NIST Interagency Report
NIST IR 8490

Physical Component Libraries for SysPhS Modeling and Simulation in Manufacturing

Charles A. Manion
Conrad Bock
Raphael Barbau

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8490

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Interagency Report
NIST IR 8490

Physical Component Libraries for SysPhS Modeling and Simulation in Manufacturing

Charles A. Manion
Conrad Bock
Smart Connected Systems Division
Communications Technology Laboratory
Raphael Barbau
Associate, Smart Connected Systems Division
Communications Technology Laboratory
University of Maryland

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8490

October 2023

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Publication History
Approved by the NIST Editorial Review Board on 2023-09-18

How to cite this NIST Technical Series Publication:
Charles A. Manion, Conrad Bock, Raphael Barbau (2023) Physical Component Libraries for SysPhS Modeling and Simulation in Manufacturing. (National Institute of Standards and Technology, Gaithersburg, MD), NIST IR 8490. https://doi.org/10.6028/NIST.IR.8490

NIST Author ORCID iDs
Charles A. Manion: 0009-0008-5273-1995
Conrad Bock: 0009-0009-3172-120X
Raphael Barbau: 0000-0002-0331-2929

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 languages and information models that capture requirements, designs, and tests (such as the Systems Modeling Language, SysML®). The SysML Extension for Physical Interaction and Signal Flow Simulation (SysPhS) is a standard for augmenting SysML models with one-dimensional (lumped parameter) simulation information and translating them to widely-used simulation platforms for testing, without respecifying the system on those platforms. It includes standard reusable models of system components (libraries) corresponding and translated to those in common between the platforms. This report presents additional physical interaction component libraries, covering areas not currently standardized. It applies the libraries to some manufacturing examples, translates them to the simulation platforms, and verifies that they give the same results across platforms.

Key words

SysML; analysis integration; 1D simulation; lumped parameter.

1. Introduction

Systems engineers (SEs) help coordinate the work of multiple other engineering disciplines (mechanical, material, electrical, software, and so on), requiring information to flow between SEs and those in other disciplines, particularly between the engineering tools they use. SEs often specify overall system requirements, structure, behavior, and tests in the Systems Modeling Language (SysML®) [1]. Each discipline has its own languages and tools for the aspects of a system concerning them, often significantly overlapping systems models and other discipline models. This leads to inconsistencies between systems and discipline models, as well as between models from different disciplines, usually discovered later as engineers interact, requiring significant additional time to resolve and rework.

One approach to addressing these problems is extend SysML with the additional information needed for each discipline and define translations between the extended systems models and discipline models. This enables SEs and discipline engineers to build on a single system model for information used by all the other engineers, ensuring their results can be reliably used by others. In particular, the results of model analysis, such as simulation and optimization, can be efficiently communicated to other engineers, checked against requirements and tests, potentially leading to changes in overall system model for everyone.

Many engineering disciplines build simulation models consisting of interconnected components (structure), with behavior specified by ordinary and algebraic differential equations (derivatives of functions of one variable, typically time). This kind of simulation is applicable to a wide range of physical interactions between components (such as mechanical, electrical, and so on) as well as communication of numeric signals [2][3][4]. This report refers to these kind of simulation models as physical interaction and signal flow (also known as lumped parameter, one-dimensional, or network models).

The SysML Extension for Physical Interaction and Signal Flow Simulation (SysPhS) [5] extends SysML to cover the information needed for this kind of simulation and gives translations to widely-used tools [6][4][2]. System structure, behavior, and simulation information can be specified once in SysML/SysPhS, then translated to simulation platforms, rather than manually recoded for each one. This enables the results of simulations to be compared against requirements and tests in SysML models to predict how well a system design will perform when built and operated. It also enables discipline engineers to use different simulation languages and tools.

SysPhS includes standard models of system components (libraries) corresponding to the elements of existing simulation platforms libraries that are largely similar between tools and languages. Models using these SysPhS libraries are translated to use the corresponding elements in platform libraries. This has the advantage of producing smaller simulation files, due to reuse of platform libraries, but severely limits application of the standard because most platform libraries differ between tools and languages, even when they cover useful areas of physical interaction. This report presents additional physical interaction

component libraries in some areas where existing simulation libraries differ too much to be reused in translating SysPhS models: rotational and translational mechanics, as well as heat transfer. They are defined in standard SysPhS/SysML and translated to simulation tools and languages using an open implementation of SysPhS [7][8]. This report applies the additional libraries to manufacturing examples, translates them to two widely used simulation platforms, and verifies that they give the same results. Section 2.1 reviews physical interaction and signal flow modeling (also known as lumped parameter, one-dimensional, or network models) independently of any particular simulation language or tool, as well how they are modeled in SysML and SysPhS in particular. Section 3 describes a model library for real signal flow developed for the paper. Section 4 presents libraries for translational mechanics, rotational mechanics, and entropy (heat) transfer, not included in SysPhS currently. Section 5 applies the libraries to manufacturing examples, translates them to two widely used simulation platforms, and presents the results. Section 6 shows the results are the same on the two platforms. Section 7 summarizes the paper and outlines future work.

2. Physical Interaction and Signal Flow Modeling with SysPhS

Section 2.1 reviews the two ways system components interact in one-dimensional (lumped parameter) modeling (physical interaction and signal flow), independently of any particular language or simulation tool. Section 2.2 outlines the capabilities of SysML needed in SysPhS, while Section 2.3 covers SysPhS itself.

2.1. Physical Interaction and Signal Flow Modeling

Physical interaction and signal flow modeling distinguishes system component interactions based on whether the things being exchanged are physical or informational (numeric and boolean only) [9]. Physical interaction is suited for specifying systems with physical behavior, while signal flow is often applied to control and signal-processing systems. In practice, physical interaction and signal flow are typically combined in the same models. For example, many systems have physical components directed by control systems via sensors and actuators.

Physical and informational things differ in that physical things cannot be:

- "copied" like information can (physical things are *conserved*, information is not).
- moved without affecting the mover ("bidirectional" effects, with effects determined during system operation or simulation), while sending of information does not affect the sender ("unidirectional" effects, from outputs to inputs, as specified in models, and not changed during system operation or simulation).
- carry energy, while information does not.

Physical interaction and signal flow modeling requires components to interact through their *ports*, where physical or information things move into or out of components, along links

between ports. The specifics of physical interaction and signal flow modeling are covered in Sections 2.1.1 and 2.1.2, respectively, while Section 2.1.3 briefly describes a common misapplication of them.

2.1.1. Physical Interaction

Physical interaction models reduce physical things to one of their physical characteristics, which are treated as moving along with those things, as well being conserved or converted to others with them. For example, electrons or positrons can be treated as their electric charge, or as their momentum, or in large ensembles (many linear momenta aggregated statistically) as their entropy. These characteristics are conserved as they move between system components, and are also conserved within components unless they are converted to others.

Conserved physical characteristics (carried by physical things) move into or out of each component at their ports, as described by two numeric variables at each port:

- *Flow rate*: The amount of substance (as a conserved characteristic) per time moving through a port, 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 between ports on the same component, such as voltage for electric charge, linear and angular velocity for momentum, and temperature for entropy.

Flow (non-zero flow rates) can only happen when potentials on the same component:

- differ between its ports and the component's resistance to flow is not infinite.
- are the same between its ports and the component's resistance to flow is zero.

Flow rate is proportional to potential difference between ports on the same component and inversely proportional to resistance of that component, though flow can happen between ports of the same potential when the component does not resist it.[1] Flow rate and potential variables for each conserved substance multiply to power (energy per time), giving a flow rate for energy moving through a port. Links between ports act as if they have no effect on the substances "moving" across them, including no resistance or transformation. This is reflected in the following mathematical relationships between physical variables (of the same name) on each port:

- The sum of flow rate variables on linked ports is zero (conservation of the flowing characteristic).

[1]The name of this "constitutive" relationship differs by domains, such as Ohm's law for electricity, Poiseuille's law for laminar flow in long pipes with constant cylindrical cross sections, and Fourier's law for heat conduction. These laws relate potential differences across a component with flow rates through them and their resistance to flow due to material characteristics [10].

- Potential variables on the ends of each link are equal (no resistance to flow).

Links cannot affect things flowing across them like physical connections can, these effects must be modeled as additional components.

2.1.2. Signal Flow

Signal flow models limit information to numbers (real or integer) or boolean values (true or false). Because these are not physical things:

- Only one variable per signal is needed at a port, giving a numeric or boolean value for the signal. It is not a flow rate or a potential to flow.
- Signal variables are either output or input at a port, with links only between output and input (flow is unidirectional).

Links between ports require signal variables (of the same name) on each port to have the same value. Multiple signal output variables cannot be linked to the same input, to prevent signal values from conflicting.[2]

2.1.3. Signal Flow of Physical Quantities?

The examples in this paper use physical interaction for modeling physical phenomena, rather than signal flow.[3] Signal flow modelers might be tempted to define (unidirectional) variables representing physical quantities, such as torque and angular velocity, but this results in much more complex models than physical (bidirectional) variables would. Consider an electrical resistor governed by equation $V = I * R$. We could model this as a signal flow block that takes current as input and provides voltage across the resistor as output. This only works when the current through the resistor is known, which is not the case when the resistor is connected to a voltage source or in parallel with other resistors (constant voltage drop across all resistors). These applications require another signal flow block that takes voltage as input and provides current as output. Determining which block to use might not be straightforward in more complicated systems and also might change during a single simulation run, as when switching dynamically between series and parallel connections.

Physical interaction models resolve these problems by maintaining two bidirectional variables on each connection, with the choice of independent variable left to the simulator, including cases where variable dependence changes during a single simulation run. Models formulated this way are more versatile and easier to compose. Signal flow of physical quantities is more suited to modeling components that control physical ones, see Section 3.

[2] Physical potential variables are not limited this way, even though they must have the same values like signal variables do, because bidirectional physical effects enable conflicting potentials to "even out".

[3] An example of applying signal flow modeling to a physical interaction system is in Annex 1.5 of [5].

2.2. SysML

System structure in SysML describes the kinds of components a system is made of (whole-part relationships), and how they are interconnected (part-part relationships). Systems and components are both modeled as *blocks*, enabling components to be entire systems themselves, and systems to be components of other systems. Each block represents potentially many systems or components built or simulated in the way the block specifies (*instances* of the block). Blocks are notated by rectangles, as shown in Figure 1. They can form taxonomies via *generalization*, relating specialized blocks to more general ones, appearing as closed head arrows pointing to more general blocks. Instances of specialized blocks are instances of more general ones, which means specifications in more general blocks apply to all its instances, including those of more specialized blocks in a taxonomy. In Figure 1, the generalizations indicate that all instances of SportsCar and 4WDCar are also instances of Car. Blocks appear in *block definition diagrams* (BDDs), indicated on the top right of the diagram frame.

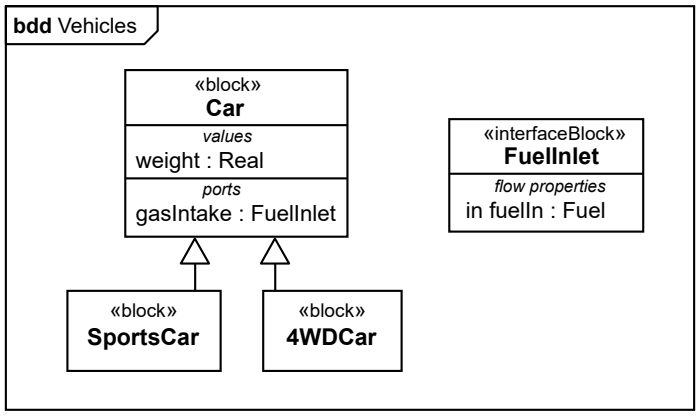

Fig. 1. SysML block definition diagram

Blocks can include *properties* that each instance can give values for. SysML distinguishes several kinds of properties, one kind being those with data values, such as numbers, booleans, or strings. These are notated in block compartments labelled *values*, as shown for the weight property in Car in Figure 1, which gives the weight of each car separately, including sports cars and four wheel drive cars. Property values are instances of the *type* of the property, shown the the right of the colon, such as Real for weight (real numbers).

Ports are properties for specifying some of the interactions of an entire block. These appear in labelled block compartments, as shown for the gasPort in Car in Figure 1. Ports are often typed by *interface blocks*, which are blocks that only mediate between the inside and outside of other blocks, rather than introducing their own behaviors as components do. They often define *flow properties* to specify the kinds of things moving across the boundary of a component, as well the possible directions of flow. The gasPort in Figure 1 is typed by the FuelInlet interface block, which defines the fuelIn flow property, typed by

Fuel to specify the kind of thing flowing through it, with direction restricted to being into components that have this kind of port. Other directions are out and inout (unrestricted).

Blocks can also include connectors between properties (part-part relationships) to specify links between values of properties of the same object (whole-part relationships). Connectors typically link ports and appear in *internal block diagrams* (IBDs). For example, Figure 2 defines a connector in a GasStation block that links ports of its customer and pump properties. Properties in IBDs are notated as rectangles, larger ones for components and smaller ones for ports. Small arrows in port rectangles indicate the directions of their flow properties. Connectors between ports that define flow properties can show a small filled triangle labelled with the types of the flow properties, Fuel in this example.

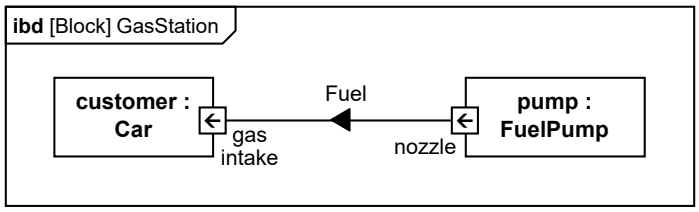

Fig. 2. SysML internal block diagram

Blocks can define equations relating values of datatype properties, usually defined in separate *constraint blocks*. The equation variables are presented as properties called *constraint parameters*, with the equation written in a textual language referring to these parameters by name. Constraint blocks are used in potentially many other blocks by being the type of *constraint properties* on those blocks. Values of constraint parameters are equated to block property values by linking them with *binding connectors* in *parametric diagrams*.

SysML enables extension of its syntax to be defined in models, via *stereotypes*, which are similar to blocks, but with property values directly on blocks they are applied to, rather than on instances of those blocks. For example, a stereotype might have a property identifying the engineers who authored each block, giving names on each block the stereotype is applied to, rather than instances of those blocks during system operation or simulation. Stereotypes and their properties can appear on any diagram showing the elements they are applied to, notated between guillemets («»).[4]

2.3. SysPhS

SysPhS extends SysML to cover information needed for physical interaction and signal flow simulation. It includes:

- An extension of SysML for adding information specific to this kind of simulation.

[4]This extension capability is provided by the Unified Modeling Language® (UML®) [11], which SysML extends. For example, SysML «block» and «flowProperty» are stereotypes of UML Class and Property, respectively.

- A human-usable textual syntax for mathematical expressions. This includes syntax for derivatives, which are always with respect to time only.
- Platform-independent libraries of simulation elements that can be reused in system models.
- Translation patterns between SysML as extended above and two widely-used simulation languages and tools for physical interaction and signal flow simulation (Modelica and Mathworks Simulink®/Simscape™).

Sections 2.3.1 and 2.3.2 describe the language extension and model libraries, respectively.

2.3.1. Stereotypes

SysPhS enables SysML properties to become simulation constants or variables by applying the *PhSConstant* or *PhSVariable* stereotype, respectively, shown in Figure 3 (see Section 2.2 about SysML language extension). Values of constant properties do not change during each simulation run, though they might between simulation runs. They must be properties of components, not ports, because they do not characterize flows between components. The stereotype for variable properties adds this information:

- isContinuous: A boolean telling whether property values change continuously during simulation (true) or discretely (false), defaulting to true. It can only be true for real-valued properties.
- isConserved: A boolean telling whether the property gives a flow rate (true) or a potential to flow (false) during simulation, defaulting to false. It can only be true when isContinuous is true and the extended property is typed by a flow rate for a conserved quantity kind from the SysPhS physical interaction library, see below.
- changeCycle: A non-negative real number for the time interval at which discrete properties change values, defaulting to 0. It can only be positive when isContinuous is false.

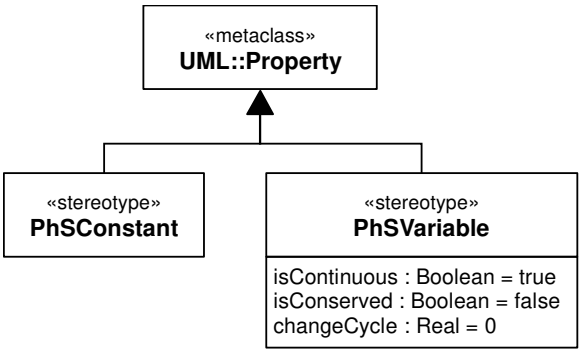

Fig. 3. SysPhS stereotypes

Properties with PhSVariable applied are either on types of

- Ports, for interaction with other components (port variables).[5] They appear on blocks characterizing flow of physical substances through ports, see below.
- Components, for specifying behavior (component variables). They appear as properties internal to component blocks, related by equations to other component properties and port variables on the same component. They are not conserved, because they do not characterize flows, even if they would be considered flow rates when used on ports.

Another property stereotype is PhSConstant which is for defining values which do not change during a simulation. For example, the spring constant of a spring may be

2.3.2. Model libraries

SysPhS provides standard models of commonly needed elements for physical interaction and signal flow modeling, divided by whether they are for ports or components (see Section 2.2 about these). It has two port libraries, for physical interaction and signal flow.[6] The component libraries are mostly for processing real numbers and booleans, plus one for electrical components. This section describes both port libraries and explains how the component libraries are translated to different platforms, as needed for this paper.

Figure 4 shows the SysPhS library for physical interaction ports. These ports are typed by the interface blocks along the bottom row of the figure, which have bidirectional (`inout`) flow properties typed by the blocks just above them, the ones with names starting "Flowing". These introduce properties for flow rates and potentials of physical characteristics as they flow through ports, such as force and velocity for linear momentum, indicated as PhSVariables with isConserved=true or false, respectively. The "flowing" blocks specialize those in the third row, for physical characteristics in general, such as linear momentum and charge, which might be flowing or not. All these physical characteristics are conserved when flowing across links between ports (see Section 2.1.1), as indicted by specializing ConservedQuantityKind, which specializes SysML's QuantityKind, the kinds of physical things (conserved or not) that are measured by units, such as length and force [12].[7]

[5] Connectors in Modelica, connection ports in Simscape

[6] The SysPhS specification refers to these as component interaction libraries, but this paper refers to them as port libraries to avoid confusion with physical interaction.

[7] The same units can measure different quantity kinds, such as newton-meters measuring work and torque. Quantity kinds provide a unit-independent way to identify physical characteristics being measured.

