulation platform, the simulation model corresponds to a SysML model that replicates the properties of general blocks on special ones (except for properties redefined by the special blocks, see below), and omits the general blocks (see the Simulink translations in Section 7.6.2 for the generalizations in Section 7.5). Modelica treats components that include all features of another component as generalizations of that other component, while SysML and Simscape require generalization to be explicitly included. This aspect of Modelica depends on consistent feature naming across components, which is not reliable in large, distributed projects, making it unsuitable for platform-independent approaches as in this article.

Simulation platforms differ in support for an equivalent to SysML property redefinition (a way of restricting properties inherited from more general blocks). Modelica supports an equivalent of SysML property redefinition. Simulink does not support redefinition, because it does not support generalization, which is addressed by the correspondences for generalization above. Simscape does not support an equivalent of property redefinition, though it supports single generalization per component. Generalizations of SysML blocks with property redefinitions correspond to Simscape in the same way as Simulink (see the Simscape translations in Section 7.6.2 for the redefinitions in Sections 7.5),[25] while generalizations of SysML blocks without redefinitions is addressed by the correspondences for generalization above.

Simulation platforms differ in support for an equivalent to SysML properties and operations that that are only accessible to behaviors and constraints defined on the block owning them and specializations of that block (*protected* properties and operations). Modelica modeling supports the equivalent of these, while Simscape does not. This is an advanced object-oriented capability that is not typically used in systems modeling. Simulation models that correspond to SysML models can equivalently have regular properties corresponding to protected ones in SysML, assuming SysML models are defined in tools that ensure protected properties and operations are accessed properly.

Simulation platforms differ between themselves and with SysML in data types they support. SysML and Modelica support integers (at least signed 32-bit in Modelica), reals (double-precision floating point in Modelica), strings, and Booleans, enumerations, as well as complex data types (called records with operators in Modelica, and value types with properties and operations in SysML). Simulink and Simscape do not support strings or complex data types. Simulink supports some number types that SysML, Simscape, and Modelica do not (single precision floating point, fixed point floating, unsigned/signed 8-/16-bit, and unsigned 32-bit integers). Simscape only supports real numbers (doubles), though Simulink's number types can be used in Simulink models that contain Simscape components. To enable translation to all platforms, the extension in Section 5 assumes SysML value types as they are, excluding strings and complex data types, and treating reals as doubles and integers as unsigned 32-bit. For Simscape, integers are treated as reals.

Some simulation platforms have capabilities that SysML and the other platforms do not, which are not included in the SysML extension:

- Simulink supports a kind of input port that enables or prevents components from reacting to the rest of their inputs.
- Simscape supports automatically setting values for parameters in domains. Values for parameters in Simscape domains can be copied in various ways across the model.

7.2. SysML Expressiveness

Translating extended SysML models to simulation platforms based only on the correspondences in Section 7.1 would produce unsimulatable models in many cases, because SysML supports a wider range of system structures than simulation platforms do. Some SysML structures that simulation modeling does not support are:

1. Association blocks. Associations in SysML can have their own interconnected parts, and be used as the type of properties, just as blocks can. The example in Sections 6.1 and 6.2 uses an association block containing properties and equations specifying the interaction between cars, wheels, and roads.
2. Defining flow properties with other properties on the same block, or having them both inherit to a block used to type ports. For example, wheels on cars support angular momentum flow as well as other characteristics, such as their radius. Modelers can separate flow properties from others by specializing library elements from Section 5.2, as in the example in Section 6.2, but the specializations cannot be used to type ports, as wheels are in that example.
3. Flow/simulation properties on components. For example, cars interacting with their environments can be modeled in SysML as having flow properties for momentum gained and lost to the environment without defining ports specifically for these interactions, as in Figure 11 (item flows imply flow properties on the linked components).
4. Nested ports. For example, the wheels in Figure 11 could have a port for inflating tires, connected to a pump in an internal block diagram for car maintenance.

Fortunately, the SysML structures above can be automatically prepared (*preprocessed*) before using the correspondences in Section 7.1 for translation. This enables modelers to use the structures above in SysML and still produce working simulation models. The preprocessing steps are:

1. Change association blocks into components. Figure 18 shows this preprocessing step applied to portions of Figures 11 and 12 in Sections 6.1 and 6.2. Some of the associations in those diagrams have their own interconnected parts, which simulation platforms do not support. The top two diagrams in Figure 18 replace one of these association blocks with its contents from Figure 14, making it into a component like the oth-

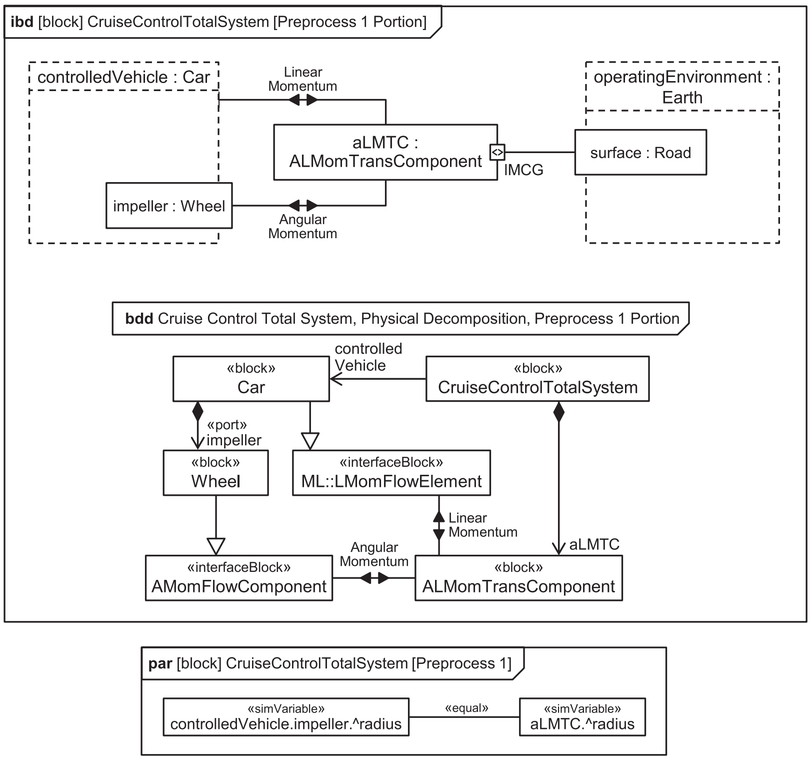
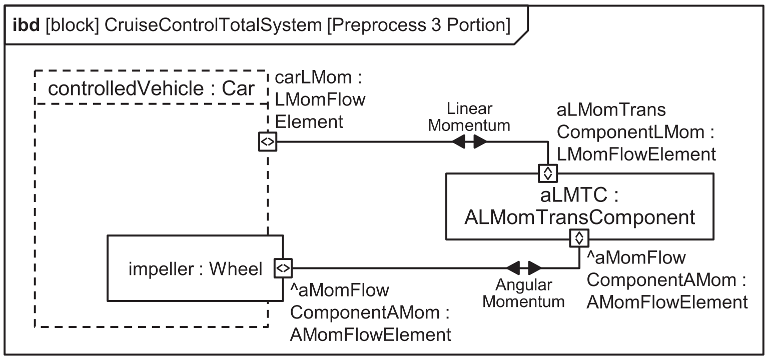

ibd [block] CruiseControlTotalSystem [Preprocess 1 Portion]
controlledVehicle : Car
Linear Momentum
operatingEnvironment : Earth
aLMTC : ALMomTransComponent
IMCG
surface : Road
impeller : Wheel
Angular Momentum
bdd Cruise Control Total System, Physical Decomposition, Preprocess 1 Portion
«block» Car
controlled Vehicle
«block» CruiseControlTotalSystem
«port» impeller
«block» Wheel
«interfaceBlock» ML::LMomFlowElement
Linear Momentum
aLMTC
«interfaceBlock» AMomFlowComponent
Angular Momentum
«block» ALMomTransComponent
par [block] CruiseControlTotalSystem [Preprocess 1]
«simVariable» controlledVehicle.impeller.^radius
«equal»
«simVariable» aLMTC.^radius

Figure 18. Replacing association blocks in Figures 11 and 12 with their contents.


ibd [block] CruiseControlTotalSystem [Preprocess 3 Portion]
controlledVehicle : Car
carLMom : LMomFlow Element
Linear Momentum
aLMomTrans ComponentLMom : LMomFlowElement
aLMTC : ALMomTransComponent
impeller : Wheel
^aMomFlow ComponentAMom : AMomFlowElement
Angular Momentum
^aMomFlow ComponentAMom : AMomFlowElement

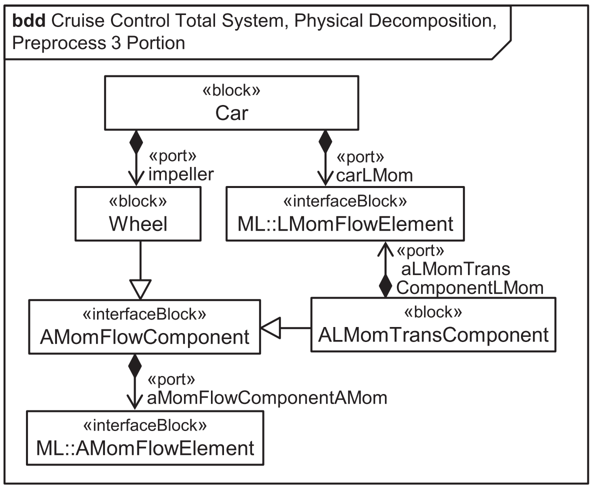

bdd Cruise Control Total System, Physical Decomposition, Preprocess 3 Portion
«block» Car
«port» impeller
«port» carLMom
«block» Wheel
«interfaceBlock» ML::LMomFlowElement
«port» aLMomTrans ComponentLMom
«interfaceBlock» AMomFlowComponent
«block» ALMomTransComponent
«port» aMomFlowComponentAMom
«interfaceBlock» ML::AMomFlowElement

Figure 19. Moving simulation/flow properties in Figure 18 from components to ports.