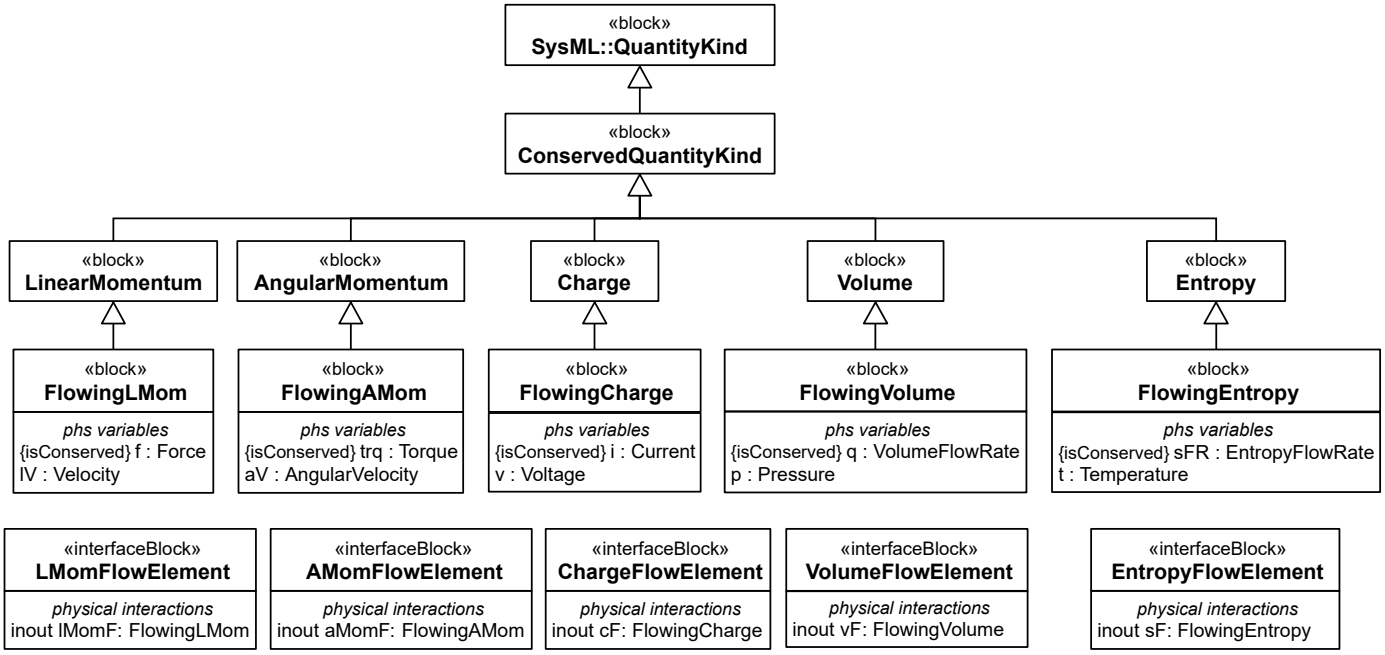

Fig. 4. SysPhS standard library for physical interaction

Figure 5 shows the SysPhS library for signal flow ports. It provides port types for each kind of signal (boolean, real, and integer) in each direction (in and out).

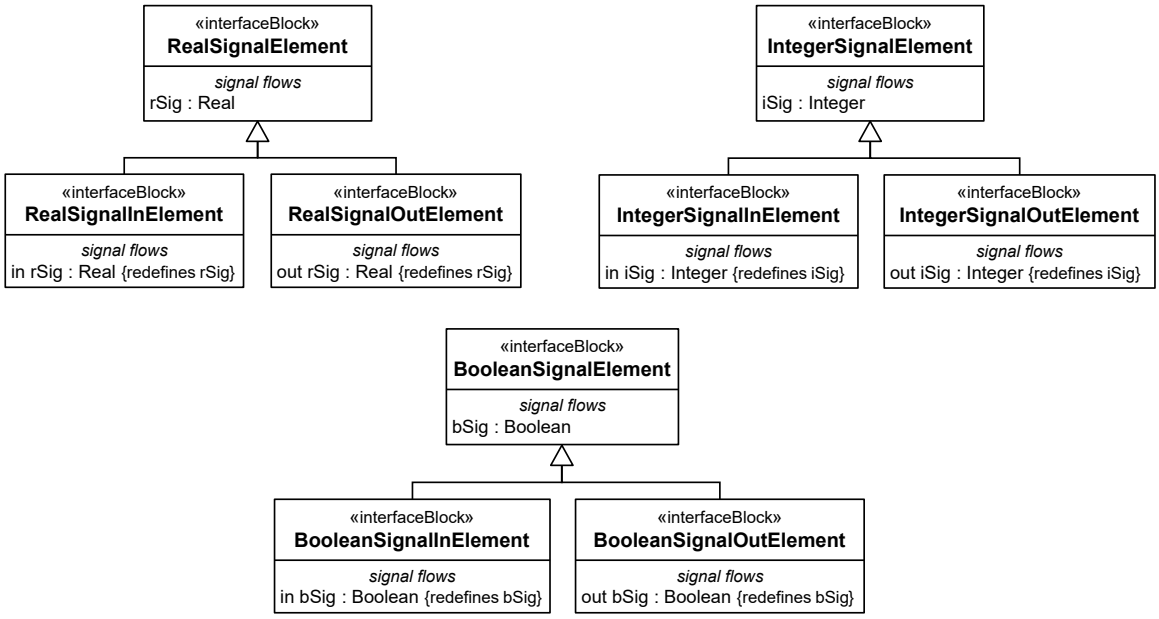

Fig. 5. SysPhS standard library for signal flow

SysPhS also provides libraries of predefined components corresponding to those in common between the two simulation platforms it gives translations for (Modelica and Simulink/ Simscape). These SysPhS components leave it to the platforms to define their behavior, rather than restating it in SysPhS. The libraries are mostly for processing real numbers and booleans, plus one for electrical components. All of them identify corresponding Modelica elements, with the real number and boolean components identifying ones in Simulink, and the electrical components identifying corresponding Simscape elements. This paper only needs real number components, but does not use the ones from SysPhS, see Section 3. SysPhS may be translated to different platforms by a translator[7]. This translator takes in a SysPhS xmi and the user selects the model to be translated. Then either a modelica .mo file or simscape .slx and simscape libraries are output.

3. Real Signal Component Library

The examples in this paper require some real number processing. SysPhS provides most of the real signal blocks needed, but they correspond to Simulink elements (see Section 2.3.2), which are not easily used with Simscape, one of the platforms for simulating SysPhS physical interaction (see Section 2.1.3). This section redefines a few of these real signal blocks and adds one specifically for control, as shown in Figure 6. Most blocks include constraint properties referencing constraint blocks defined in Figure 7 (see Section 2.2 about constraint modeling in SysML). All are described with their parametric diagrams in the rest of the section.

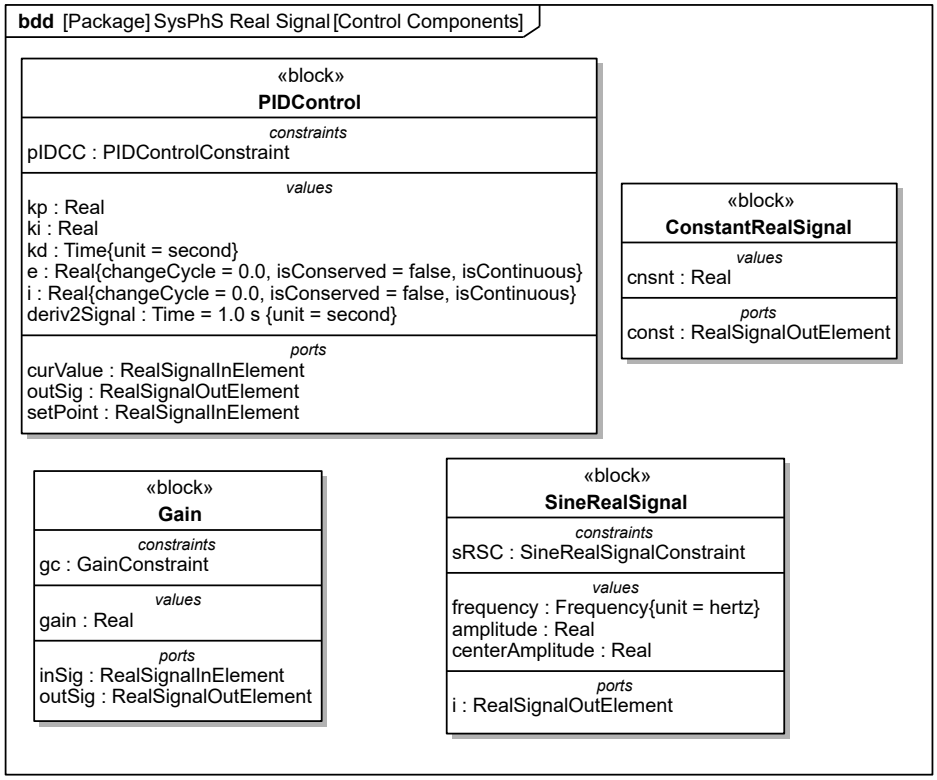

Fig. 6. Real signal component library

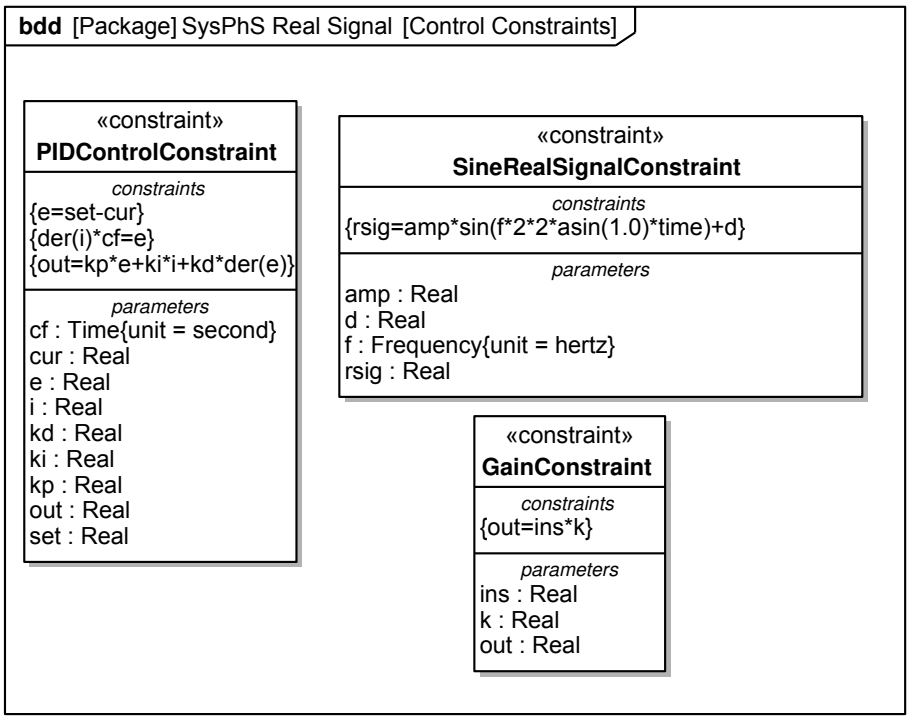

Fig. 7. Real signal component library constraints

ConstantRealSignal outputs a real signal that is constant over time, defined by the PhSConstant cnsnt, as shown in Figure 8.

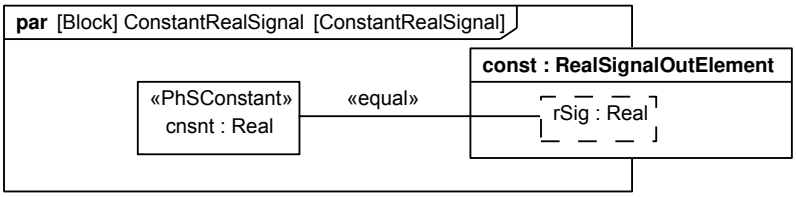

Fig. 8. Constant parametric diagram

Gain takes in a real signal and outputs a real signal multiplying it by the constant value of gain, as shown in Figure 9.

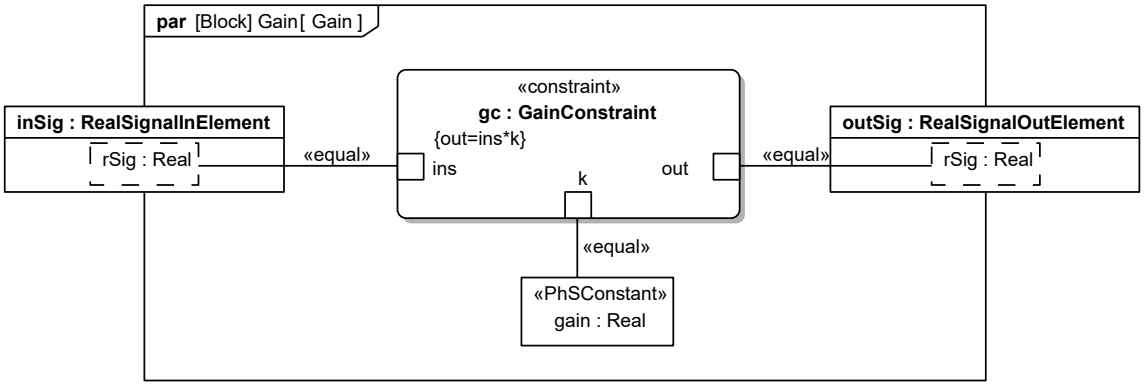

Fig. 9. Gain parametric diagram

SineRealSignal, shown in Figure 10, outputs a real signal that forms a sine wave over time. The PhSConstants frequency, amplitude, and offset characterize the wave, with frequency in units of Hz, oscillating around the value of offset.

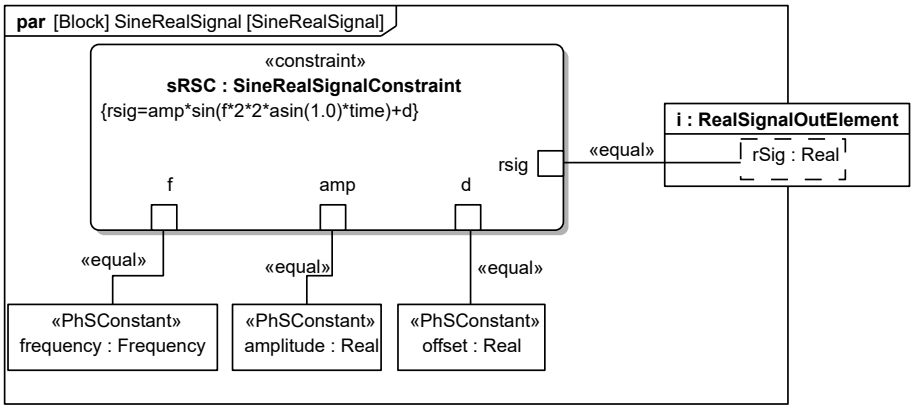

Fig. 10. SineRealSignal parametric diagram

PIDControl, shown in 11, models Proportional Integral Derivative (PID) controllers. The input setPoint gives the desired value of the plant output as a realSignal, curValue is the present value of the plant output, and outSig is the output control value to be provided to the plant. The proportional coefficient is kp, the integral coefficient is ki, and the derivative coefficient is kd.

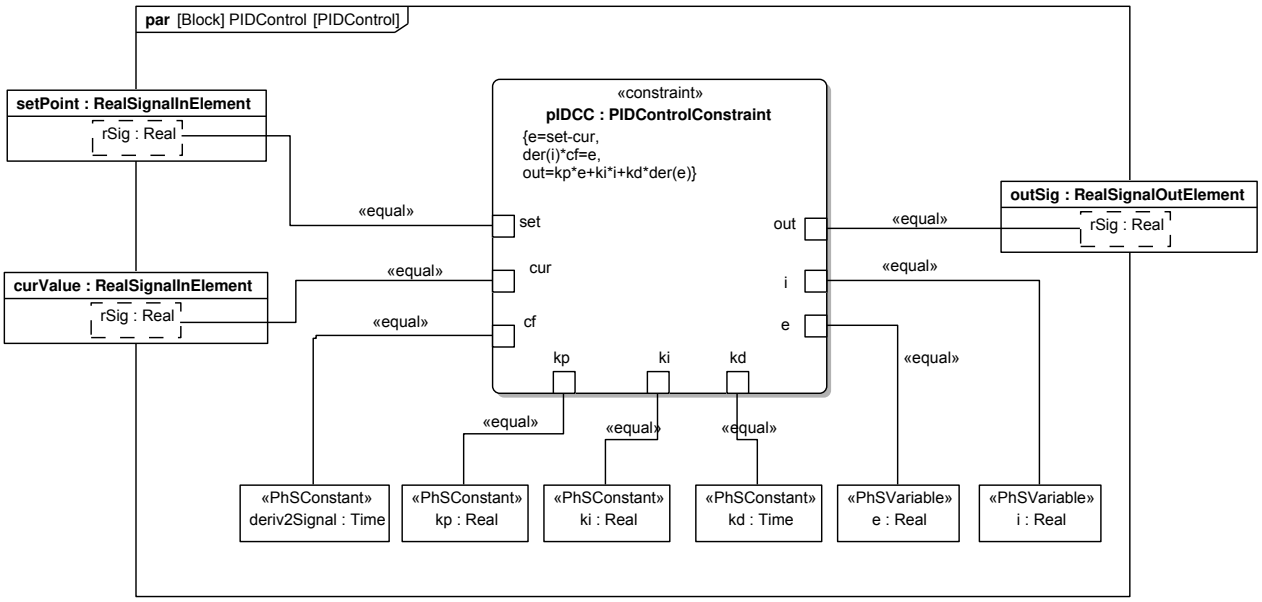

Fig. 11. PIDControl parametric diagram

The control signals in these libraries are real signals that do not have units, to avoid defining new interface blocks that add units to the real signal blocks in SysPhS. For example, PIDControl, which is widely applicable as it is, would need many specialized interface blocks with various units. Simscape, a platform SysPhS gives translations for, requires the two sides of each equations have the same units. However, many elements, such as sensors and source elements, have equations that relate physical quantities that have units to unitless real signals. The library introduces conversion factors with units that balance units on each of an equation by multiplying a real signal to add units or dividing a physical quantity to remove units. For example, a velocity sensor that outputs a unitless real signal needs to have an equation where both sides do not have units. The sensed velocity must be multiplied with a factor with inverse units of velocity to remove its units. Equations in these libraries that convert between unitless quantities and those with units are written to not require implementing new units. For example, the equation for converting a quantity with units into a real signal is typically signal = q/cf, where q is the quantity to be converted and cf is the conversion factor in the same units as quantity. This avoids introducing units that are inverse of those of quantity.

4. Physical Interaction Libraries

This section presents physical interaction component libraries for areas not covered by SysPhS currently (see Section 1). They are defined with the standard SysPhS extensions and physical interaction library (see Figures 3 and 4 in Section 2.3), enabling them to translated in a standard way to simulation tools and languages. The libraries are for translational mechanics, rotational mechanics, and entropy (heat) transfer, in Sections 4.1, 4.2, and 4.3, respectively.

4.1. Translational Mechanics Library

The translational mechanics library enables SysPhS modeling of mechanical components moving in one dimension, including point masses affected by springs, dampers, and inertia, but not changing orientation (angle). It also supports linear motion along multiple independent axes, for pseudo-2D and 3D modeling.

Figure 12 shows an example translational system to illustrate the library. It is a mass-spring-damper attached to a fixed wall, with a sinusoidally varying force applied to the mass. A sinusoidal force f is applied to m_1, giving and taking linear momentum from it, which passes through k_1 and d_1 to and from the fixed wall. The sinusoidal force and fixed wall make this an open system, but the change in system momentum equals the net impulse applied to the system (force integrated over time, which is linear momentum). The mass develops some oscillation, but its exact behavior is dependent on the values of parameters and starting conditions chosen.

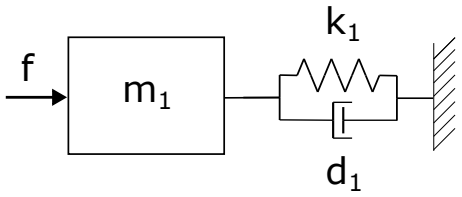

Fig. 12. An example translational system

The system in Figure 12 is modeled with SysPhS in Figure 13, an IBD connecting components defined by the translational mechanics library BDD in Figure 14 (see Section 2.2 about these kind of diagrams). Figure 13 refers to blocks in library by their names, appearing to the right of the colon at the top of each larger rectangle. The role each block plays in the system appears to the left of the colon in each, following the labels in Figure 12. The force f applied to $m1$ varies as a sine wave, controlled by a component via a signal port, notated by a small rectangle with an unidirectional arrow inside (see Section 3 about signal components).