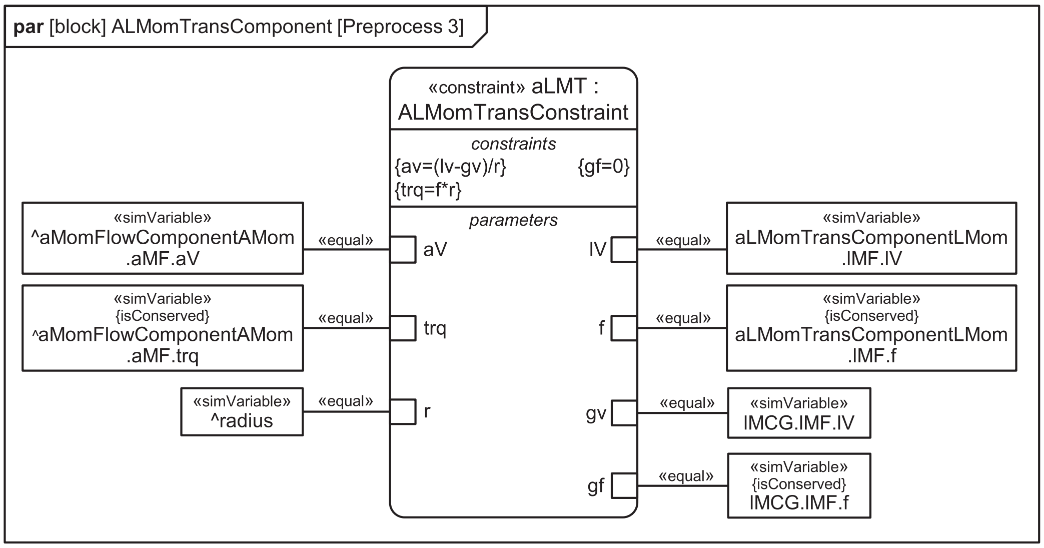

par [block] ALMomTransComponent [Preprocess 3]
«constraint» aLMT : ALMomTransConstraint
constraints
{av=(lv-gv)/r} {gf=0}
{trq=f*r}
parameters
aV
trq
r
lV
f
gv
gf
«equal»
«simVariable» ^aMomFlowComponentAMom .aMF.aV
«simVariable» {isConserved} ^aMomFlowComponentAMom .aMF.trq
«simVariable» ^radius
«simVariable» aLMomTransComponentLMom .lMF.lV
«simVariable» {isConserved} aLMomTransComponentLMom .lMF.f
«simVariable» lMCG.lMF.lV
«simVariable» {isConserved} lMCG.lMF.f

Figure 20. Changes to constraint bindings in Figure 14 for Figure 19.

ers in the total system, which simulation platforms support. Connectors in the association block are moved to the total system. The connectors linked to participant properties in the association block are changed to link directly to those participants in the total system. The binding connector in the association block is moved to a parametric diagram for the total system at the bottom of Figure 18.

2. Separate flow/simulation properties from others on the same block and move ports to parts when they are typed

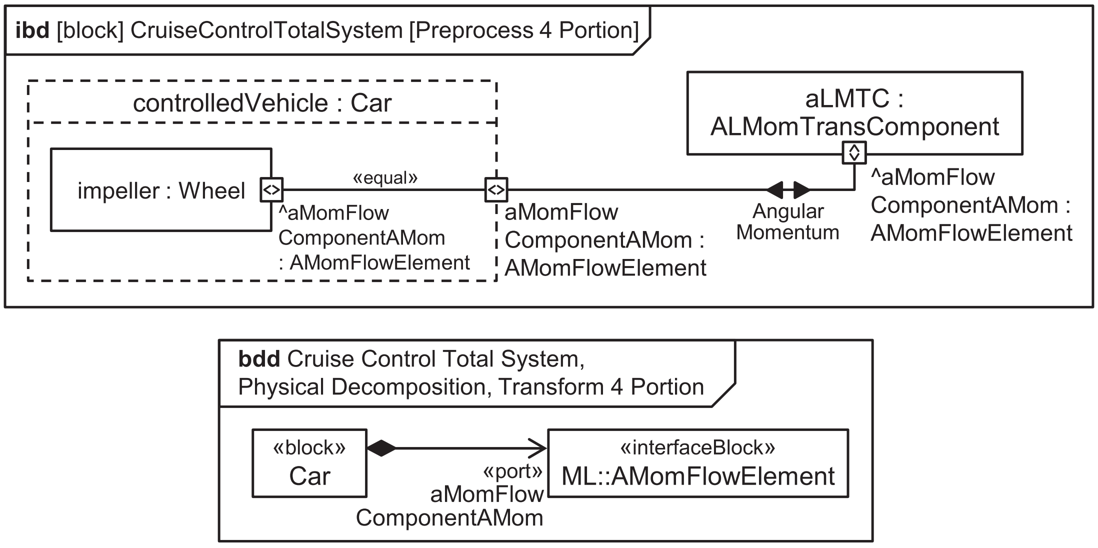

ibd [block] CruiseControlTotalSystem [Preprocess 4 Portion]
controlledVehicle : Car
impeller : Wheel
«equal»
^aMomFlow ComponentAMom : AMomFlowElement
aMomFlow ComponentAMom : AMomFlowElement
Angular Momentum
aLMTC : ALMomTransComponent
^aMomFlow ComponentAMom : AMomFlowElement
bdd Cruise Control Total System, Physical Decomposition, Transform 4 Portion
«block» Car
«port» aMomFlow ComponentAMom
«interfaceBlock» ML::AMomFlowElement

Figure 21. Reducing port nesting in Figure 19.

by blocks inheriting both. The example in Section 6.2 separates flow/simulation properties from others by specializing component interaction model library elements, see Figure 13. Models that do not specialize from the library must be preprocessed by moving simulation/flow property pairs to their own blocks that generalize the original one. This preprocessing step does not change the effect of the model, because simulation/flow properties inherit to the blocks they were defined on before. The example in Section 6.2 types a port by a wheel block that combines flow/simulation properties with others by inheritance. This must be moved to be a part, with a binding to a port that only exposes flow/simulation properties, see preprocessing steps 3 and 4.

3. Move flow/simulation properties from components to ports. Figure 19 shows this preprocessing step applied to cars, wheels, and momentum transformation in Figure 18 (roads are omitted for brevity). These components have simulation/flow properties, which simulation platforms do not support (wheels are treated as ports in this model, but they mix flow/simulation properties with others, which simulation platforms also do not support). The diagrams in Figure 19 move these properties to new ports on cars, wheels, and momentum transformation, and change connectors to link to them, which simulation platforms support. The new port properties replace generalizations from the component blocks. Constraint bindings using these flow/simulation properties change to link to the same properties in their new nested positions. Figure 20 shows this applied to the constraint on momentum transformation in Figure 14. Constraint bindings link to same properties as before, but on the new ports.
4. Reduce nesting of ports. Figure 21 shows this preprocessing step applied to portions of Figure 19. The port on wheels in Figure 19 is nested two layers down in the total system, under the wheel and car, which simulation platforms do not support. The diagrams in Figure 21 change the wheel to be an internal part, replacing it with a new port of the same type as the wheel's, which simulation platforms support. The two ports are bound together, and the external connector linked to the new port, providing the same effect as the nested port in Figure 19. Nothing else in cars changes, because wheels are still playing the impeller role.

SysML models using the capabilities described in this section can be processed through the steps above, then translated to simulation platforms according to the correspondences in Section 7.1 and 7.3. Section 7.6 uses preprocessing in an example translation.

7.3. Model Libraries

Simulation platforms have model libraries corresponding to those in Section 5.2, many of which can be reused in translation and some of which cannot:

- Physical interaction libraries on simulation platforms have elements corresponding to all the simulation blocks in Figure 7, but some platforms use flow rates or potentials that do not multiply to rate of energy flow in some cases, as required for physical interaction models. For example, Modelica uses linear and angular position for potential of linear and angular momentum, respectively, which do not produce energy flow rate when multiplied by force and torque, respectively. Similarly, Simscape and Modelica use heat energy flow rate, which does not produce energy flow rate when multiplied by temperature like entropy flow rate does. Velocity and entropy flow rate can be derived from position and heat flow rate, respectively, to give an energy rate product, but the SysML extension opts to use velocities and entropy flow rates to more closely reflect the potential and flow rate of conserved substances. These choices affect how component equations are written, which are difficult to adapt to the platforms during translation. For

Table IV. Property Values for Stereotypes in Figure 22 Applied to Signal Behavior Library Elements in Figure 9

SysML Extension Block	Simulink Block Type	Modelica Block	SysML Extension Properties	Simulink Parameters	Modelica Parameters
Derivative	Derivative	Continuous.Derivative	init	InitialCondition	y_start
Integrator	Integrator	Continuous.Integrator			
SineWave	Sin	Sources.Sine	amplitude offset frequency phase	Amplitude Bias Frequency Phase	amplitude offset freqHz phase
RealScope	Scope	Interaction. Show.RealValue			
AND	Logic	Logical.And			
OR	Logic	Logical.Or			
BooleanScope	Scope	Interaction. Show.BooleanValue			

Table V. Property Values for Stereotypes in Figure 22 Applied to Physical Behavior Library Elements in Figure 9

SysML Extension Block	Simscape Component	Modelica Model	SysML Extension Ports	Simscape Nodes	Modelica Ports	SysML Extension Properties	Simscape Parameters	Modelica Parameters
Resistor	foundation. electrical. elements. resistor	Electrical. Analog. Basic. Resistor	p n	p n	p n	r	R	R
Capacitor	foundation. electrical. elements. capacitor	Electrical. Analog. Basic. Capacitor	p n	p n	p n	c	c r=0 g=0	C
Ground	foundation. electrical. elements. reference	Electrical. Analog. Basic. Ground	p	V	p			

simplicity, the physical interaction library is translated to the platforms as any other model would be, without reusing physical interaction elements in platform libraries. See Section 8 for other approaches to this.

- Component behavior libraries on simulation platforms have elements corresponding to all of those in the SysML extension (some of which are shown in Fig. 9), but the names of these components and their features vary across platforms. The stereotypes in Figure 22 are applied to SysML extension behavior library elements to specify correspondences with Simulink, Simscape, and Modelica libraries. Tables IV and V give values of stereotype properties applied to the signal and physical components in Figure 8 and Figure 9, respectively. Columns for port names do not appear in Table IV because Simulink numbers its ports rather than naming them (Modelica's port name is used in the SysML extension in this case). Some of the columns in Table V refer to Simscape, because the rows are physical components, but Simulink stereotypes are reused to model this, for simplicity. The capacitor in Table V shows specification of values for two Simscape parameters to ensure the behavior is the same as in Modelica. The multidimensional stereotype on the lower right of Figure 22 is applied to properties in the SysML extension behavior library that will have nonscalar values, such as vectors and matrices. The dimension property is given a list of values when applied, where the length of the list is the nonscalar dimension and the elements of the list are the size of each dimension. This enables values of the property to which the stereotype is applied to form a list that can be parsed into nonscalar form. The multidimensional stereotype is used in publicly available SysML extension behavior library [Barbau and Bock, 2017].

Simulation platforms do not have libraries corresponding to Figure 8 (signal flow), because platform signal components only use the equivalent of the simulation properties, not flow properties (see Fig. 5 in Section 4).

Some simulation platform libraries have elements not available in others, and are not included in the SysML extension:

- Simscape and Modelica physical interaction libraries include elements for specifying fluid and gas interactions with two pairs of flow rate and potential variables, specifically, mass flow rate and pressure combined with variable pairs for heat, energy, or enthalpy, depending on the platform. The differences could be addressed in the SysML extension libraries (see Section 8).
- Simscape and Modelica physical component behavior libraries have only electric components in common. For example, their mechanical libraries are not compatible

Table VI. Correspondences between Modelica and MATLAB Conditional Syntax

Modelica	MATLAB
`if ...`	`if ...`
`then ...`	`...`
`elseif ...`	`elseif ...`
`then ...`	`...`
`else ...`	`else ...`
`end if`	`end`

because they use different potential variables for momentum (see beginning of this section). Another example is the components for friction, which includes a breaking force in Simscape, where friction drops significantly, but not in Modelica (even for zero breaking force the components behave differently). Simulink signal component libraries currently have some filter blocks and signal sources that Modelica does not, for example, discrete (in)finite impulse response filters and white noise generators.

A few Simulink and Modelica signal component behavior library elements intended for the same purpose have slightly different behavior. For example, Simulink's derivative component use the simulation time step, whereas Modelica's uses a modeler defined step. The behavior of the components will only be the same when Modelica's derivative component is given Simulink's step value (see Section 8). This does not affect the `der` function in the equation language of Section 5.3, which solvers treat as integration, and behaves the same on both simulation platforms.

Units are beyond the scope of this article, but to validate the SysML extension, the current implementation (see Section 7.6.1) includes a unit library, because SysML's library of units is nonnormative. The implementation uses specialized real number value types named according to the kind of physical quantities being specified, such as force or angular velocity, and giving text for particular units of these in Simscape format. Modelica does not support units (only modeler-defined strings indicating units, which are not used by Modelica) and Simulink only recently began to support units (see Section 8).

7.4. Equation Languages

The equation language for Simulink and Simscape (MATLAB) differs from the Modelica subset in Section 5.2 in minor ways that have simple correspondences:

- The syntax for conditionals (`if_equation` in the grammar subset) is different in MATLAB, as shown in Table VI. Modelica uses a `then` keyword, MATLAB does not.
- Some terminal symbols for operators in the subset above are different in MATLAB, as shown in Table VII.

Section 7.6 uses the correspondences in an example translation.

Many functions are defined in MATLAB or Modelica, but not both, and several MATLAB operators are not available in Modelica (too many to list here). These are not included in the equation language of Section 5.2 (see Section 8).

Table VII. Correspondences between Differing Modelica and MATLAB Operators

Modelica	MATLAB
`=`	`==`
`<>`	`~ =`
`not`	`~`
`and`	`&&`
`or`	`||`
`: =`	`=`
`div`	`idivide`

7.5. Simulation Scenarios

SysML models often intentionally omit information to widen their applicability, some of which is required for simulation. For example, the model in Section 6 does not specify values for:

- internal constants in the cruise controller used by the equations in Figure 17, enabling the model to be augmented with any values for these constants.
- wheel radius used by the equations in Figure 14, enabling the model to be augmented with wheels of any radius.
- speed selected by the driver, enabling the model to be augmented with any speed or variation of speed.

In general, models might omit values for simulation constants, equations for some simulation variables, and properties in the operating environment, and these values must be given for simulation to be carried out.

The additional information above needed for simulation (*scenarios*) can be given in separate models that refer to the original, but do not modify it, enabling the original model to be reused under multiple scenarios. Figure 23 shows a partial scenario for the example in Section 6. It defines new elements that add to information inherited from blocks in the original model. The first of these is for the scenario's total system, CruiseControlTotalSystemScenario1, generalized by the original total system block on the upper left. It uses property redefinition (a way of restricting properties inherited from more general blocks) to restrict the controlled vehicle to a specialized kind of car, Car1, which is generalized by the original car block on the lower left. The specialized kind of car uses property redefinition to provide values for:

- Internal constants in the cruise controller, by restricting the speed controller to a specialized kind of cruise control (CruiseControl1) that assigns the values as defaults. The default values will not change during simulation, because the properties are simulation constants.
- Wheel radius, by restricting the impeller to a specialized kind of wheel (Wheel1) that gives the radius. The radius is redefined as a simulation constant to ensure the value does not change during simulation.

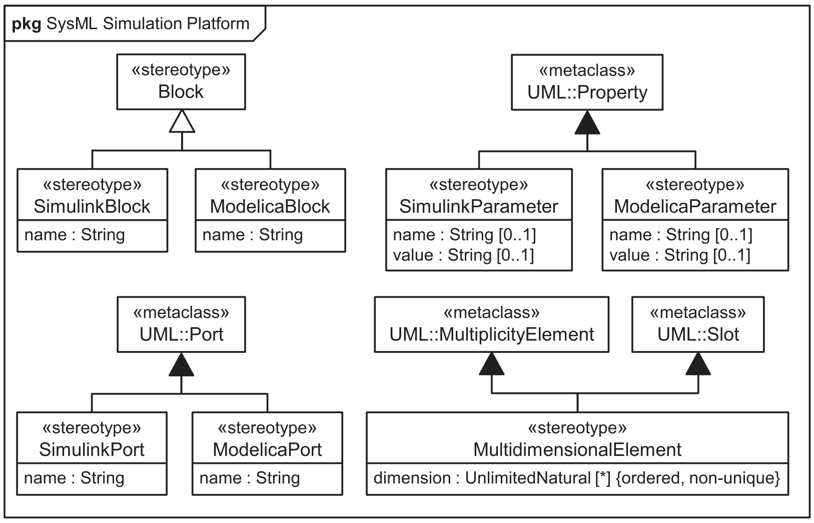

pkg SysML Simulation Platform
«stereotype» Block
«metaclass» UML::Property
«stereotype» SimulinkBlock
name : String
«stereotype» ModelicaBlock
name : String
«stereotype» SimulinkParameter
name : String [0..1]
value : String [0..1]
«stereotype» ModelicaParameter
name : String [0..1]
value : String [0..1]
«metaclass» UML::Port
«metaclass» UML::MultiplicityElement
«metaclass» UML::Slot
«stereotype» SimulinkPort
name : String
«stereotype» ModelicaPort
name : String
«stereotype» MultidimensionalElement
dimension : UnlimitedNatural [*] {ordered, non-unique}

Figure 22. Stereotypes for specifying correspondences between component behavior libraries in the SysML extension and simulation platform libraries.

- Speed selected by the driver, by restricting the driver to a specialized kind of person (Person1) that gives the speed as a function of time. This is done with a constraint (details omitted for brevity) on a signal property of a port introduced by preprocessing the original block for persons (see preprocessing step 3 in Section 7.3).

Redefinition requires translation to accommodate simulation platforms that do not support it (see Section 7.1). For example, the models on these platforms correspond to a SysML model that omits CruiseControlTotalSystem, moving its properties and connectors to CruiseControlTotalSystem1, except for the controlledVehicle property, which is already redefined there (see the Listing 2 in Section 7.6.2).