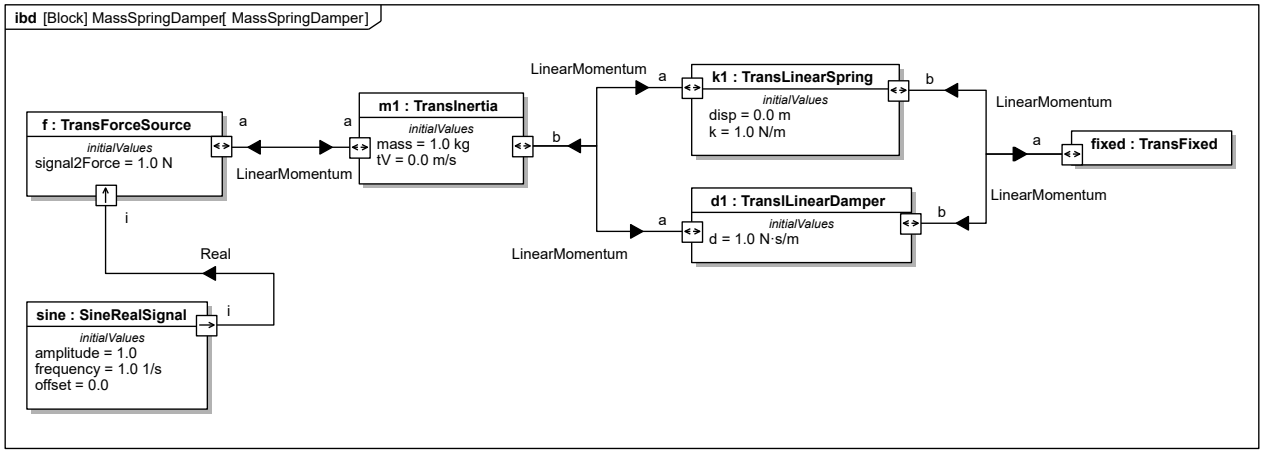

Fig. 13. Figure 12 modeled in SysPhS with initial conditions and sources

Figure 13 connects translational components at their physical interaction ports, notated by small rectangles with bidirectional arrows inside. Flows of (linear) momentum through these ports are described by their force and (linear) velocity, which are momentum's rate of flow and potential to flow, respectively (conserved and non-conserved variables, respectively).[8] SysPhS uses force and velocity to enable flows between components to be taken as energy flows, with rate of energy flow (power) being the product of the variables. Force is the rate of flow of momentum, following the physical definition that it is the rate of change of momentum. Velocity is the potential to flow of momentum, since two objects moving at the same velocity cannot exchange momentum (see Section 2.1.1 about potentials). Velocity, acceleration and force have a direction, indicated by the sign of their variables. This library assumes the sign of acceleration and the force causing it are the same.

Figure 14 defines the translational mechanics library introduced by this paper and used in Figure 13. All the components have physical interaction ports for linear momentum and some have real signal ports (see Section 2.3.2). The library includes TwoFlangeTransComponent, which has two momentum ports and a variable forceThru for the force it exerts or exerted on it (rate of momentum flow through the component), as defined in its specializations. One of these is CompliantTransComponent, where the ports can move relative to each other along the same line, such as springs and dampers. It has two variables, relV for the relative velocity between the ports, and disp, for the relative displacement between them.

[8] Some translational mechanics modeling software use absolute position and force as port variables, such as the Modelica Standard Library for Translational Mechanics [13].

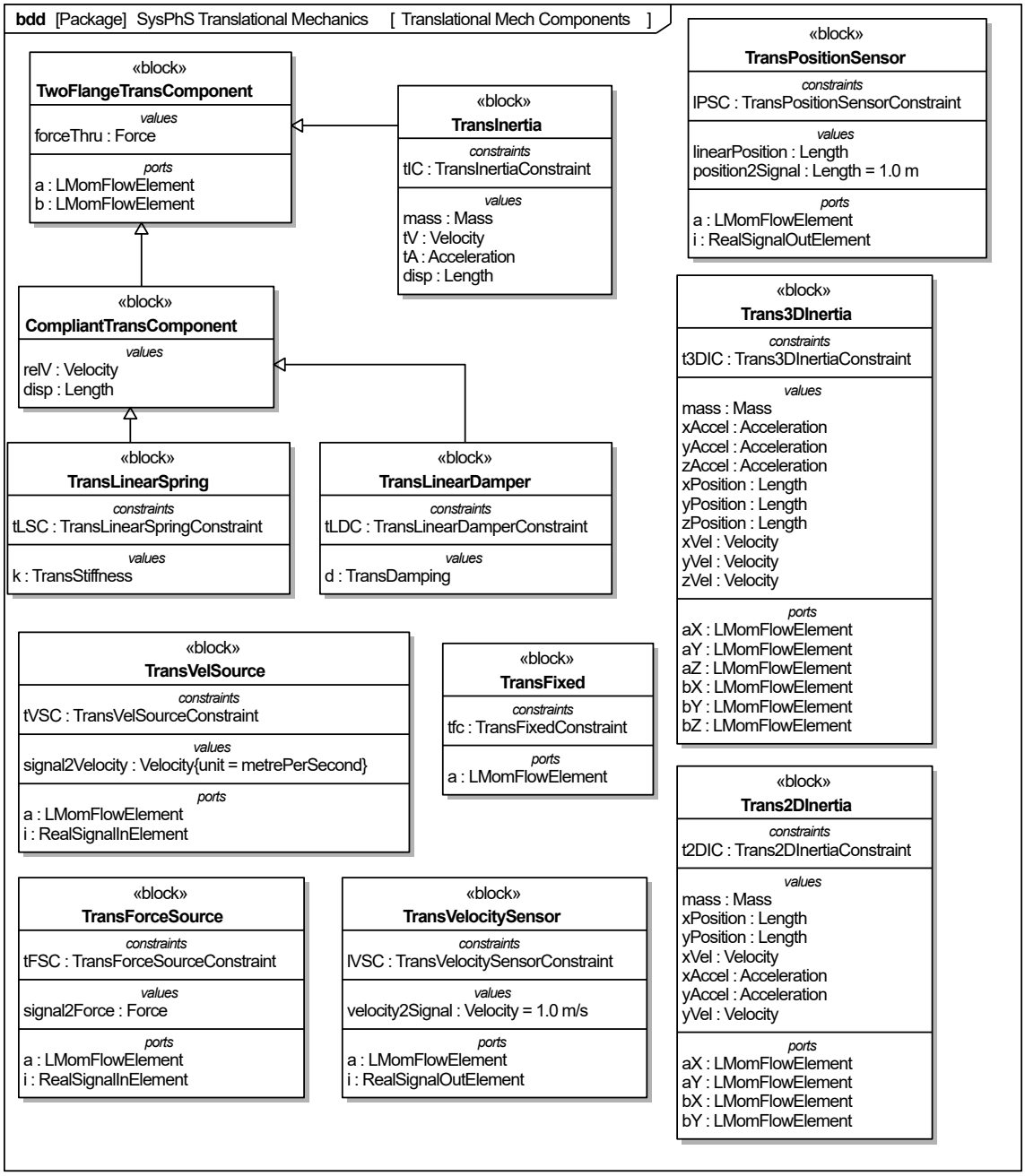

Fig. 14. Translational mechanics library

Figure 15 defines the constraint blocks referenced by constraint properties in Figure 14, appearing in a compartment of each translational library block (see Section 2.2 about constraint modeling in SysML). BinaryCompliantTransConstraint defines constraints for all components with two ports that might have different velocities, corresponding to system components with two ends that might move with respect to each other, such as springs and dampers. Forces on these components sum to zero, as specified by fa + fb = 0, while

relV=Vb-Va defines the relative velocity between ports a and b.[9] The derivative of displacement is relative velocity, expressed as der(disp)=relV, which means displacement is an integral of relative velocity. Initial displacement must be defined if it is not supplied by additional equations.

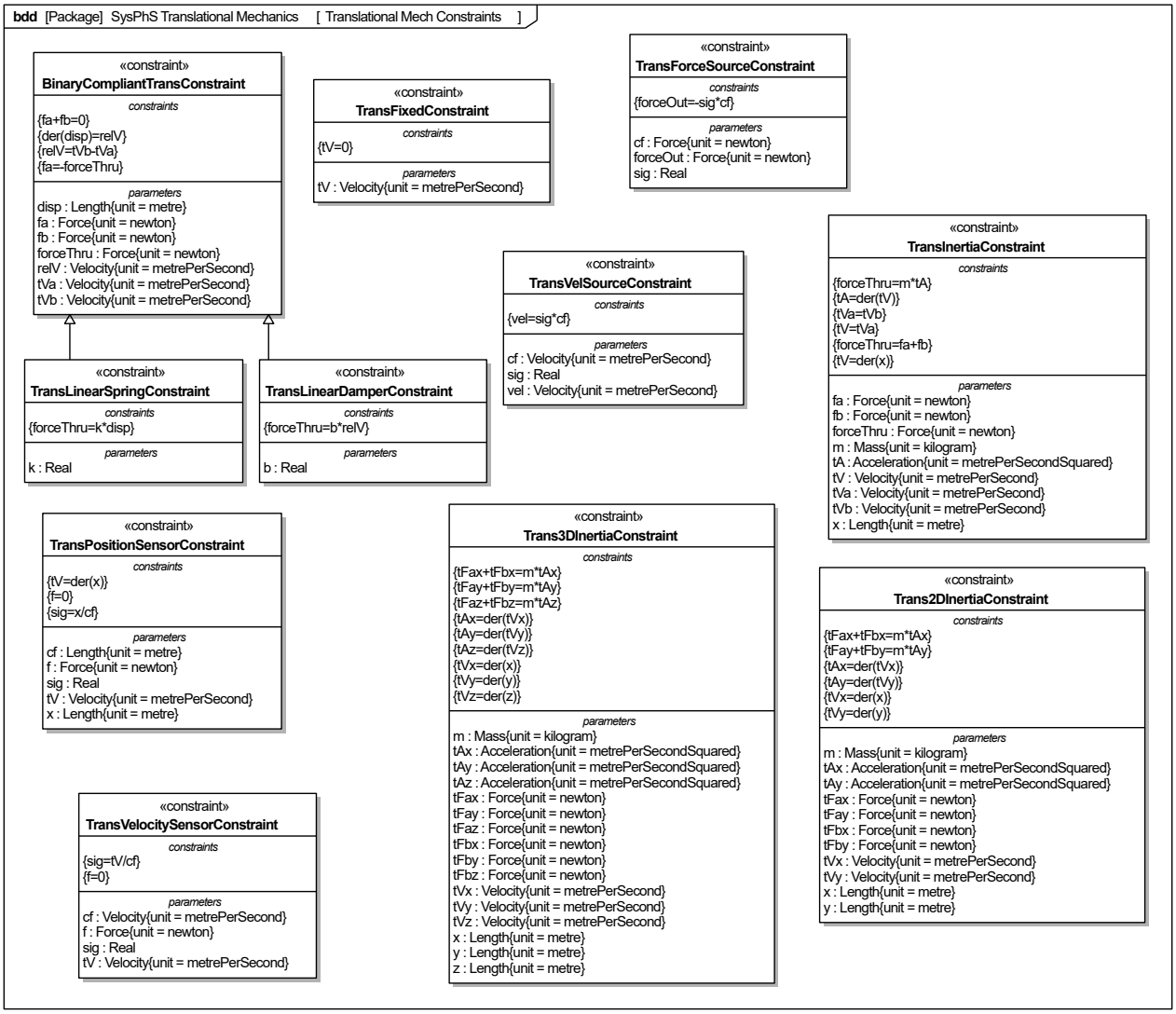

Fig. 15. Constraints for the translational mechanics library

[9] BinaryCompliantTransConstraint is analogous to BinaryElectricalComponentConstraint in the electrical example in SysPhs Annex A1.2 [5]. The two equations above in BinaryCompliantTransConstraint are analogous to current at the ports summing to zero and voltage drop across the ports, respectively.

The rest of this section covers the other components in Figure 14 and parametric diagrams that bind their properties to constraint parameters in Figure 15 (see 2.2 about these kind of diagrams).

The TransLinearSpring block in Figure 14 models a component in which force exerted at its ports is linearly proportional to the displacement between them (Hooke's Law), without any losses or inertia. This is expressed in TransLinearSpringConstraint in Figure 15, a kind of BinaryCompliantTransConstraint. Figure 16 shows a parametric diagram for TransLinearSpring that binds its properties to parameters from TransLinearSpringConstraint. Properties on the momentum ports bind to parameters on the constraint. TransLinearSpring has a parameter k for stiffness in units of N/m. The spring k_1 in Figure 12 is modeled with TrsntlLinearSpring in Figure 13.

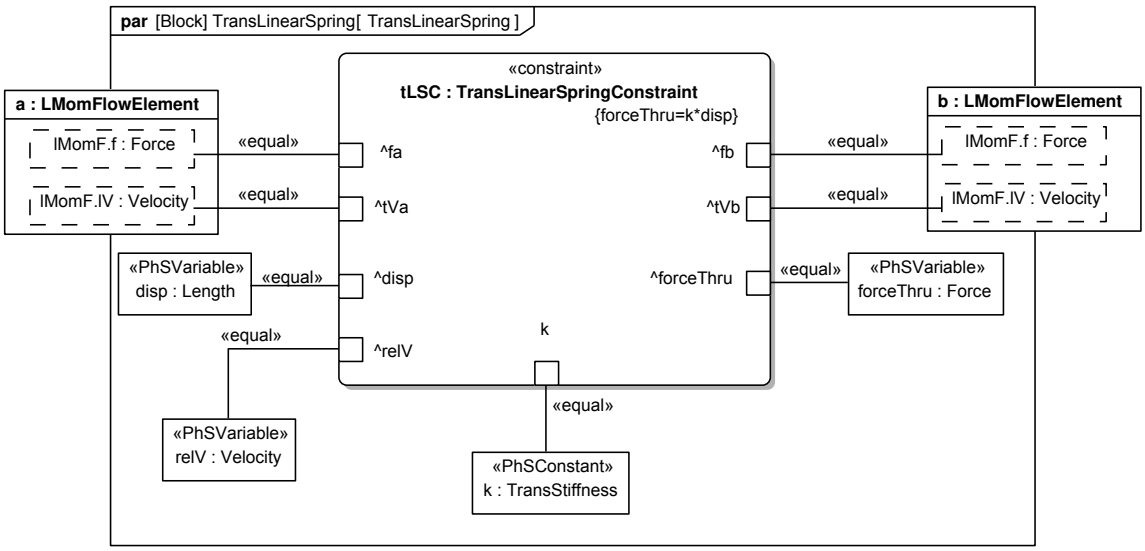

Fig. 16. TransLinearSpring parametric diagram

TransLinearDamper in Figure 14 models a component in which the force between the two ports is linearly proportional to the difference in velocity between its ports, as expressed in TransLinearDamperConstraint in Figure 15, a kind of BinaryCompliantTransConstraint. Figure 17 shows the parametric diagram that binds its properties to constraint parameters in TransinearDamperConstraint. It is applicable to dampers with a linear response, such as viscous friction between two objects or elastic deformation with heat loss. TransLinear-Damper has a parameter k specifying the damping coefficient in N*s/m. The damper k_1 in Figure 12 is modeled with a TransLinearDamper.

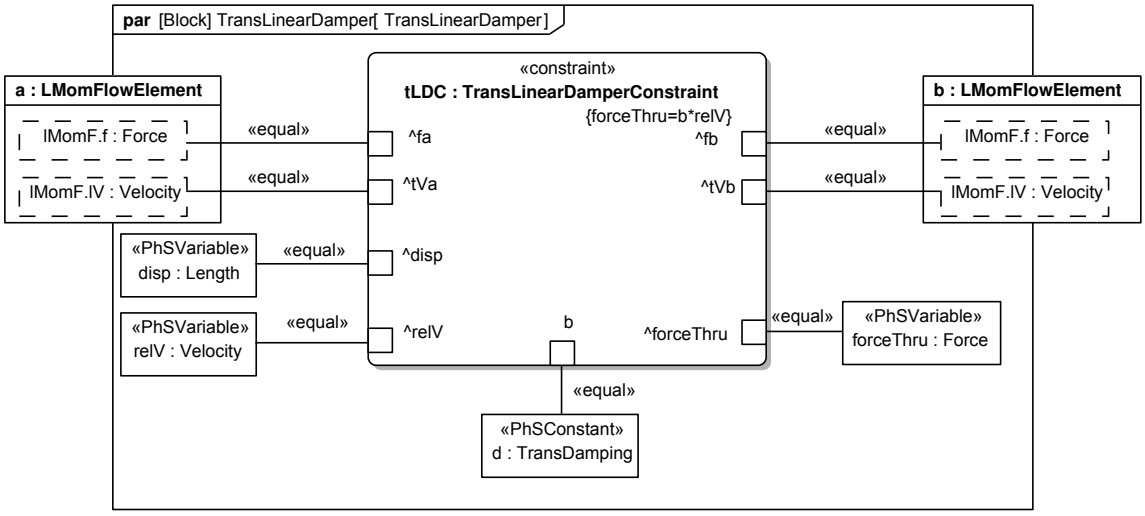

Fig. 17. TransLinearDamper parametric diagram

TransInertia in Figure 14 models a component where its acceleration is proportional to an applied force (a point mass), as expressed in TransIntertiaConstraint in Figure 15 by the equation $F = ma$. Figure 18 shows the parametric diagram that binds its properties to constraint parameters in TransIntertiaConstraint. It has two ports that have the same variable values, to resemble typical systems where other components are attached to either 'side' of the inertial object, as if the ports were moving at the same velocity (rigidly fixed to each other), though it is not necessary to connect to both ports.

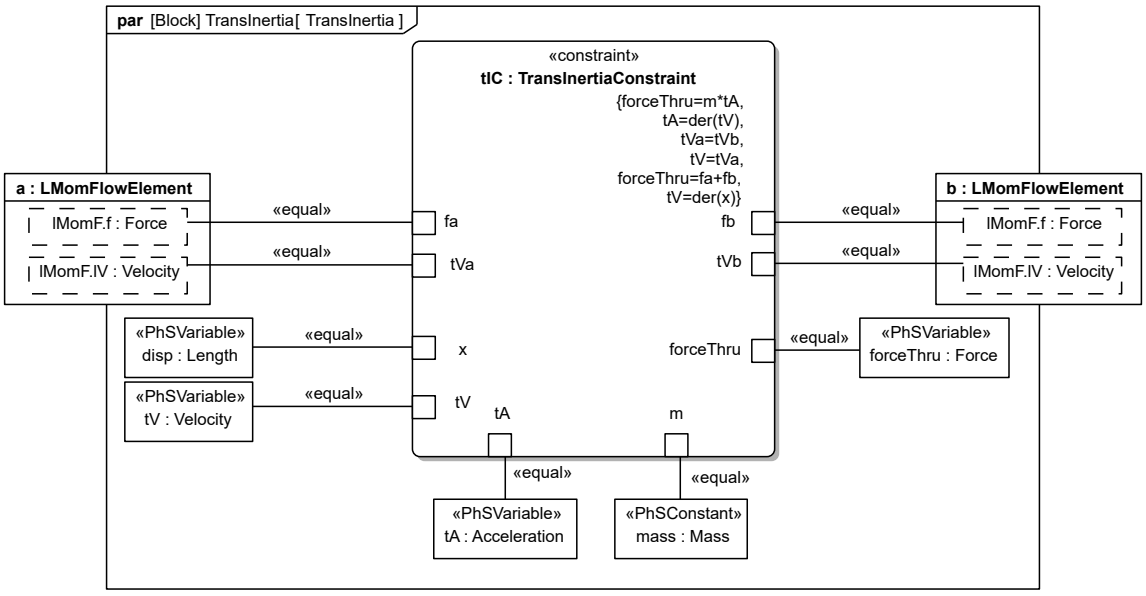

Fig. 18. Translational inertia parametric diagram

The sign convention in this library is such that a a positive force will create a positive

acceleration on a mass. Regardless of which side a positive force is applied on, a positive acceleration will result.