Executable simulation files can be automatically generated from a SysML model combining the original one in Section 6 and the scenario specializations in Figure 23, as demonstrated in Section 7.6.

7.6. Validation

This section describes validation of the SysML extension in Section 5, preprocessing in Section 7.2, and translations in Sections 7.1, 7.3, and 7.4 to Simulink, Simscape, and Modelica. Section 7.6.1 covers a software implementation, while Section 7.6.2 gives some results of applying it to the example in Section 1 and scenario in Section 7.5, then executing the simulation files on their corresponding tools to show the results are the same.

7.6.1. Implementation

The preprocessing and translations in Sections 7.1 through 7.4 are carried out in software developed by one of the authors (Barbau), as illustrated in Figure 24 (block arrows along the top show the generation process, with supporting architecture

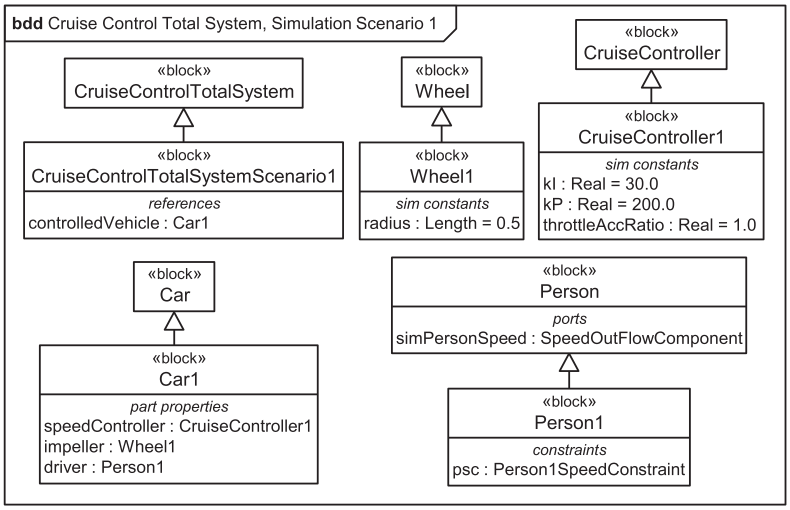
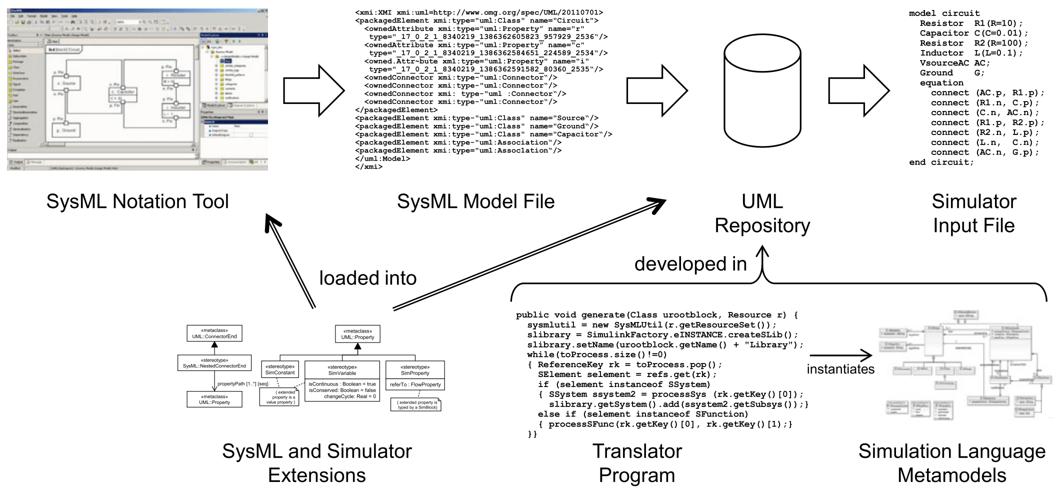

bdd Cruise Control Total System, Simulation Scenario 1
«block» CruiseController
«block» CruiseControlTotalSystem
«block» Wheel
«block» CruiseController1
sim constants
kI : Real = 30.0
kP : Real = 200.0
throttleAccRatio : Real = 1.0
«block» CruiseControlTotalSystemScenario1
references
controlledVehicle : Car1
«block» Wheel1
sim constants
radius : Length = 0.5
«block» Car
«block» Person
ports
simPersonSpeed : SpeedOutFlowComponent
«block» Car1
part properties
speedController : CruiseController1
impeller : Wheel1
driver : Person1
«block» Person1
constraints
psc : Person1SpeedConstraint

Figure 23. Simulation scenario.


SysML Notation Tool
SysML Model File
UML Repository
Simulator Input File
loaded into
developed in
instantiates
«metaclass» UML::ConnectorEnd
«stereotype» SysML::NestedConnectorEnd
propertyPath [1..*] {seq}
«metaclass» UML::Property
«metaclass» UML::Property
«stereotype» SimConstant
{ extended property is a value property }
«stereotype» SimVariable
isContinuous : Boolean = true
isConserved: Boolean = false
changeCycle: Real = 0
«stereotype» SimProperty
referTo : FlowProperty
{ extended property is typed by a SimBlock}
SysML and Simulator Extensions
Translator Program
Simulation Language Metamodels

```
<xmi:XMI xmi:uml=http://www.omg.org/spec/UML/20110701>
<packagedElement xmi:type="uml:Class" name="Circuit">
  <ownedAttribute xmi:type="uml:Property" name="r"
   type="_17_0_2_1_8340219_1386362605823_957929_2536"/>
  <ownedAttribute xmi:type-"uml:Property" name="c"
   type="_17_0_2_1_8340219_1386362584651_224589_2534"/>
  <owned.Attr~bute xm1:type="uml:Property" name="i"
   type="_17_0_2_1_8340219_1386362591582_80360_2535"/>
  <ownedConnector xmi:type-"uml:Connector"/>
  <ownedConnector xmi:type-"uml:Connector"/>
  <ownedConnector xmi: type-"uml :Connector"/>
  <ownedConnector xmi:type-"uml:Connector"/>
</packagedElement>
<packagedElement xmi:type-"uml:Class" name="Source"/>
<packagedElement xmi:type-"uml:Class" name="Ground"/>
<packagedElement xmi:type-"uml:Class" name="Capacitor"/>
<packagedElement xmi:type-"uml:Association"/>
<packagedElement xmi:type="uml:Assoclation"/>
</uml:Model>
</xmi>
```

```
model circuit
  Resistor  R1(R=10);
  Capacitor C(C=0.01);
  Resistor  R2(R=100);
  Inductor  L(L=0.1);
  VsourceAC AC;
  Ground    G;
  equation
    connect (AC.p, R1.p);
    connect (R1.n, C.p);
    connect (C.n, AC.n);
    connect (R1.p, R2.p);
    connect (R2.n, L.p);
    connect (L.n,  C.n);
    connect (AC.n, G.p);
end circuit;
```

```
public void generate(Class urootblock, Resource r) {
  sysmlutil = new SysMLUtil(r.getResourceSet());
  slibrary = SimulinkFactory.eINSTANCE.createSLib();
  slibrary.setName(urootblock.getName() + "Library");
  while(toProcess.size()!=0)
  { ReferenceKey rk = toProcess.pop();
    SElement selement = refs.get(rk);
    if (selement instanceof SSystem)
    { SSystem ssystem2 = processSys (rk.getKey()[0]);
      slibrary.getSystem().add(ssystem2.getSubsys());}
    else if (selement instanceof SFunction)
    { processSFunc(rk.getKey()[0], rk.getKey()[1];}
  }}
```

Figure 24. Generation process and generator architecture.

below). The example models are drawn in a SysML tool that supports export to an OMG standards-compliant file format [NoMagic 2015] [Object Management Group 2015b] [Object Management Group 2017b]). These files are loaded into a repository supporting UML's meta-model (including profiles such as SysML and the ones in the SysML extension) [Steinberg et al., 2008]. Repositories create instances of UML meta-classes and stereotypes according to the input files, including links between them for stereotype application, and give programmatic access to these instances [Bock, 2003]. The generator program (written in the language of the repository [Gosling et al., 2014]) accesses the instances, applies the preprocessing of Section 7.2 to them, and emits simulation input files according to the correspondences in Sections 7.1, 7.3, and 7.4.

Development of the translation portion of the generator was facilitated by building meta-models of Simulink, Simscape, and Modelica in the repository, which supports meta-model construction and generation of skeleton class files corresponding to the meta-models, as illustrated in the lower right of Figure 24. Operations were defined manually on visitors of each class [Gamma et al., 1998], with methods for generating files in the format of each simulation language. The translation program reads instances of UML's meta-classes in the repository, as well as SysML and simulation stereotype instances linked to them, finds instances for extended SysML concepts that map into simulation languages, as given Table III, instantiates the corresponding simulation language meta-classes, and links them together according to the correspondences in Sections 4 and 5. Once the simulation model is created, the generation operations on visitors are invoked to produce files in simulator input formats.

The software above generates simulation files for the example in Section 6 and scenario in Section 7.5 in around 10 seconds on a high-end travel laptop at time of writing this article (see Section 7.6.2 for portions of these files). Most of the time is taken in preprocessing, which scans the SysML model for elements matching the patterns in Section 7.2 and prepares them for translation. Translation then takes one or two seconds. The software can also translate from generated simulation files to extended SysML models and back to simulation files that execute the same way as the originals. The generation process and program was developed only to validate the proposed simulation extension and is not claimed to be optimal in any respect. The software with documentation and examples are publicly available, including the simulation platform meta-models and source code, as well as translations for state machines not covered in this article (see Section 8) [Barbau and Bock, 2017].

7.6.2. Example Simulation Generation and Execution

This section shows portions of Simulink, Simscape, and Modelica simulation files generated by the software in Section 7.6.1 for the example in Section 6 and scenario in Section 7.5. Listing 1 and Listing 2 show portions relating to Figure 11 in Section 6.1, under the scenario in Section 7.5. Simulation languages have a textual syntax for components, such as cars, playing roles, such as controlledVehicle, in the total system. Modelica shows the component name before the role name on each line, while Simscape shows it after, with an equals sign in between. Simscape requires component types to be referenced through a library (appearing as `CCTSLib` in the listings), with instances of those types created for simulation as needed. Modelica combines links between components and equations in the same section, while Simscape separates them, but has the same syntax except for using a double equals sign in equations. Modelica supports redefinition with generalization (redeclaration with extension), but Simscape does not. The translation achieves the same effect as redefinition in Simscape by moving all features of the general total system (except the redefined one) to its scenario and omitting the general system (see Section 7.1).

The car, wheel, and momentum transformation in Listing 1 and Listing 2 are defined in Listing 3 through Listing 6, generated from portions of Figure 12 through Figure 14 in Section 6.2, under the scenario in Section 7.5. Modelica shows the value of simulation constants at the end of the line in parentheses, while Simscape shows in them in a separate setup section. Modelica defines port properties like any other variable, whereas Simscape uses a separate nodes section. Port types in Modelica are connectors, in Simscape they are domains. Conserved variables in Modelica are indicated with a flow keyword, in Simscape they are marked as balanced. The momentum transformation equations in Listing 5 and Listing 6 are generated from Figure 14, as preprocessed in Figure 20, by replacing parameters used in the constraints with the names of properties (or property paths with simulation properties removed) bound to them in the parametric diagram. For example, the variables trq, f, and r in the constraint {trq = f*r} are replaced by `aMomFlow ComponentAMom.trq, aL MomTransComponentLMom.f`, and `radius`, respectively.[26]

The cruise controller, person, and engine used in Listing 1 and Listing 2 are defined in Listing 7 and Listing 8, respectively, generated from portions of Figure 15 through Figure 17 in Section 6.3, under the scenario in Section 7.5. Modelica indicates inputs and outputs individually, while Simscape has separate sections for them. Both refer to constants as parameters. The cruise controller and person are purely signal flow (unidirectional) components, which mean they can be translated to Simulink, as shown in Listing 9 and Listing 10. However, Simscape does not currently support Simulink components as parts of Simscape models. Simulink components must be used outside Simscape models and linked into them with specialized connectors at all levels of the Simscape model down to where the Simulink component is used. This significantly complicates Simscape models for engineers and is difficult to automatically produce. For SysML models that use only signal flow, the preprocessing and translation described in this article produces Modelica and Simulink files

```
model CruiseControlTotalSystem
  replaceable Car controlledVehicle;
  Earth operatingEnvironment;
  LMomPotEngTransComponent gravVehicleLink;
  FluidEffect airVehicleLink;
  ALMomTransComponent aLMTC;
equation
  connect(operatingEnvironment.lMomentumGroundLMom,aLMTC.lMCG);
  connect(gravVehicleLink.lMomPotEngTransComponentLMom,
          controlledVehicle.carLMom);
  connect(operatingEnvironment.airLMom,airVehicleLink.air);
  connect(airVehicleLink.car,controlledVehicle.carLMom);
  connect(controlledVehicle.aMomFlowComponentAMom,
          aLMTC.aMomFlowComponentAMom);
  connect(controlledVehicle.carLMom,aLMTC.aLMomTransComponentLMom);
  aLMTC.radius=controlledVehicle.impeller.radius;
end CruiseControlTotalSystem;

model CruiseControlTotalSystemScenario1
  extends CruiseControlTotalSystem(redeclare Car1 controlledVehicle);
end CruiseControlTotalSystemScenario1;
```

Listing 1. Modelica translation of cruise control total system

```
component CruiseControlTotalSystemScenario1
  components
    controlledVehicle=CCTSLib.Car1;
    operatingEnvironment=CCTSLib.Earth;
    gravVehicleLink=CCTSLib.LMomPotEngTransComponent;
    airVehicleLink=CCTSLib.FluidEffect;
    aLMTC=CCTSLib.ALMomTransComponent;
  end
  connections
    connect(operatingEnvironment.lMomentumGroundLMom,aLMTC.lMCG);
    connect(gravVehicleLink.lMomPotEngTransComponentLMom,
            controlledVehicle.carLMom);
    connect(operatingEnvironment.airLMom,airVehicleLink.air);
    connect(airVehicleLink.car,controlledVehicle.carLMom);
    connect(controlledVehicle.aMomFlowComponentAMom,
            aLMTC.aMomFlowComponentAMom);
    connect(controlledVehicle.carLMom,
            aLMTC.aLMomTransComponentLMom);
  end
  equations
    aLMTC.radius==controlledVehicle.impeller.radius;
  end
end
```

Listing 2. Simscape translation for cruise control total system

```
model Car
  replaceable Person driver;
  replaceable CruiseController speedController;
  Engine powerSource;
  replaceable Wheel impeller;
  LMomFlowElement carLMom;
  AMomFlowElement aMomFlowComponentAMom;
equation
  connect(driver.personSSignal,speedController.speedDriverJack);
  connect(impeller.wheelSSignal,speedController.speedSensorJack);
  connect(speedController.throttleActuatorJack,powerSource.engineThSignal);
  connect(powerSource.crankshaft,impeller.hub);
  connect(impeller.aMomFlowComponentAMom,aMomFlowComponentAMom);
end Car;

model Car1
  extends Car(redeclare Person1 driver,
              redeclare CruiseController1 speedController,
              redeclare Wheel1 impeller,
end Car1;

connector LMomFlowElement
  flow Force f;
  Velocity lV;
end LMomFlowElement;

connector AMomFlowElement
  flow Torque trq;
  AngularVelocity aV;
end AMomFlowElement;
```

Listing 3. Modelica translation for car

```
component Car1
  components
    driver=CCTSLib.Person1;
    speedController=CCTSLib.CruiseController1;
    impeller=CCTSLib.Wheel1;
    powerSource=CCTSLib.Engine;
  end
  nodes
    carLMom=CCTSLib.LMomFlowElement;
    aMomFlowComponentAMom=CCTSLib.AMomFlowElement;
  connections
    connect(driver.personSSignal,speedController.speedDriverJack);
    connect(impeller.wheelSSignal,speedController.speedSensorJack);
    connect(speedController.throttleActuatorJack,
            powerSource.engineThSignal);
    connect(powerSource.crankshaft,impeller.hub);
    connect(impeller.aMomFlowComponentAMom,aMomFlowComponentAMom);
  end
end

domain LMomFlowElement
  variables(Balancing=true)
    f;
  end
  variables
    lV;
  end
end

domain AMomFlowElement
  variables(Balancing=true)
    trq;
  end
  variables
    aV;
  end
end
```

Listing 4. Simscape translation for car

```
model Wheel
  extends AMomFlowComponent;
  AMomFlowElement hub;
  output LinearVelocity wheelSSignal;
equation
  wheelSSignal=-hub.aV*radius;
  hub.aV=-aMomFlowComponentAMom.aV;
  hub.trq+aMomFlowComponentAMom.trq=0;
end Wheel;

model Wheel1
  extends Wheel(redeclare parameter Length radius(start=0.5,fixed=true));
end Wheel1;

model AMomFlowComponent
  AMomFlowElement aMomFlowComponentAMom;
  replaceable Length radius;
end AMomFlowComponent;

model ALMomTransComponent
  extends AMomFlowComponent;
  LMomFlowElement lMCG;
  LMomFlowElement aLMomTransComponentLMom;
equation
  aMomFlowComponentAMom.aV=(aLMomTransComponentLMom.lV-lMCG.lV)/radius;
  aMomFlowComponentAMom.trq=aLMomTransComponentLMom.f*radius;
  lMCG.f=0;
end ALMomTransComponent;

model LMomentumGround
  LMomFlowElement lMomentumGroundLMom;
equation
  lMomentumGroundLMom.lV=0;
end LMomentumGround;
```

Listing 5. Modelica translation for wheel and momentum transformation

```
component Wheel1
  parameters
    radius={0.5,'m'};
  end
  nodes
    aMomFlowComponentAMom=CCTSLib.AMomFlowElement;
    hub=CCTSLib.AMomFlowElement;
  end
  outputs
    wheelSSignal;
  end
  equations
    wheelSSignal==-hub.aV*radius;
    hub.aV==-aMomFlowComponentAMom.aV;
    hub.trq+aMomFlowComponentAMom.trq==0;
  end
end

component AMomFlowComponent
  nodes
    aMomFlowComponentAMom=CCTSLib.AMomFlowElement;
  end
  variables
    radius;
  end
end

component ALMomTransComponent < CCTSLib.AMomFlowComponent
  nodes
    lMCG=CCTSLib.LMomFlowElement;
    aLMomTransComponentLMom=CCTSLib.LMomFlowElement;
  end
  equations
    aMomFlowComponentAMom.aV==(aLMomTransComponentLMom.lV-lMCG.lV)/radius;
    aMomFlowComponentAMom.trq==aLMomTransComponentLMom.f*radius;
    lMCGf==0;
  end
end

component LMomentumGround
  nodes
    lMomentumGroundLMom=CCTSLib.LMomFlowElement;
  end
  equations
    lMomentumGroundLMom.lV==0;
  end
end
```