Trans2DInertia and Trans3DInertia in Figure 14 model components that can translate along two and three perpendicular axes, but not rotate, enabling pseudo-2D and 3D simulation, as in Sections 5.2 and 5.3. They have two ports for each degree of freedom.[10] This is simpler and less error-prone than modeling the same mass repeatedly for the dynamics of each axis separately, which would need to be kept consistent. The parametric diagrams for Trans2DInertia and Trans3DInertia are shown in Figure 19 and Figure 20, respectively.

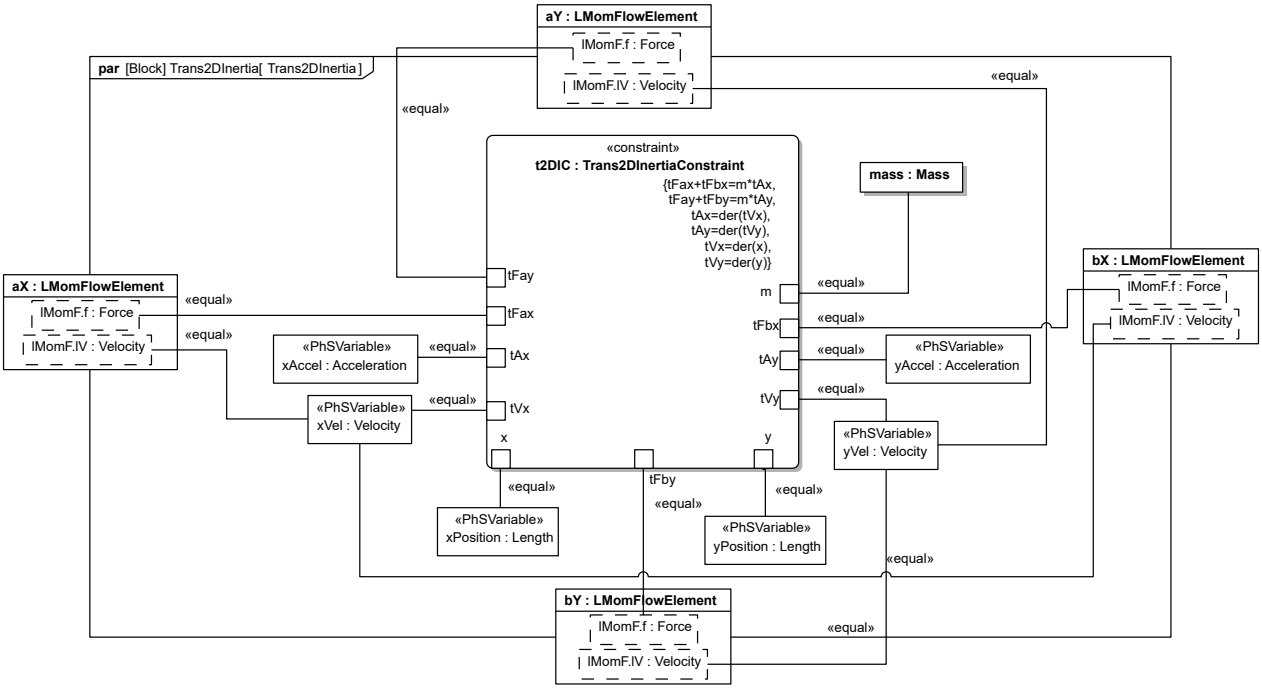

Fig. 19. Translational 2D inertia parametric diagram

[10] TransInertia with only two ports is 1D.

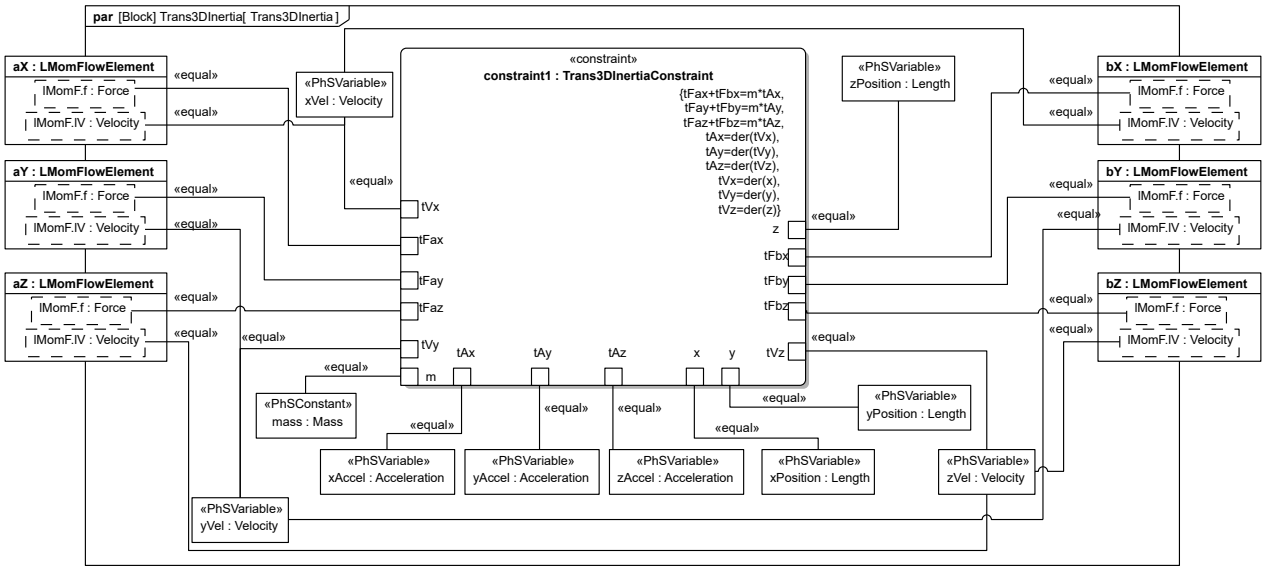

Fig. 20. Translational 3D inertia parametric diagram

TransVelocitySensor and TransPositionSensor in Figure 14 are components that have a linear momentum port and a signal output port that produces a real number proportional to the linear velocity or position derived from the momentum port variables, respectively. Neither affects the momentum of components connected to the momentum port, as ensured by the rate of flow through the port (force) always being zero, as shown by their parametric diagrams in Figures 21 and 22. TransVelocitySensor has a parameter 'velocity2Signal' for the inverse conversion factor between output signal and velocity measured on the port given in units of m/s. TransPositionSensor integrates the velocity of the momentum port over time and starting at TransPosition, the initial position. The parameter 'position2Signal' gives the inverse conversion factor between position and realSignal given in units of m.

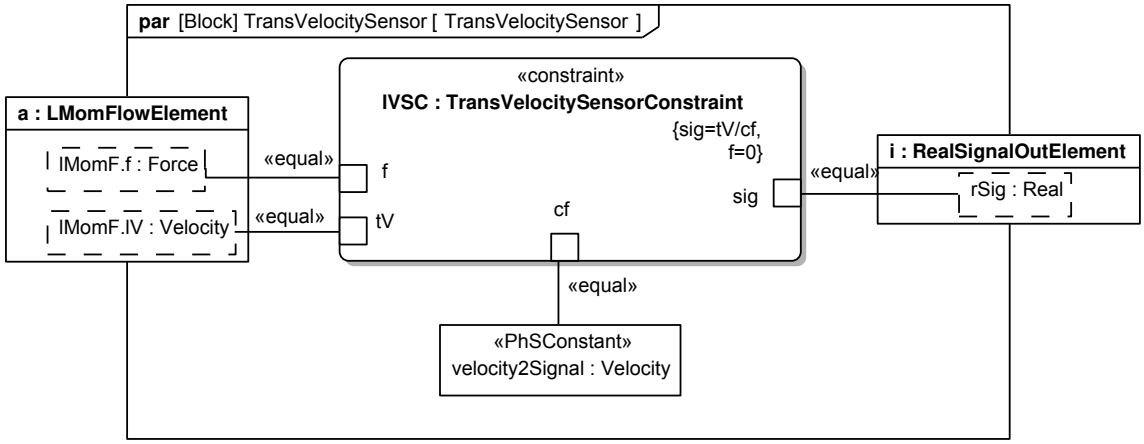

Fig. 21. TransVelocitySensor parametric diagram

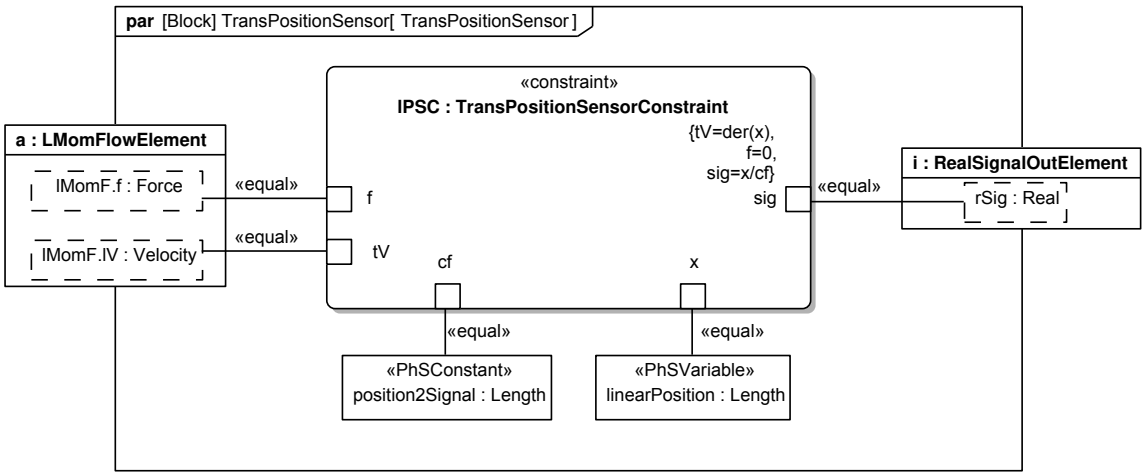

Fig. 22. TransPositionSensor parametric diagram

TransForceSource and TransVelSource in Figure 14 are components that accept a real signal and apply a force or velocity to other components via a momentum port, respectively. Figures 23 and 24 show the parametric diagrams for these components. TransForceSource has a parameter signal2Force for the conversion factor between signal and force in units of N. The force f in Figure 12 is modeled with a TransForceSource connected to a sinusoidal realSignal in Figure 13. TransVelSource applies a velocity is relative to a fixed frame. The parameter 'signal2Velocity' gives the conversion factor between velocity and realSignal in units of m/s.

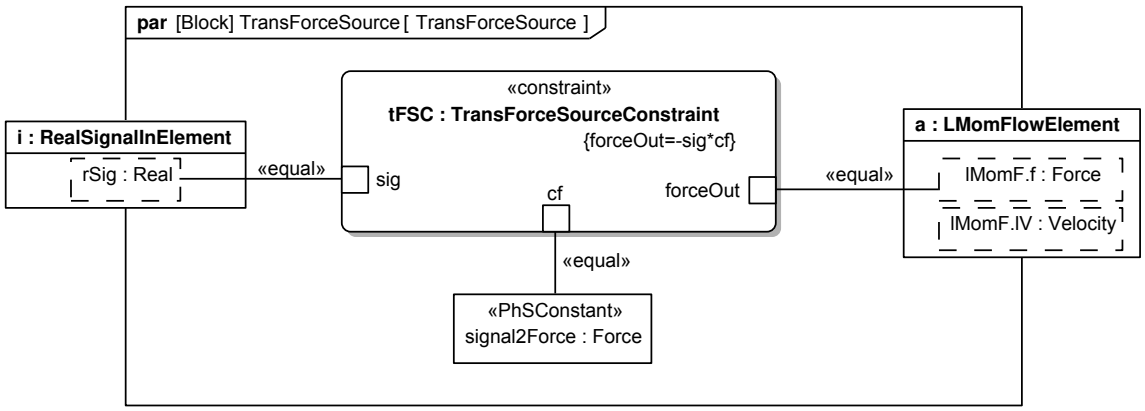

Fig. 23. Parametric diagram for force source

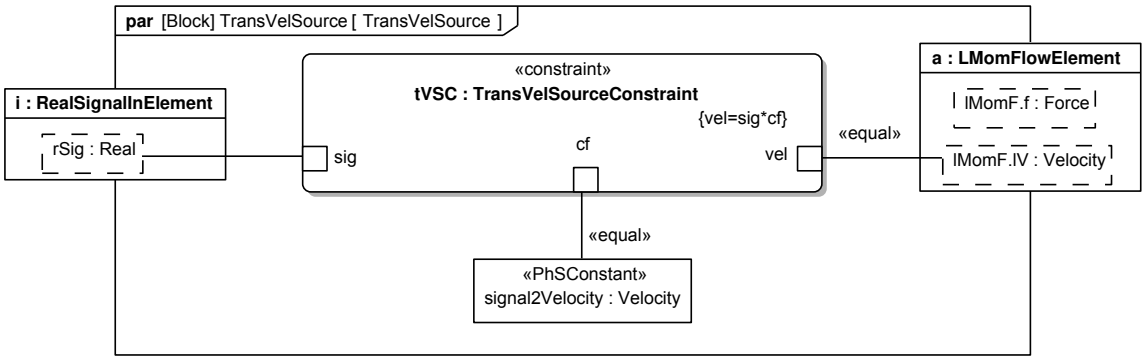

Fig. 24. Parametric diagram for velocity source

TransFixed in Figure 14 has only a momentum port, with a velocity required to be zero, as shown in Figure 25. The fixed boundary condition in Figure 12 is modeled by TransFixed in Figure 13.

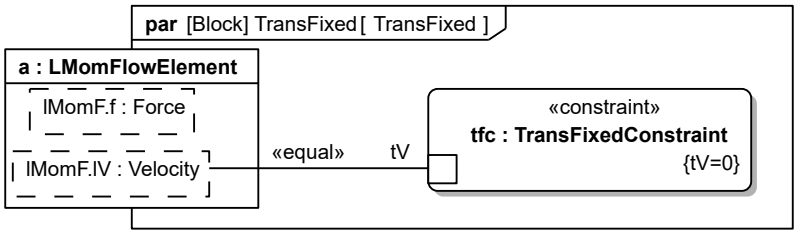

Fig. 25. Parametric diagram for fixed boundary condition

4.2. Rotational Mechanics Library

The rotational mechanics library enables SysPhS modeling of mechanical components changing orientation (angle) in one dimension, such as rotary springs, dampers, inertia, as well as gear trains, but not changing position. Position and orientation of rotational axes, and multi-axis rotational effects, such as gyroscopic precession, are not considered.

Figure 26 shows an example rotational system, based on one from [14], Figure 4.16. It consists of flywheels with significant inertia (i1,i2,i3) connected by long shafts (k1,k2) acting as torsional springs, gears (G1,G2) with non-negligible inertia (i4,i5), a torsion spring (k3), a rotary damper (d3) and an angle sensor.

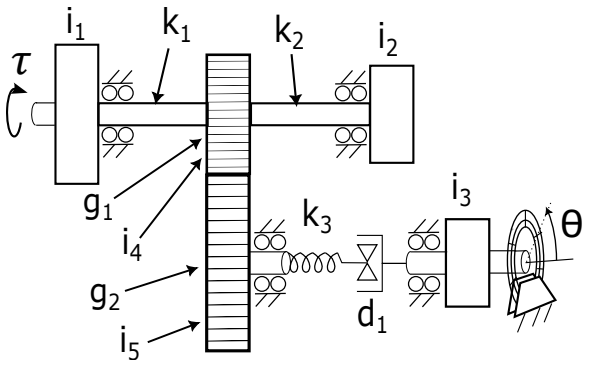

Fig. 26. An example rotational system

The system in Figure 26 is modeled with SysPhS in Figure 27, an IBD connecting components defined by the rotational mechanics library BDD in Figure 28 (see Section 2.2 about these kind of diagrams). Figure 27 refers to blocks in library by their names, appearing to the right of the colon at the top of each larger rectangle. The role each block plays in the system appears to the left of the colon in each, following the labels in Figure 27. The applied torque τ to i_1 varies as a sine wave, controlled by a component by a signal (unidirectional) port (see Section 3 about signal components).

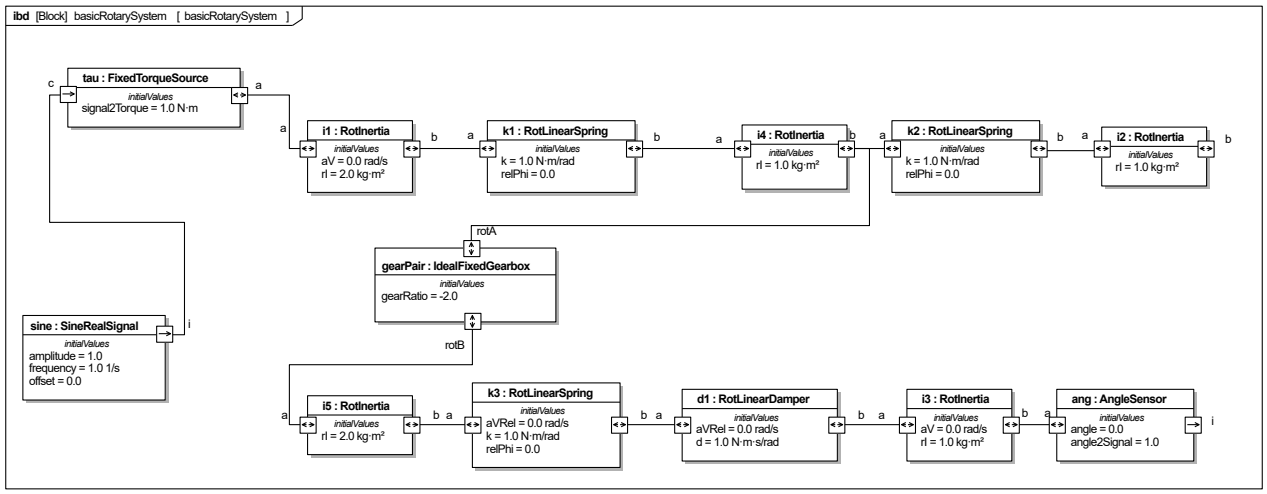

Fig. 27. Figure 26 modeled in SysPhS with initial conditions and sources

Figure 27 connects rotary components at their physical interaction ports, notated by small rectangles with bidirectional arrows inside. Flows of angular momentum through these ports are described by torque and angular velocity, which are angular momentum's rate of flow and potential to flow, respectively (conserved and non-conserved variables, respectively).[11] Angular velocity is the potential to flow of angular momentum, since two objects

[11] Some rotational mechanics modeling software use absolute angle and angular velocity as port variables,

rotating at the same velocity cannot exchange angular momentum. Angular velocity, acceleration and torque have a direction, indicated by the sign of their variables. This library assumes the sign of angular acceleration and the torque causing it are the same. Torque is the rate of change of angular momentum, allowing allow one to consider angular momentum flow separately from rotational inertia.

Torques on ports of the same rotational inertia element can differ when some of the angular momentum is stored in the element or released from it. Figure 28 defines the rotational mechanics library introduced by this paper and used in Figure 27. It includes TwoFlangeRotComponent, which includes components with two angular momentum ports. CompliantRotComponent is a TwoFlangeRotComponent where the ports rotate relative to each other, such as rotational springs and dampers. It has two component variables, aVRel for the relative angular velocity between the ports and torqueThru for the rate at which angular momentum is flowing through the component. This component does not include a variable for relative displacement between the two ports, because it may be desirable to have rotary components which do not integrate displacement (relPhi).

such as the Modelica Standard Library for Rotational Mechanics [15]. SysPhS uses torque and angular velocity to enable flows between components to the taken as energy flows, with rate of energy flow (power) being the product of the variables.

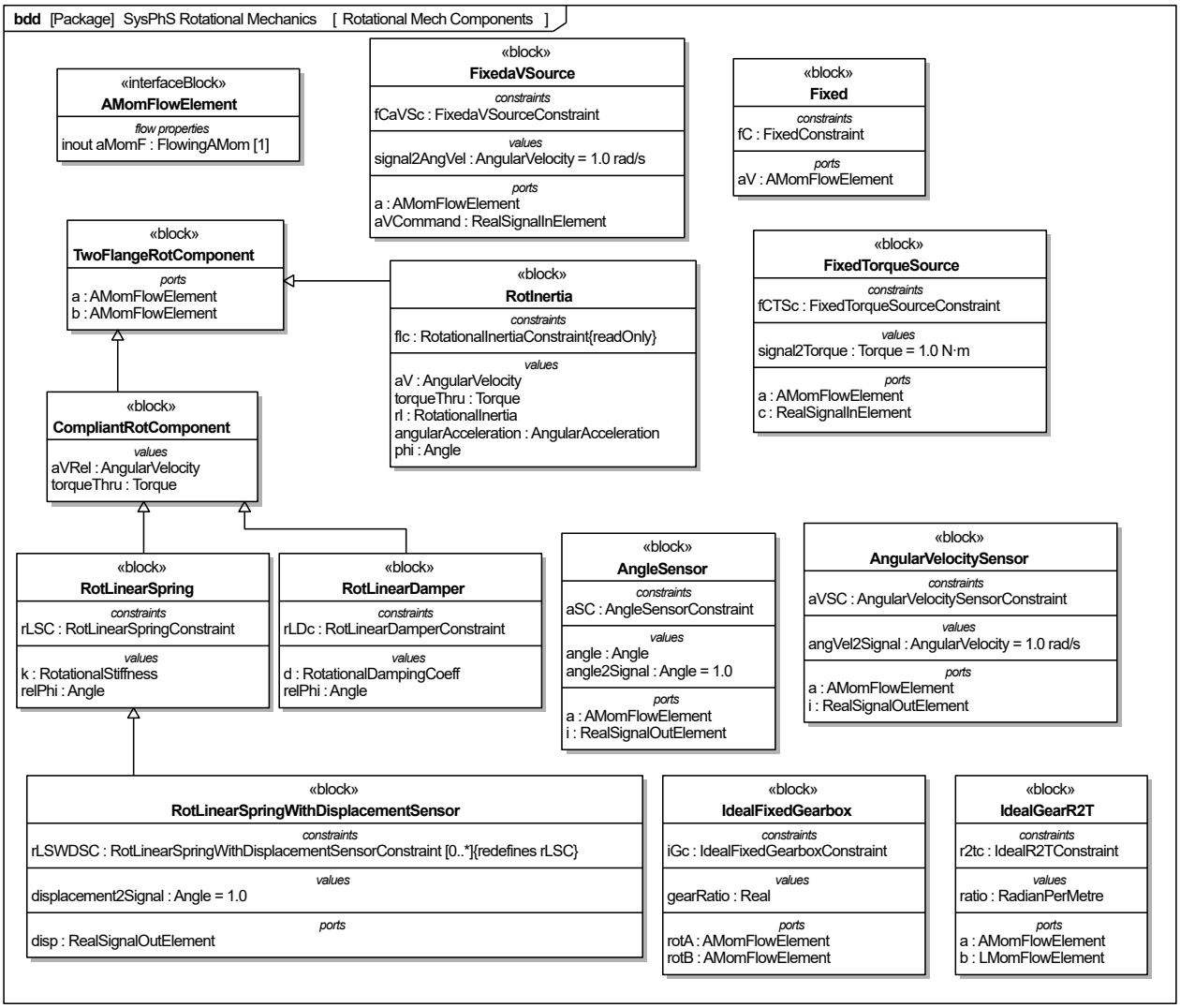

Fig. 28. Rotational mechanics library

Figure 29 defines the constraint blocks referenced by constraint properties in Figure 28, appearing in compartments of each rotational library block (see Section 2.2 about constraint modeling in SysML). BinaryCompliantRotConstraint defines constraints for all components with two ports that might have different angular velocities, corresponding to system components with two ends that might rotate with respect to each other, such as rotational springs and dampers. Torques on these components sum to zero, as specified by ta + tb = 0, while aVRel=aVb-aVa defines the relative angular velocity between ports a and b.[12] The derivative of relative angle is relative angular velocity, expressed as der(relPhi)=aVRel, which means relative angle is an integral of relative angular velocity. Integration requires an initial value, defined with a PhSVariable or by additional equations.

[12]Footnote 9 in Section 4.1 about applies to BinaryCompliantRotConstraint also, except for angular momentum instead of linear.

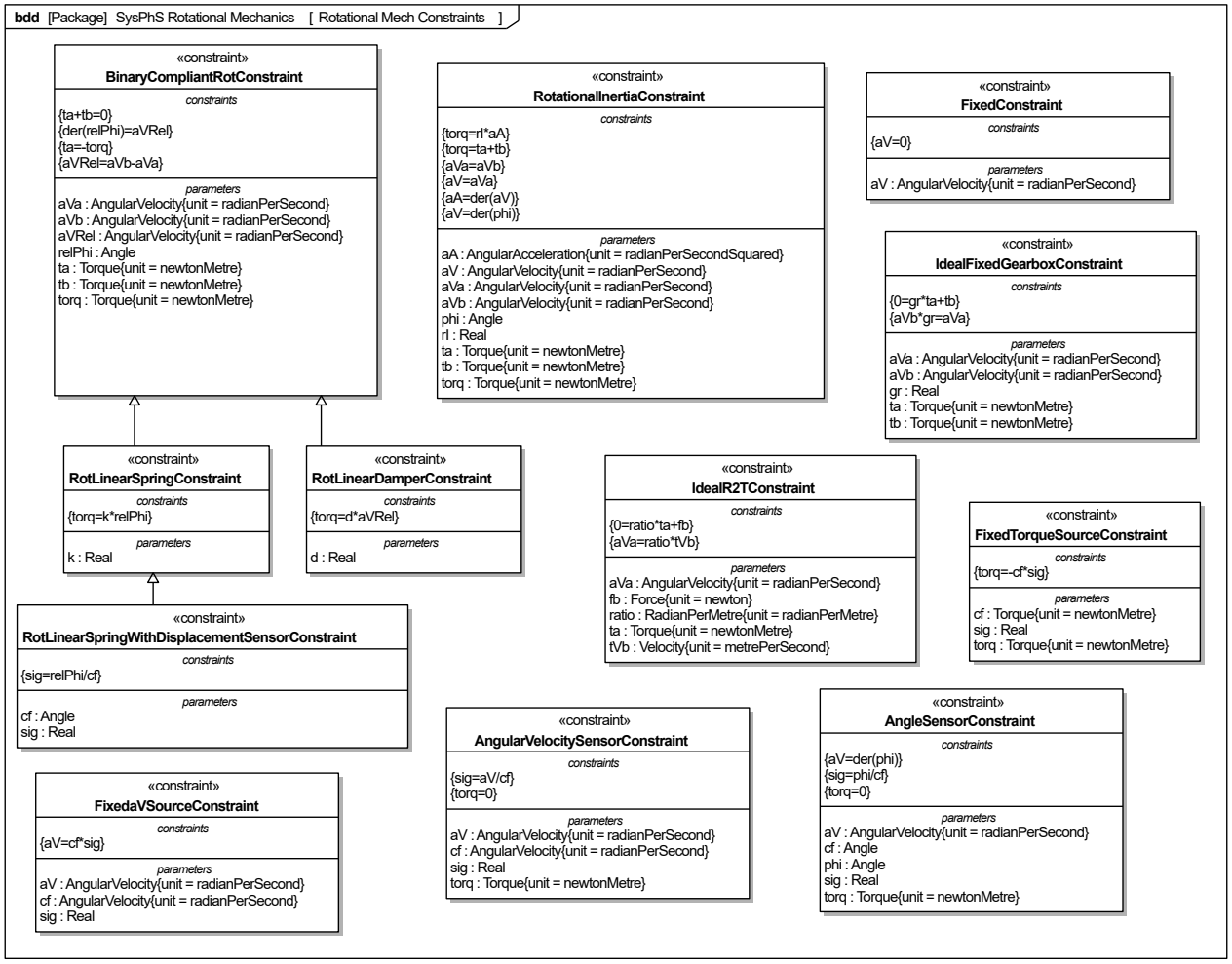

Fig. 29. Constraints for rotational mechanics library

The rest of this section covers the components in Figure 28 and parametric diagrams that bind their properties to constraint parameters in Figure 29 (see 2.2 about these kind of diagrams).

The block RotLinearSpring in Figure 28 models a torsional spring in which torque is linearly proportional to angle between its ports, without any losses or inertia, as expressed in RotLinearSpringConstraint, a specialization of BinaryCompliantRotConstraint in Figure 29. Figure 30 shows the parametric diagram for RotLinearSpring. That is this spring implements the rotary version of Hooke's law. RotLinearSpring has a parameter k for rotational stiffness which is in units of N*m/rad. The RotLinearSpringConstraint is derived from BinaryCompliantRotConstraint, so angle is calculated by integrating relative angular velocity between the two ports. In Figure 26, the long shafts k_1, k_2, and torsional spring k_3 are modeled using RotLinearSpring blocks in Figure 27.

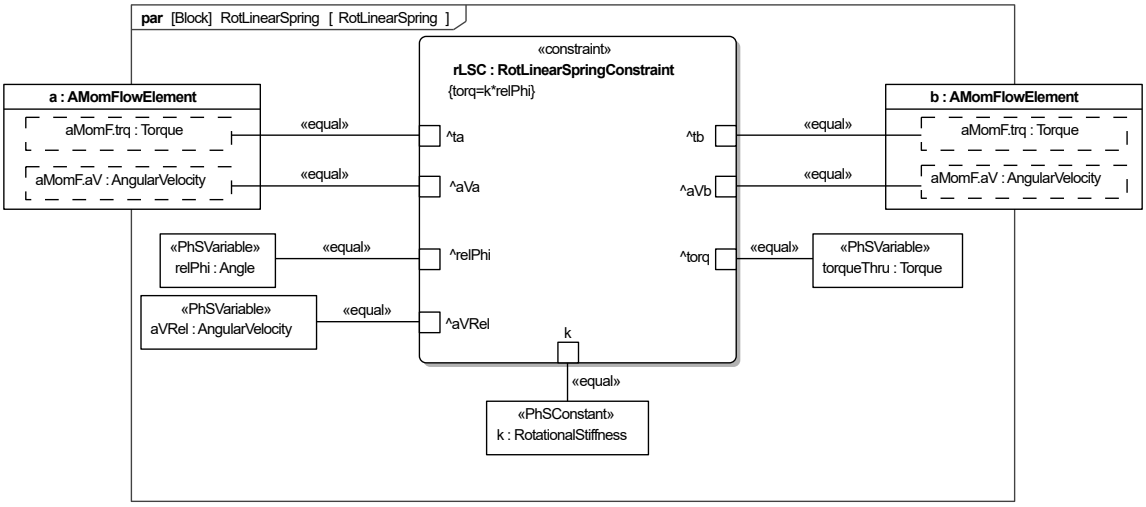

Fig. 30. Rotary linear spring parametric diagram

RotLinearDamper in Figure 28 defines a component where torque is linearly proportional to the angular velocity difference between the two ports, as expressed in RotLinearDamper-Constraint in Figure 15. Figure 31 shows a parametric diagram for RotLinearDamper that binds its properties to parameters from TransLinearSpringConstraint. Figure 31 shows the parametric diagram for RotLinearDamper. Rotational damping coefficient is defined in units of N* m*s/rad. This component exhibits no inertia and torque is linearly proportional to velocity difference. This component is used to model rotary damper k_1 in Figure 26. This may also be used to model linear/viscous friction between components or be used to add a simple model of mechanical loss. Although care should be taken when using this component to model friction as viscous friction and coulomb friction result in very different behavior.

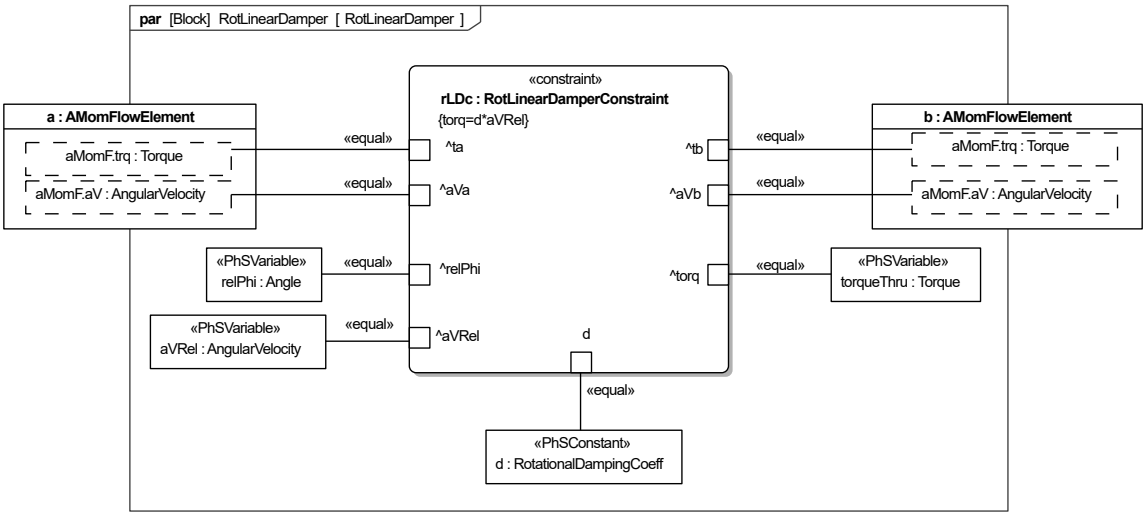

Fig. 31. Rotary damper parametric diagram

One component in Figure 28 that does not use a constraint generalized from BinaryCompliantConstraint is RotInertia, where the rate of change of angular velocity of a rotational inertia is proportional to applied torque (the rotational equivalent of $F = ma$), as expressed in RotInertiaConstraint in Figure 15. Figure 32 shows the parametric diagram for RotInertia that binds its properties to parameters from RotInertiaConstraint.RotInertia has two ports that have the same variable values, to resemble typical systems where other components are attached to either 'side' of the inertial object, as if the ports were rotating at the same angular velocity (rigidly fixed to each other), though it is not necessary to connect to both ports. This means Switching the connections on each 'side' of RotInertia does not change the direction of rotation. The whole inertial element has the same angular velocity, which is given with respect to a global frame. RotInertia also has an angle which is given relative to a user specified start angle.

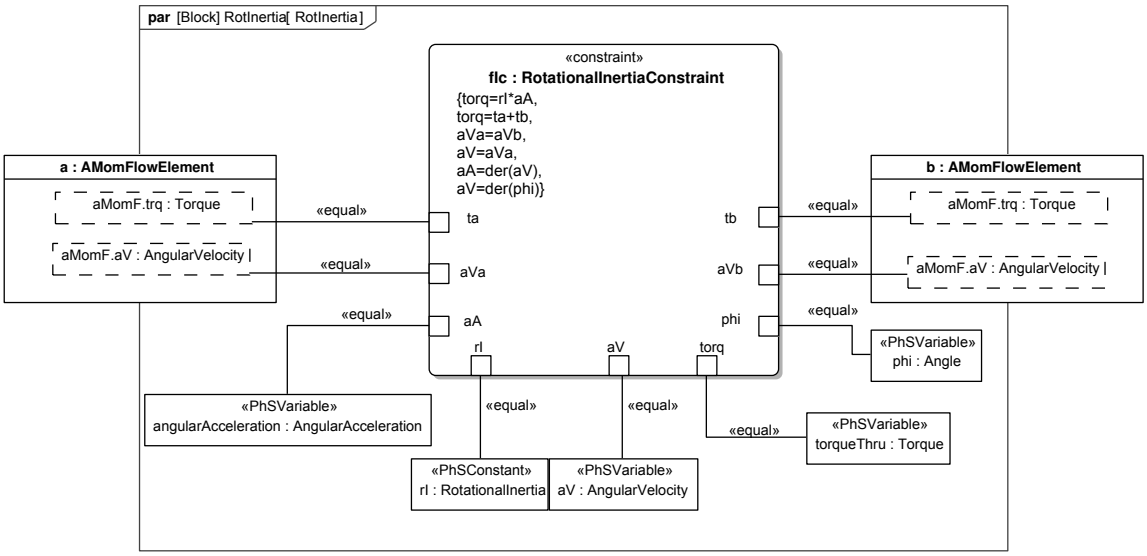

Fig. 32. Rotational inertia parametric diagram

RotInertia is applied in Figure 26, where it is used to model flywheels i_1,i_2, and the inertia of gearwheels i_4 and i_5. Inertia i_3 connects with a damper on the left side and a rotation sensor on the left side. It would be equivalent to connect the rotation sensor to the left port, however, by connecting to the right port, we can better capture that the rotation sensor is on the right of the component. RotInertia has a parameter I which defines rotational inertia in units of kg*m^2.

The library in Figure 28 provides two elements for transforming angular momentum, IdealFixedGearBox and IdealGearR2T. IdealFixedGearbox(parametric diagram shown in Figure 33) models a ideal rotary transformer that permits transformation of rotation with unlimited rotation angle. It assumes no damping, backlash, elasticity, and that the gearbox itself may not rotate. IdealFixedGearbox has two ports, rotA and rotB and PhSConstant gearRatio is the ratio of the angular velocity of rotA to rotB. When it is used to model a pair of external

gears,[13] as with G_1 and G_2 in Figure 26, gearRatio should be a negative number to ensure the gears will rotate in opposite directions. This component will not necessarily convserve angular momentum because it is implicity fixed. Some angular momentum flows to or from the rotational ground. Some elasticity, damping, and inertia can be approximated by connecting the elements for those effects. Figure 27 does this for the inertia of each gear, modeled with RotInertia blocks connected on either side of the IdealFixedGearbox.

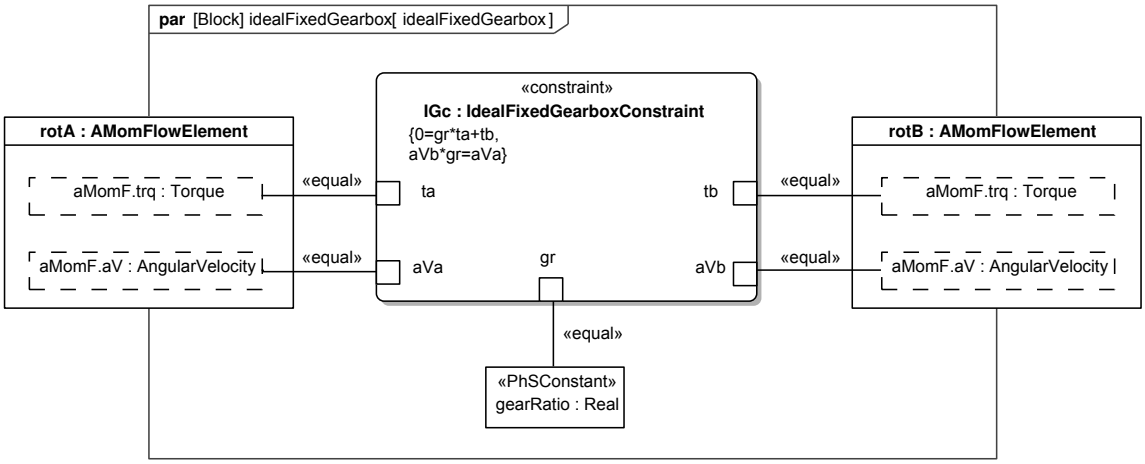

Fig. 33. Ideal fixed gearbox parametric diagram

IdealGearR2T(parametric diagram shown in Figure 33) enables transforming angular momentum to translational momentum without losses, elasticity, backlash in a manner similar to how a gear rack converts rotary motion to linear motion and vice versa. IdealGearR2T has a parameter ratio which defines the ratio of between angular velocity to linear velocity in units of rad/m. This component can be used to represent an ideal gear rack or a belt/screw drive in which elasticity and losses are ignored.