Listing 6. Simscape translation for wheel and momentum transformation

```
model CruiseController
  output Torque throttleActuatorJack;
  input LinearVelocity speedDriverJack;
  input LinearVelocity speedSensorJack;
  replaceable parameter ICoefficient kI;
  replaceable parameter PCoefficient kP;
  Acceleration accCmd;
  Length errorInteg;
  replaceable parameter ThrottleAccelerationRatio
                           throttleAccRatio(start=1.0,fixed=true);
equation
  der(errorInteg)=speedDriverJack-speedSensorJack;
  accCmd=kP*(speedDriverJack-speedSensorJack)+kI*errorInteg;
  throttleActuatorJack=accCmd*throttleAccRatio;
end CruiseController;

model CruiseController1
  extends CruiseController
    (redeclare ICoefficient kI(start=30.0,fixed=true),
     redeclare PCoefficient kP(start=200.0,fixed=true),
     redeclare ThrottleAccelerationRatio
                 throttleAccRatio(start=1.0,fixed=true));
end CruiseController1;

model Person
  output LinearVelocity personSSignal;
end Person;

model Person1
  extends Person;
equation
  personSSignal=f(t);
end Person;

model Engine
  AMomFlowElement crankshaft;
  input Torque engineThSignal;
equation
  crankshaft.trq=engineThSignal;
end Engine;
```

Listing 7. Modelica translation for cruise controller, person, and engine

```
component CruiseController1
  parameters
    kI={30.0};
    kP={200.0};
    throttleAccRatio={1.0};
  end
  outputs
    throttleActuatorJack={0};
  end
  inputs
    speedDriverJack;
    speedSensorJack;
  end
  variables
    accCmd;
    errorInteg;
  end
  equations
    der(errorInteg)==speedDriverJack-speedSensorJack;
    accCmd==kP*(speedDriverJack-speedSensorJack)+kI*errorInteg;
    throttleActuatorJack==accCmd*throttleAccRatio;
  end
end

component Person
  outputs
    personSSignal;
  end
end

component Person1 < CCTSLib.Person
  equations
    personSSignal=f(t);
  end
end

component Engine
  nodes
    crankshaft=CCTSLib.AMomFlowElement;
  end
  inputs
    engineThSignal;
  end
  equations
    crankshaft.trq==engineThSignal;
  end
end
```

Listing 8. Simscape translation for cruise controller, person, and engine

```
<Block BlockType="SubSystem" Name="CruiseController1" SID="1">
  <P Name="Ports">[2,1]</P>
  <System>
    <Block BlockType="Outport" Name="throttleActuatorJack" SID="5">
      <P Name="Port">1</P> </Block>
    <Block BlockType="Inport" Name="speedDriverJack" SID="6">
      <P Name="Port">1</P> </Block>
    <Block BlockType="Inport" Name="speedSensorJack" SID="7">
      <P Name="Port">2</P> </Block>
    <Block BlockType="M-S-Function" Name="cc" SID="8">
      <P Name="FunctionName">
         CruiseController1_cc_CruiseControllerConstraint</P>
      <P Name="Ports">[2,1]</P> </Block>
    <Line>
      <P Name="Src">6#out:1</P>
      <P Name="Dst">8#in:1</P> </Line>
    <Line>
      <P Name="Src">7#out:1</P>
      <P Name="Dst">8#in:2</P> </Line>
    <Line>
      <P Name="Src">8#out:1</P>
      <P Name="Dst">5#in:1</P> </Line>
  </System>
</Block>
<Block BlockType="SubSystem" Name="Person1" SID="2">
  <P Name="Ports">[0,1]</P>
  <System>
    <Block BlockType="M-S-Function" Name="pc" SID="3">
      <P Name="FunctionName">Person1_pc_Person1Constraint</P>
      <P Name="Ports">[0,1]</P> </Block>
    <Block BlockType="Outport" Name="personSSignal" SID="4">
      <P Name="Port">1</P> </Block>
    <Line>
      <P Name="Src">3#out:1</P>
      <P Name="Dst">4#in:1</P> </Line>
  </System>
</Block>
```

Listing 9. Simulink translation for cruise controller and person

```
function CruiseControllerConstraint(block)
  setup(block);
end
function setup(block)
  block.NumInputPorts =2;
  block.NumOutputPorts =1;
  block.OutputPort(1).SamplingMode = 'sample';
  block.NumContStates = 2;
  block.RegBlockMethod('Derivatives',@Derivative);
  block.RegBlockMethod('Outputs',@Outputs);
end
function Derivative(block)
  block.Derivatives.Data(2)=
    block.InputPort(1).Data-block.InputPort(2).Data;
  block.ContStates.Data(1)=
    200*(block.InputPort(1).Data-block.InputPort(1).Data)+
      30*block.ContStates.Data(2);
end
function Outputs(block)
  block.OutputPort(1).Data=block.ContStates.Data(1)*1.0;
end
```

Listing 10. Simulink translation for cruise controller constraint

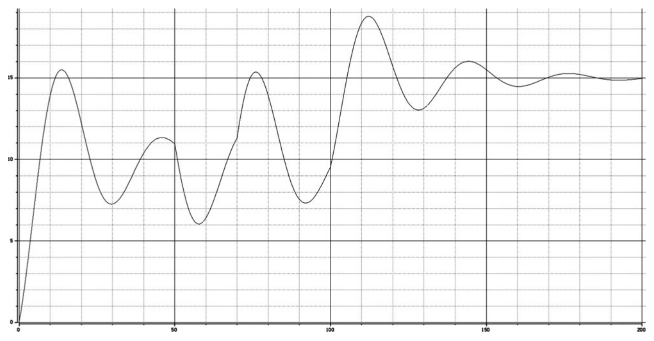


Figure 25. Execution results from example simulation files.

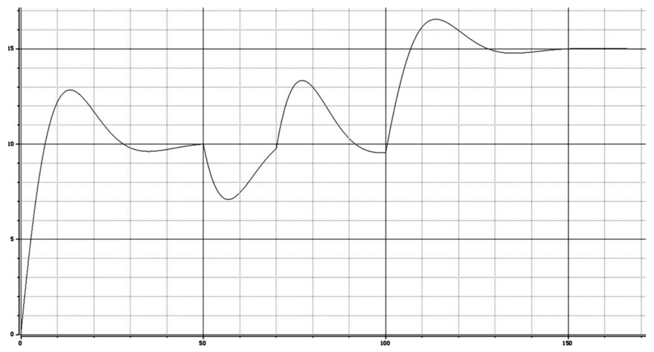


Figure 26. Execution results with different values for cruise controller constants.

that execute the same way (examples of this are publicly available [Barbau and Bock, 2017]).

Results of executing the automatically produced simulation files are the same for Modelica and Simscape, illustrated by the speed of the car over time in Figures 25 and 26. The driver sets the speed at the beginning, then again half way through (the scenario's constraint on the driver's desired speed in Fig. 23 is defined as a conditional over time). The road is flat except in the second section from the left, where the car is going up a hill. The controller constants `kI` and `kP` have values from the scenario in Figure 23 that cause too much oscillation in Figure 25. They are changed to reach the desired speed more quickly and with less variation in Figure 26.

The software in Section 7.6.1 (implementing preprocessing and translations in Sections 7.1 through 7.4) can also translate the generated simulation files above back to extended SysML models, and then generate simulation files that execute the same way as above. This and other examples are publicly available, with generated simulation files [Barbau and Bock, 2017].

8. SUMMARY AND FUTURE WORK

This article presents an extension of SysML enabling integration with physical interaction and signal flow simulation platforms based on ordinary and algebraic differential equation solvers (also known as lumped parameter, one-dimensional, or network simulation), see Section 5. The extension is developed from a comparison of SysML and simulation modeling, see Sections 3 and 4. Simulation modeling and SysML overlap significantly, with both describing systems, subsystems, components, parts, as well as links and directional and bidirectional flows between them, plus equalities to define the behavior of components. SysML has additional capabilities to specify the kinds of things flowing between components, while simulation languages have additional capabilities to distinguish properties as constant or variable, continuous or discrete, and obeying two kinds of physical law (for flow rate and potential to flow, respectively). The extension adds these simulation modeling capabilities to SysML, with semantics equivalent to the corresponding simulation concepts. This enables generation of simulation text files that execute the same way as those built natively in simulation tools, but without rewriting simulation files for each tool separately, as demonstrated in Sections 6 and 7. This increases engineering efficiency by enabling simulation tools to be used more easily in systems engineering processes.