[13] Gears with teeth on the outside

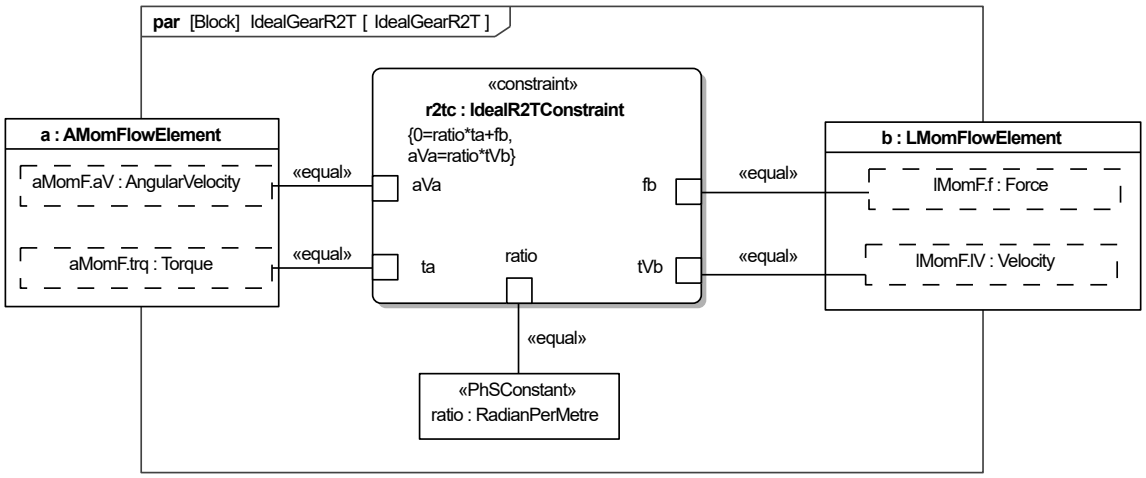

Fig. 34. Ideal rotary to translational converter

Figure 27 models gears G_1 and G_2 as a single IdealFixedGearbox with a gear ratio of -2, which is negative to capture that the gear pair reverses rotation direction between the shafts connected to it. If one were to reverse the way rotA and rotB were connected in the diagram, in order for the model to be equivalent the gear ratio would need to be -1/2 because gear ratio is defined as rotA angular velocity to rotB angular velocity.

The library in Figure 28 has source and sensing elements for providing or sensing quantities derived from ports. This library has two source elements for providing either angular velocity or torque. Both source elements take in a real signal and output the corresponding controlled variable on an angular momentum port.

FixedTorqueSource(parametric diagram shown in Figure 35) is a block which connects to an angular momentum port and applies a torque proportional to an input real signal. It is considered fixed as it is implicitly fixed to the ground. As opposed to being free floating and applying a torque between two components it applies a torque between a fixed ground and anything attached to its port. Fixed torque source has a parameter signal2Torque which specifies the conversion factor between the unitless real signal and torque output which is in units of Nm. A FixedTorqueSource is added to the system shown in Figure 26 and Figure 27 to model the torque boundary condition. FixedTorqueSource is connected to a block which outputs a sine wave real signal and to the left port of i_1.

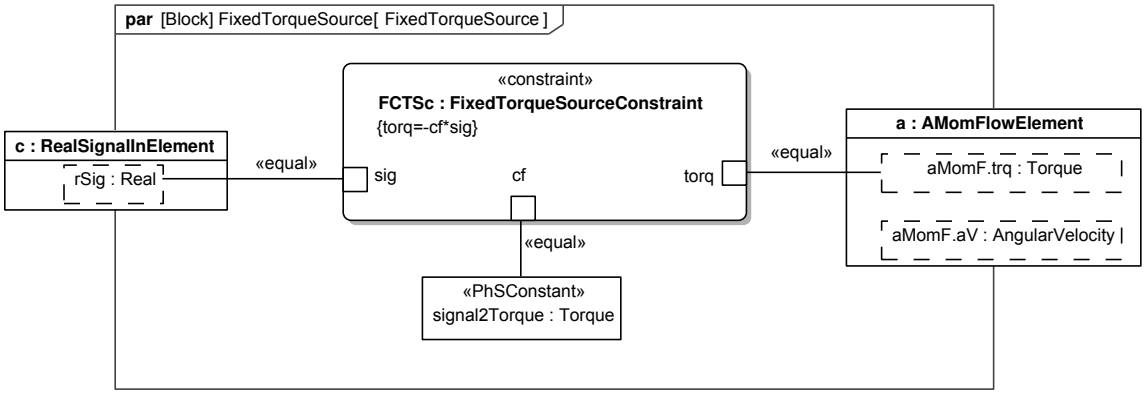

Fig. 35. Fixed torque source parametric diagram

FixedaVSource provides angular velocity at a port which is proportional to an input real signal. It is fixed in the same manner as FixedTorqueSource is fixed. It applies an angular velocity with respect to a non rotating frame. As shown in Figure 36 FixedaVSource, defined in Figure 36, has a parameter signal2AngVelocity which is a conversion factor between unitless signal and angular velocity, it is given in units of rad/s.

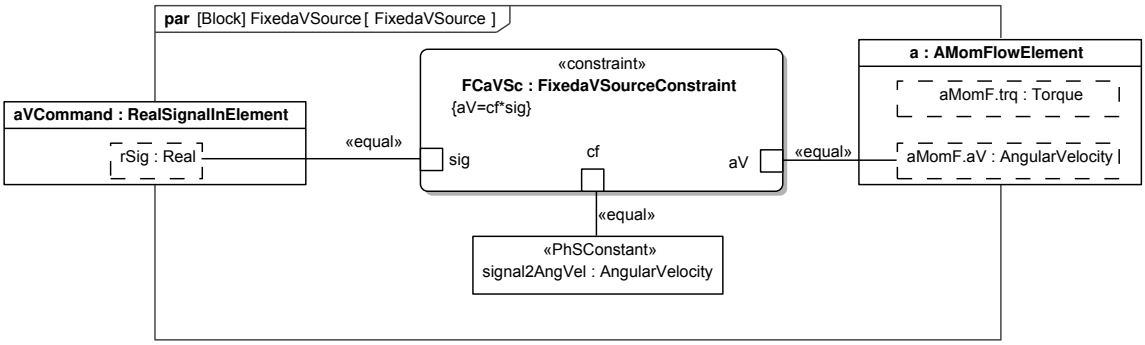

Fig. 36. Fixed angular velocity source parametric diagram

In contrast to source elements there are also sensing elements. Sensing elements connect to a port and output a real signal proportional to a quantity sensed on that port. The library in Figure 28 includes two sensing elements AngularVelocitySensor and AngleSensor.

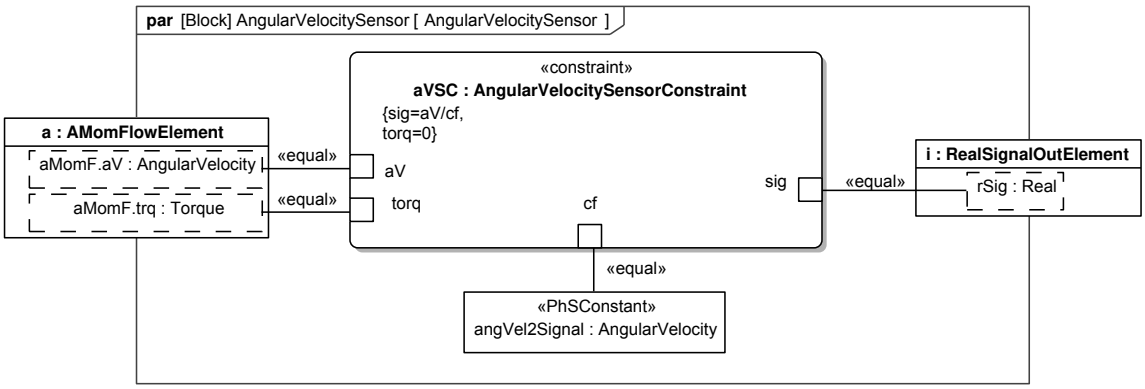

Fig. 37. Angular velocity sensor parametric diagram

AngularVelocitySensor(parametric diagram shown in Figure 37) outputs a real signal proportional to the angular velocity on the attached angular momentum port. The parameter angVel2Signal gives the inverse of the conversion factor between angular velocity and signal and is given in units of rad/s.

AngleSensor, defined in Figure 38, outputs a real signal proportional to angle. It must be noted that the angle sensor works by integrating velocity and requires that an initial angle be specified. This sensor has a parameter angle2Signal which specifies the inverse of the conversion factor between angle and signal, and is given in units of rads. There is also a generalization of rotational linear spring, RotLinearSpringWithDisplacementSensor which outputs a real signal proportional to the displacement of the spring. RotLinearSpringWithDisplacementSensor has a parameter displacement2Signal which specifies the inverse conversion factor between realSignal and displacement. This is given in units of rads.

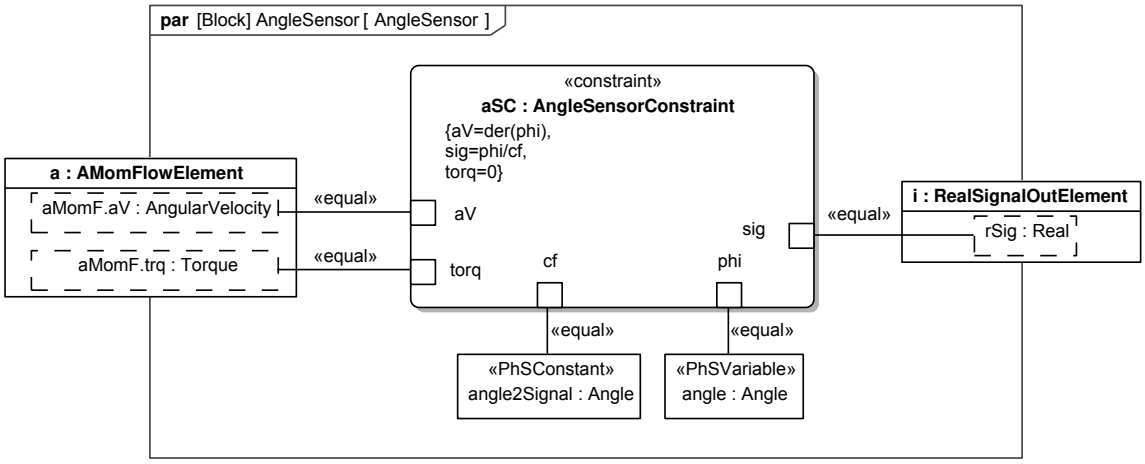

Fig. 38. Angle sensor parametric diagram

4.3. Entropy (Heat) Transfer Library

The entropy (heat) flow library enables SysPhS modeling of one dimensional movement of entropy (heat). It includes 1D models of all basic heat transfer processes including conduction, radiation, and convection.

Figure 39 shows an example thermal system. It is the heated bed of a Fused Deposition Modeling (FDM) 3D printer, see Section 5.2.4. Heat flows from a heater on the bottom to an insulating layer above it (glass), which stores some heat and sends the rest through its top surface to air of constant temperature. The heater and surface are treated as having uniform temperatures T_{heater} and T_{surf}, respectively, with the surface temperature measured by a sensor on the surface of the bed. The air is assumed to be a large thermal reservoir with constant temperature T_{air}. The heated bed insulation and heater are assumed to have some heat capacity, but with negligible heat loss through the sides and bottom of the bed. While the uniform temperature assumption for the heater and insulation is not necessarily realistic, it offers a simple approximation useful for estimating the warm up time and a rough simulation of whether the controller will keep the surface temperature within acceptable bounds. The effects of changes in bed temperature due to addition of hot filament have been excluded. Simulation of this system is covered in Section 5.2.

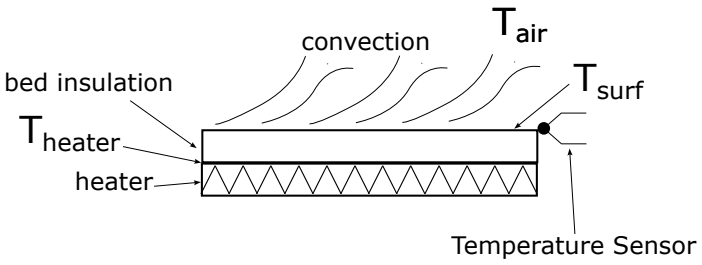

Fig. 39. An example thermal system

The system in Figure 39 is modeled with SysPhS in Figure 40, an IBD connecting components defined by the thermal library BDD in Figure 41 (see Section 2.2 about these kind of diagrams). Figure 40 refers to blocks in library by their names, appearing to the right of the colon at the top of each larger rectangle. The role each block plays in the system appears to the left of the colon in each, following the labels in Figure 40.

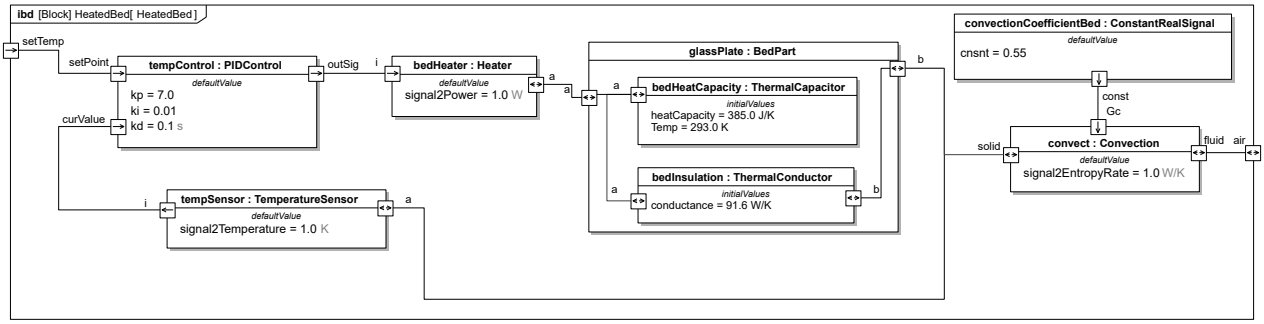

Fig. 40. Figure 39 modeled in SysPhS with initial conditions and ports for control signals and coupling to environment

Figure 40 connects thermal components at their physical interaction ports, notated by small rectangles with bidirectional arrows inside. Flows of entropy through these ports are described by entropy flow rate and temperature, which are entropy's rate of flow and potential to flow, respectively (conserved and non-conserved variables, respectively).[14] Temperature is the potential to flow of entropy, since two objects at the same temperature cannot exchange entropy.[15]

The heated bed includes a 214 x 214 mm glass pane that is 4 mm thick. The thermal conductivity of glass is about 0.8 W/(m*K) and thermal conductance along a rectangular section is $g = k * A/L$, where g is the thermal conductance, k is the thermal conductivity of the material the section is made from, L is the thickness, and A is the area of the section, resulting in G= $0.8W/(m*K)*(0.214m)^2/0.004m = 91.592W/k$, the conductance of the ThermalConductor block bed insulation. It is assumed the heat capacity of the heated bed is largely due to the heat capacity of the glass. The specific heat of glass is taken as 0.84 J/(g*K) and the density of glass is 2500 kg/m^3, giving a thermal capacity of 385 J/K. Assuming a constant convection coefficient of 12, which is reasonable for a horizontal plate in free convection, the convection conductance is 0.55 W/K.

Entropy flow ports use the variables of entropy flow rate and temperature, which multiply to power. Entropy flow can be more general than heat transfer. This convention has been shown to be useful in the analysis of systems including heat engines and heat pumps [17]. One should keep in mind that while entropy is the analog of electric charge and momentum, it is often not conserved in heat transfer. All heat transfer elements in this library do not conserve entropy. So the entropy flow rate out of each element will exceed the entropy flow rate into each element for any finite temperature difference. In heat conduction with a finite temperature difference entropy is produced. So in the elements which model this, Thermal Conductor, ConductiveBar, Convection, and Radiation, with any temperature difference,

[14] Heat transfer is typically modeled with power (energy rate) and temperature, such as the Modelica Standard Library for Heat Transfer [16]. SysPhS uses entropy rate and temperature to enable flows between components to the taken as energy flows, with rate of energy flow (power) being the product of the variables.

[15] This assumes no mass flow, which this library does not currently address.

the entropy flow rate out will exceed the entropy flow rate in because these elements generate entropy. Although between ports, entropy flow is always conserved. Non-zero entropy flow rates can happen when temperature on the same component differs between two ports.

Figure 41 defines the thermal library introduced by this paper and used in Figure 40. It includes ThermalTwoPort, which has two entropy ports and is specialized to describe elements which transfer heat between the ports. Figure 42 defines the constraint blocks referenced by constraint properties in the library, appearing in compartments of each thermal block (see Section 2.2 about constraint modeling in SysML).