The SysML extension could be improved with:

- Platform-independent modeling of solver directives. Sometimes equivalent simulation models produced according to Section 7 from the same extended SysML model will execute differently due to characteristics of solvers in simulation tools. Some of these characteristics are under modeler control, such as time step size or the increment for derivative library components, and can be set to ensure equivalent simulation models execute the same way. It would be useful for the SysML extension to include a model for solver directives that could be translated to simulation platforms.
- More abbreviated signal flow modeling. Unidirectional flow properties could be typed by numbers or Booleans to indicate signal flow without using simulation properties. The SimVariable stereotype could be applied to these flow properties to specify discretely changing signals.
- Functions and operators in the equation language that are available in MATLAB or Modelica, but not both. Missing functions on the platforms could be defined equivalently to those on the other, and used in translation.
- Component behavior library elements in the SysML extension that include equations for components available in Simulink/Simscape or Modelica, but not both. Translation could reuse the components on the platform supporting them and translate to the other platform as normal.
- Physical interaction library elements for other conserved substances, such as magnetism (characterized by magnetic flux and magnetomotive force), as well as for specifying fluid and gas interactions with two pairs of flow rate and potential variables. Differences in fluid and gas units between Simscape and Modelica could be overridden by introducing elements for this in the SysML extension library (using simulation blocks with two pairs of simulation variables or supporting multiple values for simulation properties) and translating to the platforms as

normal, or by generating unit conversions during translation to the platform libraries.

- Unit models. These are beyond the scope of this article, but the implementation in Section 7.6.1 includes a unit model to validate the SysML extension. The conserved substances in Figure 7, Section 5.2, correspond to quantity kinds for units in international standards [International Organization for Standardization 2016] [International Organization for Standardization 2012], which SysML includes in a nonnormative model library. Ideally, this library would become normative, and the SysML extension for simulation could specialize it to provide translation to simulation platforms, including Simulink's recent support for signal units. This would provide a full set of units and kinds of physical quantities.

Preprocessing for additional SysML capabilities could be added for:

- Other kinds of ports:[27]
  - Ports typed by blocks with behaviors, constraints, and other constructs not modeled with properties. Preprocessing could detect these and carry out the same transformations it currently does for port types with nonflow properties.
  - Behavior ports. These stand in for the objects that have the port. When they are typed by blocks that have nonflow/simulation properties, preprocessing could use those properties on the objects having the ports, rather than moving them to an internal part.
- Properties used in constraints not indicated as simulation variables. Preprocessing could find these by analyzing constraints, then apply the stereotype automatically.
- Constraints defined without constraint blocks. Processing could create constraint blocks and bindings for constraints that are not reused.

Translation could be improved by:

- Including diagram information. SysML graphical information (position of nodes and routing of lines on diagrams) corresponds to at least some graphical information in simulation files. These could be translated to each other to preserve the layout of diagrams between SysML and simulation platforms.
- Using component physical interaction elements in simulation libraries when they match elements in the corresponding SysML extension library. Translation could also use simulation library elements that do not match by translating SysML constraints to reuse these platform elements.
- The current implementation translates simulation files generated from SysML models back into SysML, but cannot recover the original names of flow and simulation properties or the original conserved substances. These could be recorded in generated simulation files as comments. In addition, SysML models requiring preprocessing will have a different structure after translation to simulation and back. Preprocessing operations could also be recorded in simulation files and reversed when translating back to SysML. Some additional capabilities in simulation files might be useful to translate to SysML, for example, those with MATLAB S-functions that introduce intermediate variables, or that define Modelica functions and operators for additional data types.
- Documenting the implementation's support for state machines. Simulink integrates with StateFlow®, a modeling and simulation tool for state machines [The MathWorks, Inc 2016d]. Modelica includes a library for state machines [Modelica Association 2016]. The implementation in Section 7.6.1 translates between basic SysML state machines and these simulation platforms. Additional preprocessing would be needed to handle all the expressiveness of SysML state machines.

Enhancements to SysML could provide more opportunities to apply this extension. For example, additional support for variant modeling in SysML would simplify development of alternative designs. Simulation-based exploration and evaluation of these alternatives would be faster using integrations enabled by this extension.

ACKNOWLEDGMENTS

The authors thank members of the Object Management Group SysML-Modelica Revision Task Force led by Axel Reichwein for helpful discussions on the topics of this paper, as well as Anantha Narayanan for his comments. Commercial equipment and materials might be identified to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the U.S. National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose. This material is based in part on work supported by U.S. National Institute of Standards and Technology contract SB1341-13-SU-1055 and grant award 70NANB14H249 to Engisis, LLC, and in part on work supported by the NIST-ARRA Measurement Science and Engineering Fellowship Program award 70NANB10H026, through the University of Maryland, and in part on work supported by NIST grant award to the University of Maryland.

1. Many concepts and notations in SysML are shared with UML, but for brevity this article will describe them all as SysML.
2. The extension is applied to translation between extended SysML and simulation platforms for validation only. Other aspects of interaction between SysML and simulation tools, such as model synchronization, are beyond the scope of the article [Johnson et al., 2012; Reichwein et al., 2012].
3. SMP documentation recognizes the importance of integrating systems engineering and simulation information, but puts it explicitly out of scope, see [European Cooperation for Space Standardization, 2011], Volume 1 (Principles and requirements), first paragraph under Figure 4 (SMP high level overview).

4. Simulation tools sometimes use the term "constant" for properties that have the same value across all simulation executions, such as the physical constant for gravitational acceleration, rather than just during each simulation execution separately.
5. In mathematical terms, a variable $x(t)$ is continuous if for every time instant t_0, the limit of $x(t)$ as t approaches t_0 exists and is equal to $x(t_0)$.
6. In mathematical terms, the time evolution of a discrete variable is piecewise constant. During their constant periods, discrete variables have the same values at nearby times in both the future and past, like continuous variables might, but at the beginning or end of constant periods, discrete variable values are only the same at nearby times in the future or past, respectively, but not both.
7. Potential to flow is also equal to the energy or work applied to physical substances divided by the amount of substance acted on, but simulators do not use this relationship because they are not concerned with the particular substances involved.
8. Flow rates and potential differences are one-dimensional vectors (scalars), enabling them to be represented by real numbers, with sign indicating direction. Direction for flow rates is in or out of components, while for potentials it is between one port or internal potential of a component to another (single potentials can be considered differences from zero).
9. Physical interactions can be unidirectional in the sense that flowing substances might happen to pass only in one direction across links, as in diodes and backflow preventers, but these are still physical interactions, rather than signal flow. The direction of physical flow is restricted by additional component equations (see Section 3.2.3), rather than by ports. This is useful when modeling the physical basis of signal flow.
10. This is because potential to flow, such as water pressure or voltage, passes through physical substances in waves, which are only temporary displacements, resulting in zero flow rates.
11. This differs from nonnumeric information flow, such as in software, where signals can be combined in any manner.
12. These *constitutive* equations are known for many engineering domains, such as Ohm's law for electric resistors, Poiseuille's law for laminar flow in pipes, and Fourier's law for heat conduction. They determine changes in potential across a component based on the flow rate through it and its material characteristics.
13. Splitting and merging physical substances and signals generalize this by having more than two ports. For example, pipes can have more than two ends, for dividing or combining fluids. Similarly, signal processors can add varying numeric values into one, or split one into multiple based on frequency ranges.
14. SysML properties can be *read-only*, which means they cannot be modified after objects are created, but simulation constants can be modified between simulation runs.
15. Simulation properties typed by simulation blocks can be used for signal flow, but they would typically only have one (nonconserved) variable. Simulation properties with numeric or Boolean values are shorthands for simulation blocks with a single nonconserved variable. See the Signal Flow subsection of 3.2.2 about modeling signals with their underlying physical basis.
16. Flow can only occur when flow properties have compatible directions and kinds of things flowing (property types). For example, flow properties providing oil cannot be connected to flow properties providing oil (direction mismatch) or to flow properties accepting water (type mismatch), but can be connected to flow properties for accepting liquids.
17. SysML is less restrictive, allowing inout flow properties to be connected to in and out flow properties.
18. Out and inout flow properties are not restricted in the number of other flow properties they can connect to.
19. The signal flow library enables SysML models to avoid conjugated ports, which have a semantics that reverses the direction of flow properties on blocks typing them. Conjugation simplifies modeling a bit by using the same block for both directions, but complicates implementations that analyze and simulate SysML models because the block typing conjugated ports is not in the model. The signal flow library can be used with conjugated ports by using elements for only one direction.
20. A standard constraint language that integrates with SysML is available [Object Management Group, 2014], but is not as suitable for specifying equations for simulation as those on simulation platforms.
21. Earth would normally be an instance of a block for planets, but is treated in Figure 11 as a block for a one-of-a-kind thing. The model could be generalized to planets, to support more kinds of vehicles.
22. The transformation between linear momentum and potential energy is not modeled as connector between the car and earth's gravitational field to highlight that momentum converted to potential energy can only be transferred back to the car, as compared to momentum transferred to the air, which can be transferred to other objects.
23. The radius is specified as a variable, to support applications such as cams, even though it is constant in this example (see Section 7.5 about setting its value).
24. The constraint property cannot be on the association block directly (without aLMTC), because the connector implies additional equations that would conflict with them (see the Physical Interaction subsection of 3.2.2). This requires nonbinding connectors in parametric diagrams (for momentum flows in this example), but parametrics are specialized internal block diagrams, which support general connectors (see Section 4).
25. An alternative for SysML blocks with redefinitions is to account for multiple generalizations as in the previous paragraph, then Simscape models would correspond to SysML models that omit redefined properties from the one remaining general block (and constructs referring to them, such as connectors), leaving the specialized block with redefining properties and constructs moved from the general block that referred to redefined properties.
26. Simscape requires conserved (balancing) variables reached through ports (node variables) to be replaced

in equations with a single variable specified in branch sections. For example, `simAMFCAMom.trq` and `simAMFCLMom.f` in the equations of Listing 5 would be replaced by `simAMFCAMomtrq` and `simAMFCLMomf`, respectively, which would be defined as equivalent to the original navigations (dot notations) in a separate branch section. The original navigations without branches are used in the listings for brevity.