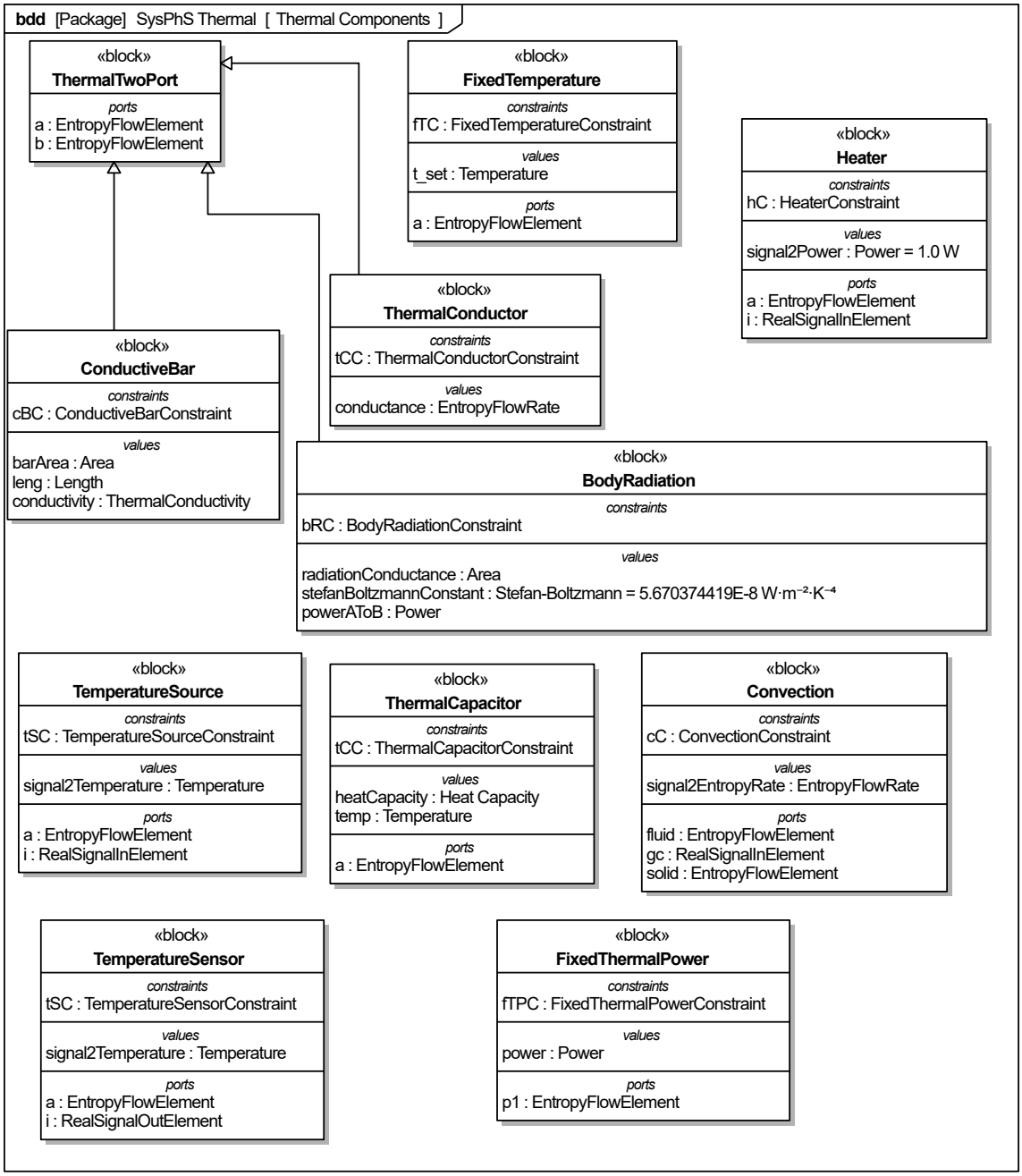

Fig. 41. Entropy (heat) transfer library

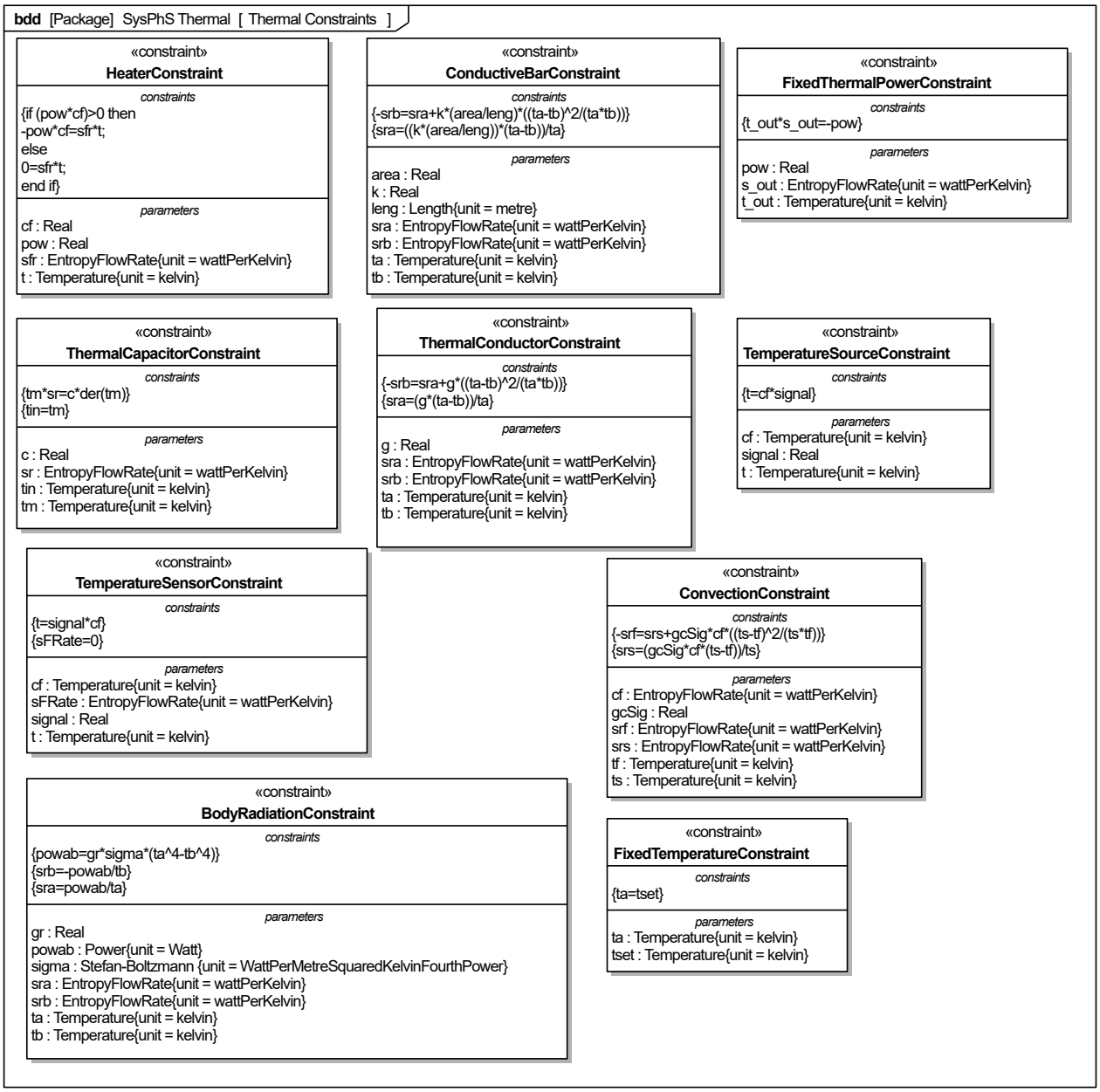

Fig. 42. Constraints for the thermal library

The rest of this section covers the components in Figure 41 and parametric diagrams that bind their properties to constraint parameters in Figure 42 (see 2.2 about these kind of diagrams).

The ThermalCapacitor block in Figure 41 models an component that can store heat, as expressed in ThermalCapacitorConstraint in Figure 42. Figure 43 shows the parametric diagram for ThermalCapacitor that binds its properties to parameters from ThermalCapacitorConstraint. The component has a variable for its temperature, assumed to be the same

throughout, and a parameter for heat capacity in units of J/K[16]. Heat capacity of an object is the specific heat of its material times its mass. The heat capacity of the bed in Figure 40, bedHeatCapacity, is modeled with a ThermalCapacitor.

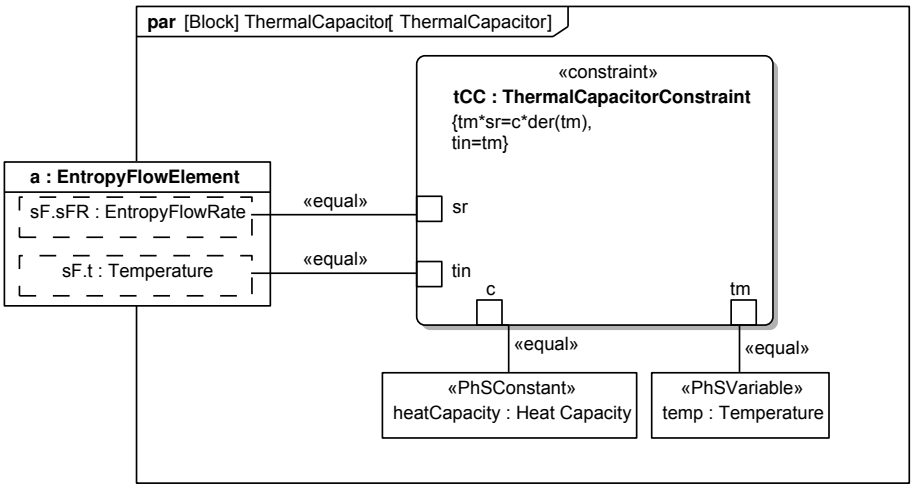

Fig. 43. Thermal capacitor

ThermalConductor in Figure 41 models a component that conducts heat without storing or generating any, as expressed in ThermalConductorConstraint in Figure 42. Figure 44 shows the parametric diagram for thermalConductor that binds its properties to parameters from ThermalConductorConstraint. ThermalConductor is based on the description of thermal conductors given in [14] and [17], but reformulated in SysPhS rather than bond graphs. ThermalConductor has a parameter conductivity for the thermal conductance of the connection given in W/K, a measure of entropy flow rate. Calculating the thermal conductance depends on the properties of the connection. For heat conduction along the length of an object with constant area, thermal conductance may be calculated as $G = k * A/L$. Where k is the thermal conductivity of the material in W/(m*K), A is area of the cross section in m^2 and L is the length of the section. The block ConductiveBar acts the same as ThermalConductor, except it allows one to specify the thermal conductivity, area, and length rather than a thermal conductance. Heat transfer through the heated bed insulation in Figure 40, bedInsulation, the thermal resistance of the heated bed insulation, is modeled with ThermalConductor. It should be noted that bedInsulation and bedHeatCapacity are single physical object which is modeled with the block glassPlate.

[16]This is the same units as entropy, but heat capacity is a different physical phenomena than entropy. Change in entropy for a reversible process is k$S = \frac{\delta Q}{T}$ whereas heat capacity is defined as $C = \frac{\Delta Q}{\Delta T}$

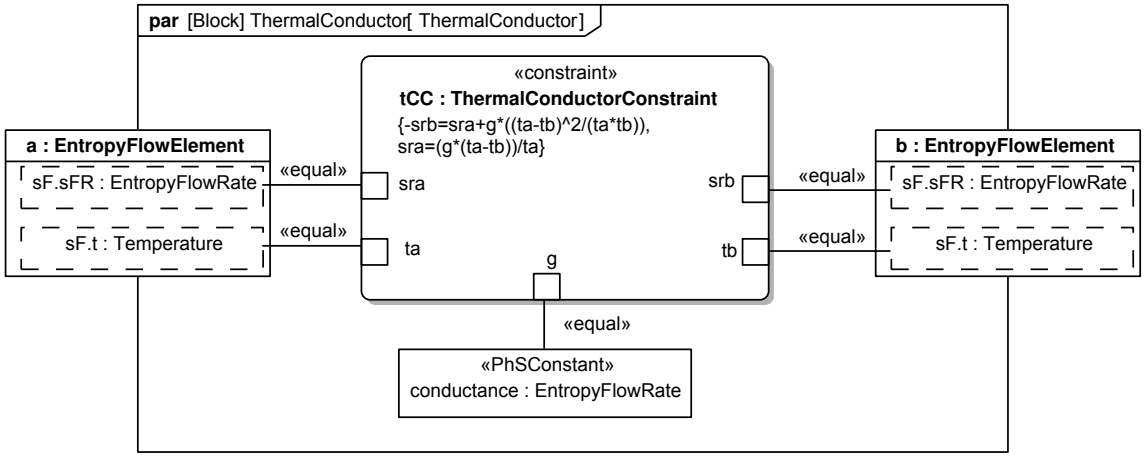

Fig. 44. Thermal conductor parametric diagram

Convection in Figure 41 is for modeling heat transfer between a solid and a fluid. Figure 45 shows its parametric diagram that binds its properties to parameters from Convection-Constraint. This is modeled as if the convection medium were a thermal conductor that has a conductance controllable by a realSignal, to approximate convection from a solid object by changing the conductivity according to some chosen function of fluid flow. This conductivity is in W/K and may be calculated as $h * A$ for simple geometry and heat transfer scenarios, where A is the area in contact with the fluid and h is the convection coefficient, which depends on the fluid, flow properties, geometry, and scenario in question. Heat transfer from the surface of the heated bed to the air, bedConvection in Figure 40, Section 4.3, is modeled with Convection. Its convection conductivity is determined by the constant real signal convectionCoefficientBed input. The surface area of the bed is constant, and the free convection (in which fluid motion is not driven by any external source such as a fan or pump) on the bed is assumed to be constant so the convective conductivity is constant. This block is analogous to the convection element in the Heat Transfer library of the Modelica Standard libraries [16]. The convection model does not include thermal fluid effects and or address temperature change of a fluid as it moves through a pipe.

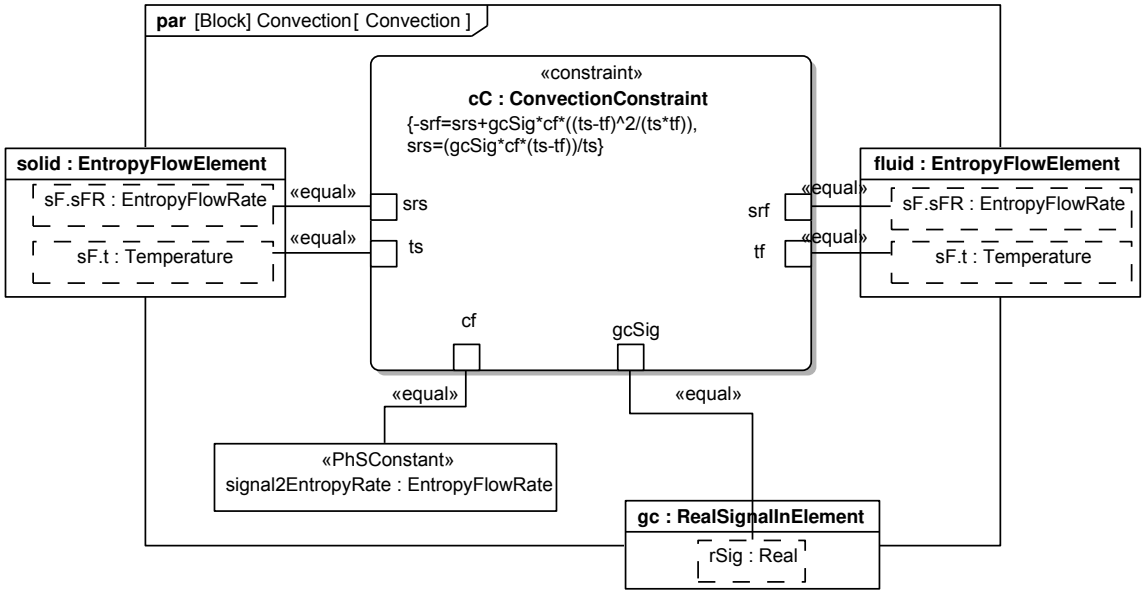

Fig. 45. Convection parametric diagram

BodyRadiation in Figure 41 is for modeling heat transfer via radiation between the surfaces of two bodies due to radiation. Figure 46 shows the parametric diagram for BodyRadiation that binds its properties to parameters from BodyRadiationConstraint. This has a parameter radiationConductance which specifies the radiation conductance in units of m^2. The value of radiation conductance depends on the emissivities, area, and geometry of the bodies in which radiation transfer occurs. BodyRadiation has a PhSConstant stefanBoltzmannConstant with an initial value of 5.670374419E-8 $\mathrm{W{\cdot}m^{-2}{\cdot}K^{-4}}$, which should not be changed in the library or its specializations.

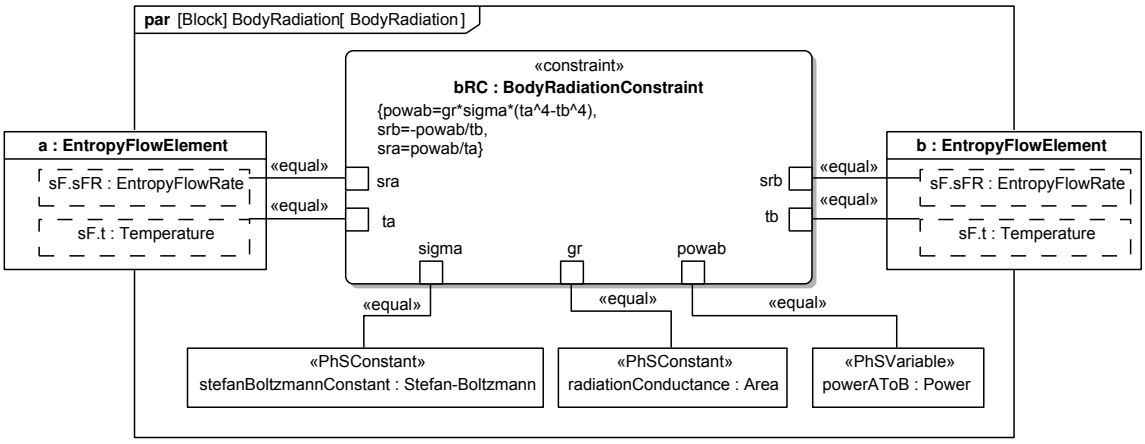

Fig. 46. Body radiation parametric diagram

The only thermal sensing element in the library is a temperature sensor, since entropy flow sensors do not appear to exist. It outputs a real signal proportional to the temperature on

its entropy port. The PhSConstant signal2Temperature (shown in Figure 47) gives the conversion factor between the real number output and the temperature measured. As sources, a fixed temperature source, a controllable temperature source, a fixed thermal power, and a model of a heater are used. There is no entropy flow source even though entropy flow is a power conjugate variable because it is not clear what an entropy flow source would correspond to in the physical world .

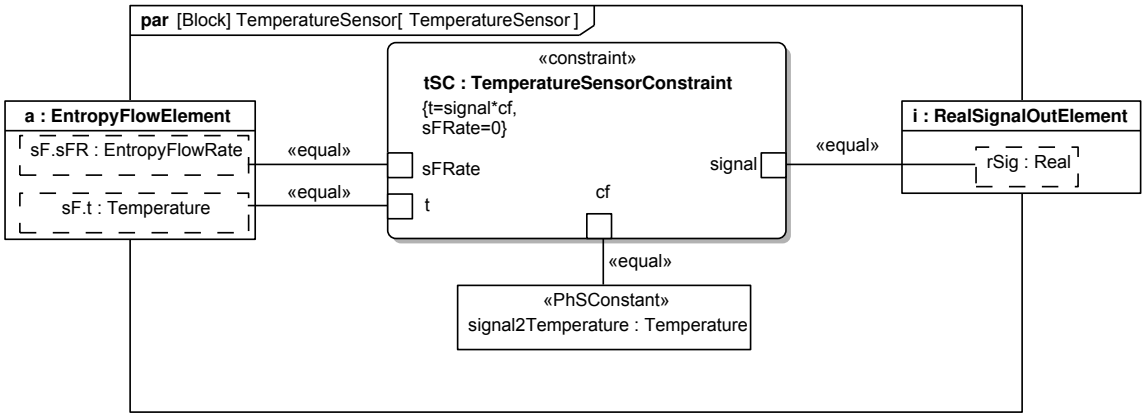

Fig. 47. Temperature sensor parametric diagram

Heater in Figure 41 is a heat source controlled by a real signal. Figure 48 shows the parametric diagram for Heater, which binds its properties to parameters from HeaterConstraint. It takes in a real signal and outputs a heat flux out of this part, expressed as a negative flow rate, proportional to the real signal, as shown in heaterConstraint. If the real signal is less than zero, no power is output, preventing the heater from extracting heat from the system rather than providing it. The heater in Figure 39, bedHeater, is modeled in Figure 40 with Heater.

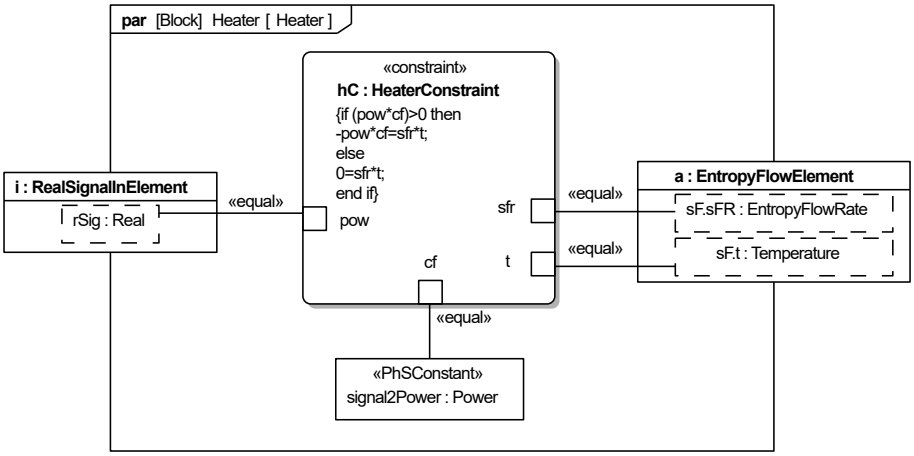

Fig. 48. Heater parametric diagram

FixedThermalPower in Figure 41 models thermal power boundary conditions that do not change with time, specified by the PhSConstant power in Figure 49, a parametric diagram that binds its properties to parameters from FixedThermalConstraint. The constant can be positive, for entropy (heat) flow out of the component, or negative, for flow into the component.