27. The preprocessing in Section 7.2 also does not support proxy ports that have multiple bindings to internal parts, in part because SysML does not currently provide a usable semantics for these. Other uses of binding connectors, such as between internal part properties, cannot be translated to simulation platforms because each object during simulation is the value of at most one simulation part property.

REFERENCES

R. Barbau and C. Bock, Implementation of an extension of the systems modeling language for physical interaction and signal flow simulation, https://github.com/usnistgov/saismo/releases/download/sejournal/syspisf.zip, 2017.

K. Berkenkotter, S. Bisanz, U. Hannemann, and J. Peleska, The HybridUML profile for UML 2.0, Intl J Software Tools Technol Transfer 8(2) (2006), 167–176.

C. Bock and J. Odell, Ontological behavior modeling, J Object Technol 10(3) (2011), 1–36.

C. Bock, SysML and UML 2 support for activity modeling, Syst Eng 9(2) (March 2006), 160–185.

C. Bock, UML 2 composition model, J Object Technol 3(10) (November 2004), 47–73.

C. Bock, UML without pictures, IEEE Software Special Issue on Model-Driven Dev 20(5) (September/October 2003), 33–35.

Y. Cao, Y. Liu, H. Fan, and B. Fan, SysML-based uniform behavior modeling and automated mapping of design and simulation model for complex mechatronics, Computer-Aided Design 45(3) (March 2013), 764–776.

F. Cellier, H. Elmqvist, and M. Otter, "Modeling from Physical Principles," in W. Levine (Editor), Control System Fundamentals, CRC Press, Abingdon, UK, 1999, pp. 99–108.

Controllab Products, Getting Started with Sim-20 4.6, http://www.20sim.com/downloads/files/20simGettingStarted46.pdf, 2015.

M. Dadfarnia, C. Bock, and R. Barbau, An improved method of physical interaction and signal flow modeling for systems engineering, Conference on Systems Engineering Research, 2016.

R. Dorf and R. Bishop, Modern control systems, 13th edition, Prentice Hall, Upper Saddle River, New Jersey, January 2016.

D. Dori, A. Renick, and N. Wengrowicz, When quantitative meets qualitative: Enhancing OPM conceptual systems modeling with MATLAB computational capabilities, Res Eng Design 27(2) (April 2016), 141–164.

European Cooperation for Space Standardization, Simulation modeling platform, ECSS-E-TM-40-07, http://www.ecss.nl/wp-content/uploads/standards/ecss-e/ECSS-E-TM-40-07-Volume1A25January2011.pdf, January 2011.

S. Friedenthal, A. Moore, and R. Steiner, A practical guide to SysML, 3rd edition, Morgan Kaufmann, Burlington, Massachusetts, November 2014.

P. Fritzon, Introduction to modeling and simulation of technical and physical systems with Modelica, Wiley-IEEE Press, Hoboken, New Jersey, September 2011.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: Elements of reusable object-oriented software, Addison-Wesley, Boston, Massachusetts, May 1998.

J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, The Java language specification, 8th edition, Addison-Wesley, Boston, Massachusetts, May 2014.

J. Holt and S. Perry, SysML for systems engineering, 2nd edition, The Institution of Engineering and Technology, Stevenage, UK, 2013.

IEEE Standards Association, 1516-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), http://standards.ieee.org/findstds/standard/1516-2010.html, August 2010.

International Organization for Standardization, ISO 80000, Quantities and units, https://www.iso.org/committee/46202/x/catalogue, 2016.

International Organization for Standardization, Joint Committee on Guides for Metrology, International vocabulary of metrology—Basic and general concepts and associated terms (VIM), 3rd edition, http://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2012.pdf, 2012.

A. Iserles, A first course in the numerical analysis of differential equations, 2nd edition, Cambridge University Press, Cambridge, UK, December 2008.

T. Johnson, A. Kerzhner, C. Paredis, and R. Burkhart, Integrating models and simulations of continuous dynamics into SysML, J Comput Information Sci Eng 12(1) (March 2012), 011002-1–011002-11.

R. Kawahara, R. Dotan, T. Sakairi, K. Ono, H. Nakamura, A. Kirshin, S. Hirose, and H. Ishikawa, Verification of embedded system's specification using collaborative simulation of SysML and simulink models, Proceedings of the International Conference on Model-Based Systems Engineering, March 2009.

X. Liu and Y. Cao, Design of VA V flight control system virtual prototype using Rhapsody and Simulink, Proceedings of the International Conference On Computer Design And Applications, 2010.

I. Matei and C. Bock, An analysis of solver-based simulation tools, National Institute of Standards and Technology Interagency Report 7846, March 2012a.

I. Matei and C. Bock, Modeling methodologies and simulation for dynamical systems, National Institute of Standards and Technology Interagency Report 7875, August 2012b.

Modelica Association, Functional Mock-up Interface for Model Exchange and Co-Simulation, http://www.fmi-standard.org/downloads#version2, July 2014a.

Modelica Association, Modelica®-A Unified Object-Oriented Language for Systems Modeling, Language Specification, version 3.3, revision 1, https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf, July 2014b.

Modelica Association, StateGraph, https://github.com/modelica/Modelica/blob/v3.2.2/Modelica/StateGraph.mo, 2016.

NoMagic, MagicDraw user manual, http://www.nomagic.com/files/manuals/MagicDraw%20UserManual.pdf, 2015.

C. Nytsch-Geusen, The use of the UML within the modelling process of Modelica-models, Proceedings of the 1st International

Workshop on Equation-Based Object-Oriented Languages and Tools, July 2007.

Object Management Group, Machine readable file for SysML 1.5, http://www.omg.org/spec/SysML/20161101/SysML.xmi, May 2017b.

Object Management Group, OMG Systems Modeling Language™, version 1.5, http://www.omg.org/spec/SysML/1.5, May 2017a.

Object Management Group, OMG Unified Modeling Language™, version 2.5, http://www.omg.org/spec/UML/2.5, March 2015a.

Object Management Group, Object constraint language, http://www.omg.org/spec/OCL, February 2014.

Object Management Group, SysML-Modelica transformation specification, http://www.omg.org/spec/SyM/1.0, November 2012.

Object Management Group, XML Metadata Interchange, version 2.5.1, http://www.omg.org/spec/XMI/2.5.1, June 2015b.

Open Source Modelica Consortium, OpenModelica User's Guide, https://openmodelica.org/doc/OpenModelicaUsersGuide/OpenModelicaUsersGuide-latest.pdf, January 2016.

C. Paredis, J. Bernard, R. Burkhart, H. de Koning, S. Friedenthal, P. Fritzson, N. Rouquette, and W. Schamai, An overview of the SysML-Modelica transformation specification, Proceedings of the 20th International Council on Systems Engineering International Symposium, July 2010.

H. Paynter, Analysis and design of engineering systems, MIT Press, Cambridge, Massachusetts, June 1960.

M. Rahman and M. Mizukawa, Modeling and design of mechatronics system with SysML, Simscape and Simulink, Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, July 2013.

F. Raven, Automatic control engineering, 5th edition, McGraw-Hill, New York, New York, January 1995.

A. Reichwein, Application-specific UML profiles for multidisciplinary product data integration, PhD thesis, Universität Stuttgart, 2011.

A. Reichwein, P. Witschel, C. Paredis, P. Stelzig, and R. Wasgint, Maintaining consistency between system architecture and dynamic system models with SysML4Modelica, Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems, 6th Workshop on Multi-Paradigm Modeling, October 2012.

W. Schamai, P. Fritzson, C. Paredis, and A. Pop, Towards unified system modeling and simulation with ModelicaML: Modeling of executable behavior using graphical notations, Proceedings of the 7th Modelica Conference, September 2009.

C. Secchi, C. Fantuzzi, and M. Bonfe, On the Use of UML for Modeling Physical Systems, Proceedings of the IEEE International Conference on Robotics and Automation, April 2005.

C. Sjostedt, J. Shi, M. Torngren, D. Servat, D. Chen, V. Ahlsten, and H. Lonn, Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and transformations, Proceedings of the 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems, 2008.

C. Sjostedt, D. Chen, P. Cuenot, P. Frey, R. Johansson, H. Lonn, D. Servat, M. Torngren, Developing dependable automotive embedded systems using the EAST-ADL; representing continuous time systems in SysML, Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools, July 2007.

R. Snyder, D. Bocktaels, and X. Feigenbaum, Functional validation with a practical SysML/Simulink transformation, Proceedings of the NEPTUNE Workshop, 2010.

D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, EMF: Eclipse modeling framework, 2nd edition, Addison-Wesley, Boston, Massachusetts, December 2008.

The MathWorks, Inc., MATLAB® Primer, 2016b.

The MathWorks, Inc., Simscape™ Language Guide, 2016a.

The MathWorks, Inc., Simulink® User's Guide, 2016c.

The MathWorks, Inc., StateFlow® User's Guide, 2016d.

S. Turki, S. Thierry, and A. Sghaier, Mechatronic systems modeling with SysML: A Bond Graph addendum for energy analysis, World Sci Eng Acad Soc Trans Syst 4(5) (May 2005), 617–624.

P. Vasaiely, Interactive Simulation of SysML Models using Modelica, Department of Computer Science, Hamburg University of Applied Sciences, 2009.

P. van den Bosch and A. van den Klauw, Modeling identification and simulation of dynamical system, CRC-Press, Abingdon, UK, July 1994.