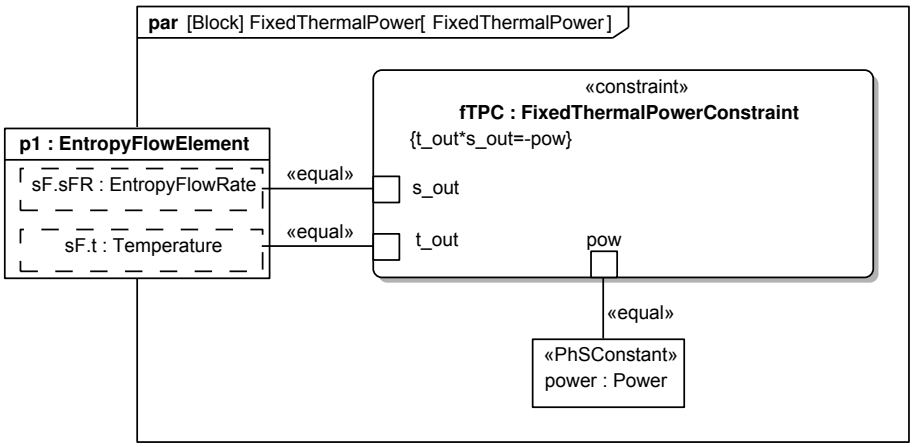

Fig. 49. Fixed thermal power parametric diagram

The library in Figure 41 has two components that approximate a boundary condition where change in temperature due to heat addition or subtraction is negligible. FixedTemperature provides a constant temperature boundary condition, while TemperatureSource provides one that is proportional to an input real signal, as shown in Figures 51 and 50, respectively.The conversion factor between the input real signal and temperature output on temperature source is defined by the PhSConstant signal2Temperature. TemperatureSource is useful for modeling cases such as environmental temperature variation due to a day night cycle. The real signal could be constant, but FixedTemperature does this without requiring an input, only the PhSConstant t_set to specify the temperature.

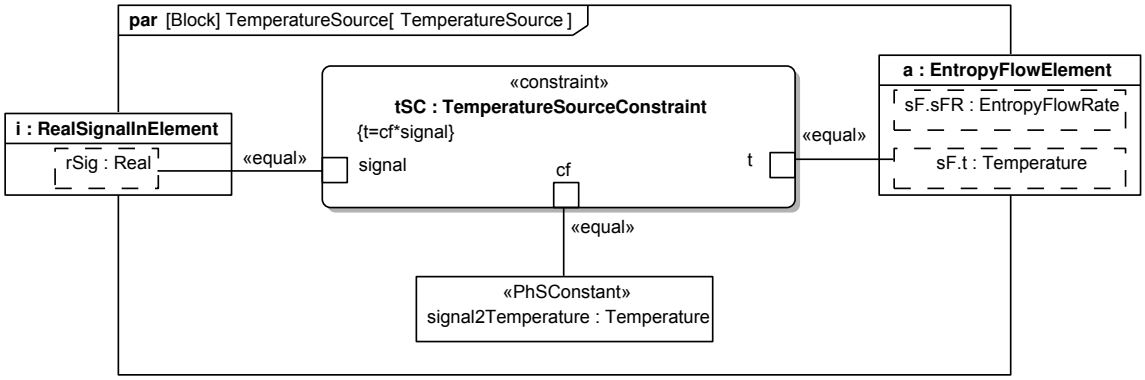

Fig. 50. Parametric diagram for temperature source

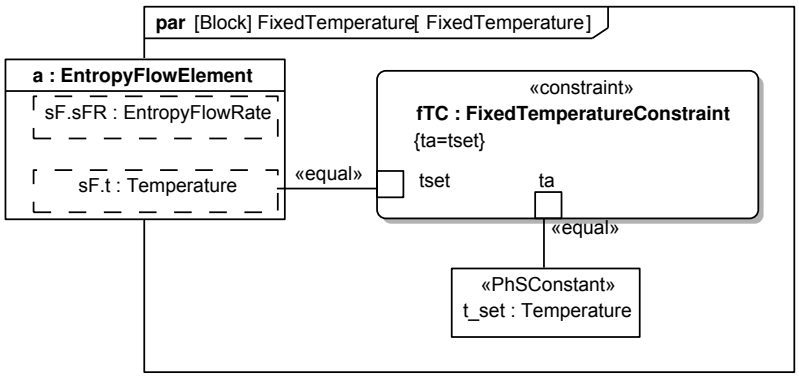

Fig. 51. Parametric diagram for fixed temperature

5. Manufacturing Examples

This section applies the model libraries in Section 4 to example manufacturing systems, translates them to simulation platforms on an open implementation of SysPhS [7][8], and presents simulation results. They are a collaborative robot, 3D printer, and polishing machine, in Sections 5.1, 5.2, 5.3, respectively. The collaborative robot model uses the rotational library, the 3D printer applies the translational and thermal libraries, and the polishing machine example uses the translational and rotational libraries. See Section 6 for more information about the simulation tools used in this section.

5.1. Weight Compensating Robot

Collaborative robots, or robots which work with people, are becoming more common in manufacturing. One task they perform is helping people handle heavy objects, such as positioning a large tool during manufacturing or guiding an object through a complex path during assembly, by compensating for its weight. Operators move or rotate an object while a robot holds it up, assisting over a more complicated path than is possible with passive devices, such as constant force springs or combinations of linkages and springs. Operators can adjust the compensating force or remove it completely if an object is no longer held. Modeling and simulation help determine whether designs for these robots will provide the necessary control response, minimizing forces on the operator and ensuring safe robot behavior.

The example of weight compensation in this section is a robot arm with a single fixed rotary joint, with the object to be moved attached on the end, as illustrated in Figure 52. This system may be modeled as a pendulum. The robot senses the arm's angle and applies a counter-torque τ to the arm, counter-acting the weight of the object mg, but still allowing the operator to move it up and down, as if it were weightless.

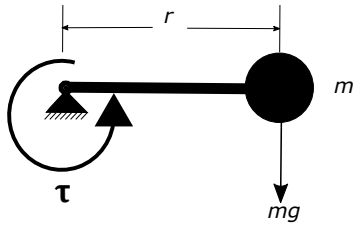

Fig. 52. Weight compensating robot example

The system in Figure 52 is modeled with SysPhS in Figure 53, an IBD connecting components defined in this section. The robot includes an actuator for the arm (sea0) pdfcommentComment: why the difference in numbering notation for sea0 and pend1? Reply: Will fix before publication., directed to compensate for weight by a controller (gravityCompensationController) depending on the current angle of the arm/pendulum (pend1) as measured by a sensor attached to it. The operator connects to the robot to move the arm around.

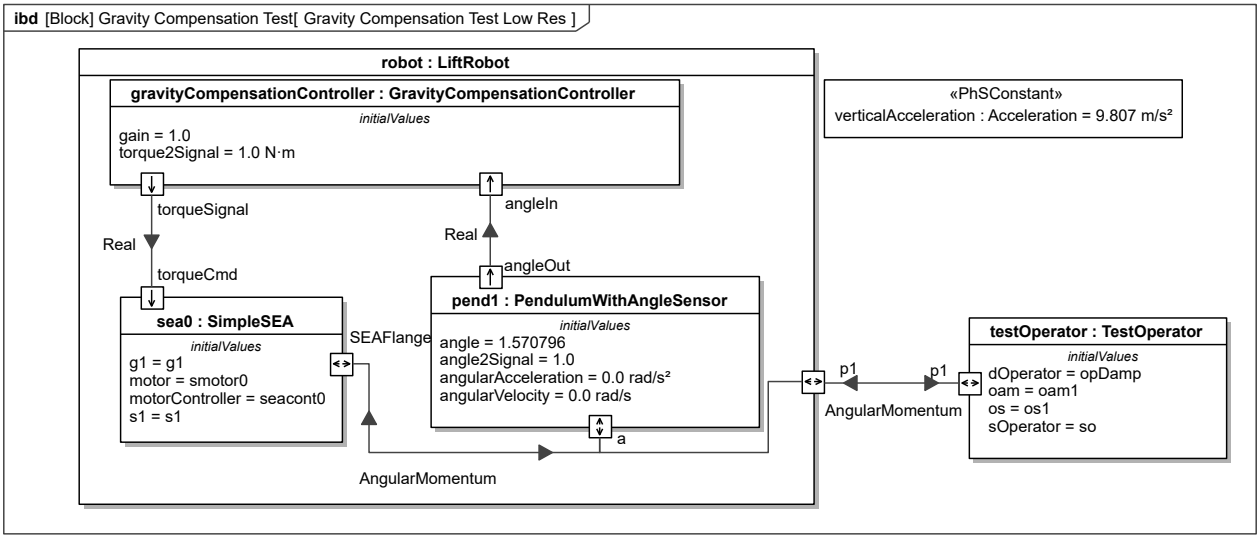

Fig. 53. Figure 52 modeled in SysPhS

5.1.1. Pendulum

Pendulum in Figure 54 models a point mass subject to a continuous vertical acceleration and attached at some distance from a rotational center, without any damping forces. It can be coupled to other components using its angular momentum flow port. It has a parameter cmgDist, giving the distance of the point mass from the center of rotation in units of m, and a parameter mass for the point mass.

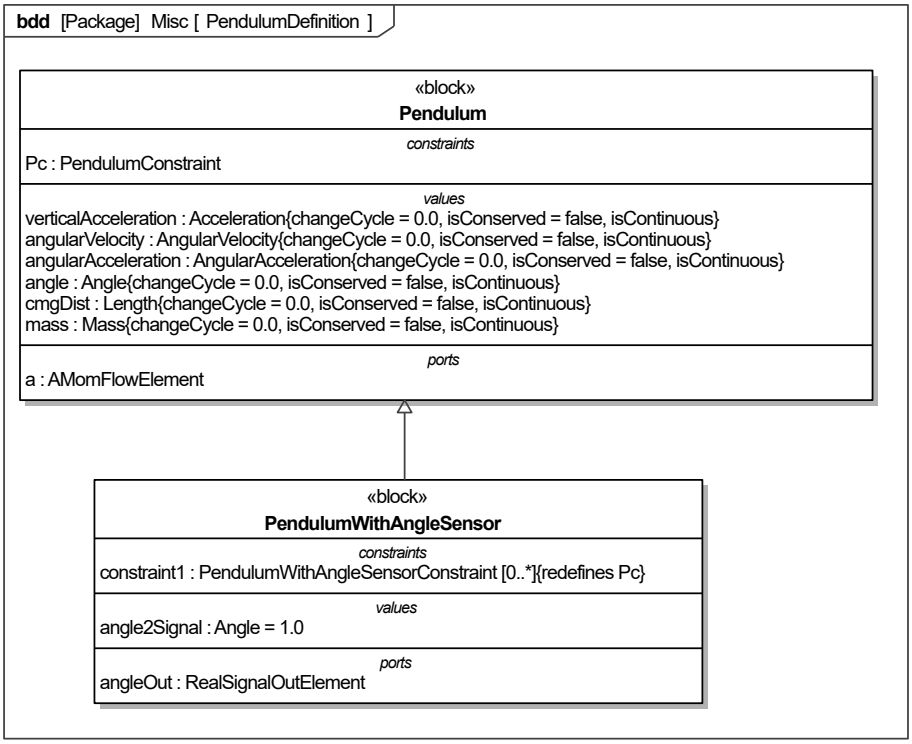

Fig. 54. Pendulum and PendulumWithAngleSensor

Figure 55 shows the pendulum's parametric diagram. The rotational inertia of a point mass pendulum is calculated by $mass * cmgDist^2$. The angle of the pendulum is taken to be zero when it is at rest.

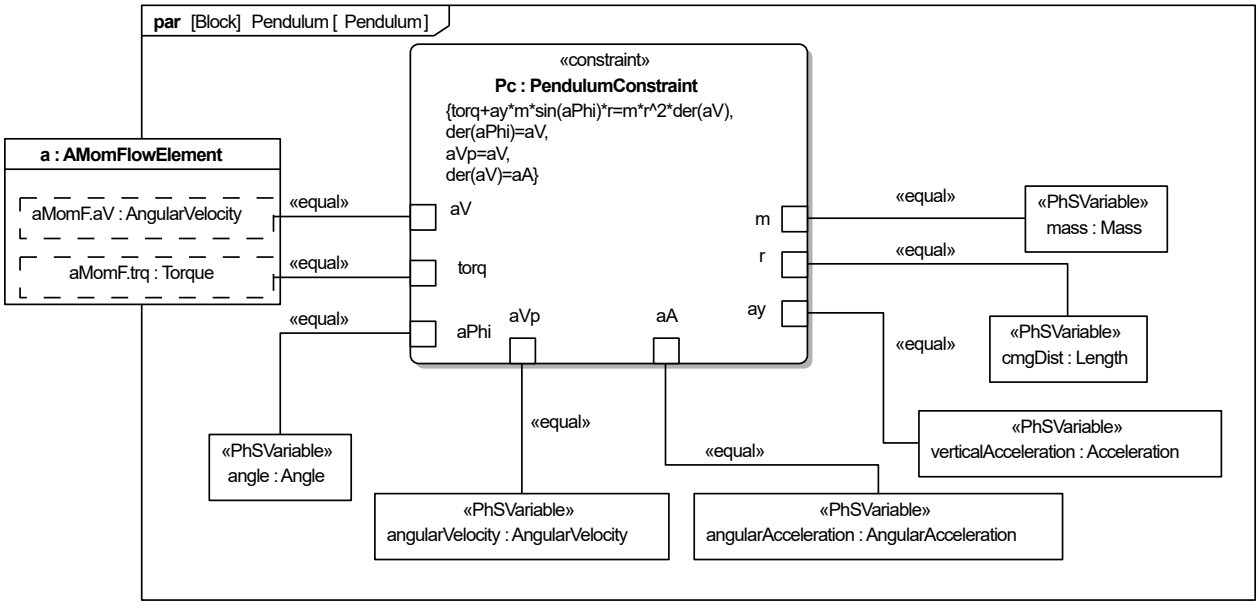

Fig. 55. Pendulum parametric diagram

The PendulumWithAngleSensor used in Figure 53 is defined in Figure 54 as a specialization of a Pendulum block which has a sensor built into it to measure angle. PendulumWithAngleSensor, parametric diagram shown in Figure 56, outputs this angle on a real signal port.

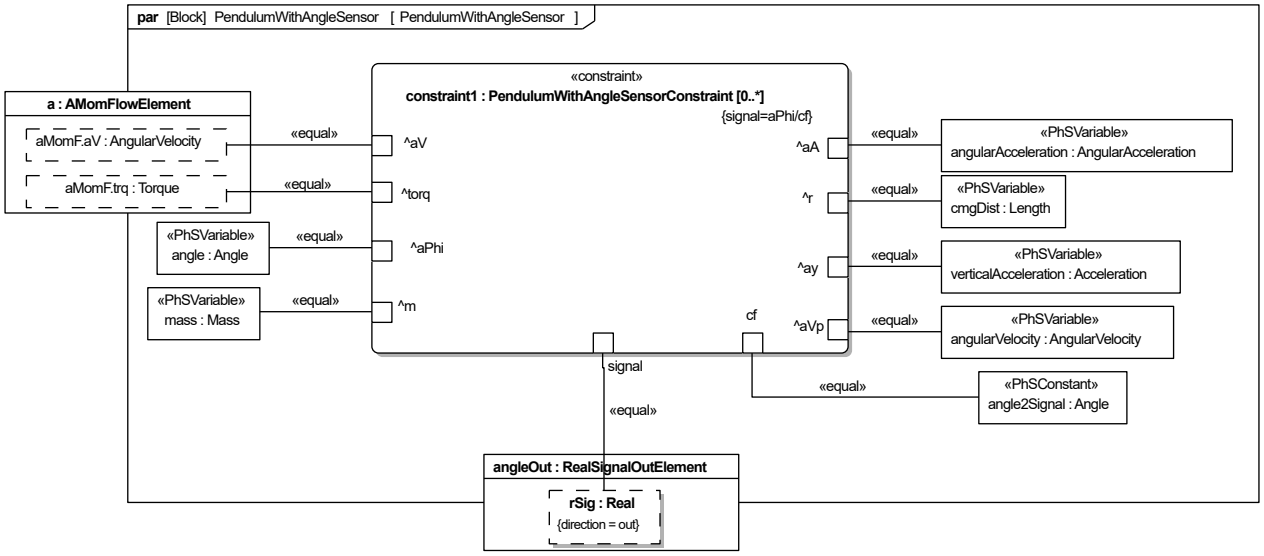

Fig. 56. PendulumWithAngleSensor parametric diagram

5.1.2. Series Elastic Actuator

The actuator sea0 in Figure 53 is a series elastic actuator, a type of actuator where a spring is placed between the speed reduction transmission and the load being driven, as shown in Figure 57 [18]. By measuring displacement of the spring the force or torque the actuator applies can be measured and adjusted rapidly by changing the position or angle of the transmission. The spring also decouples the inertia of the motor and transmission from the load, decreasing the chance of injury should a robot using these actuators collide with a person. In addition, the spring also helps protect the high reduction transmission from impacts that might damage it.

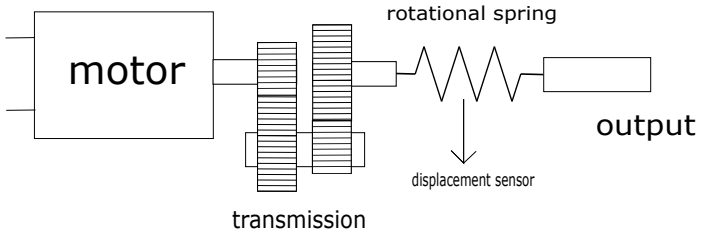

Fig. 57. Series elastic actuator

The actuator in Figure 57 is modeled with SysPhS in Figure 58, an IBD connecting components defined in Section 4.2 and this one. It includes an electric motor, gearbox, and rotational spring with a displacement sensor connected in series. The series elastic actuator takes in a desired output torque value, measures displacement of the spring to estimate current torque, and a controller sends a signal to the motor, attempting to achieve the desired torque value.

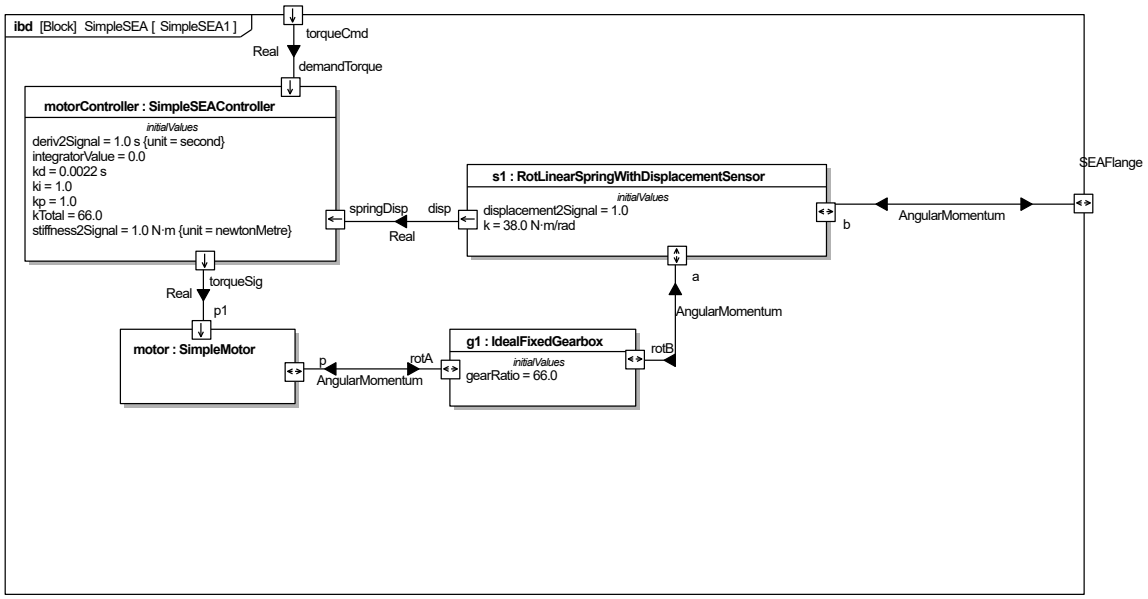

Fig. 58. Figure 57 modeled in SysPhS

Figure 59 shows a parametric diagram for the series elastic actuator controller modeled in Figure 58. It implements proportional–integral–derivative (PID) control of applied torque, comparing the desired and current torques, estimated from the current displacement of the spring, to calculate the signal that should be sent to the motor. The PID controller output is multiplied by a value proportional to the gear ratio, which is output to the motor. [17]

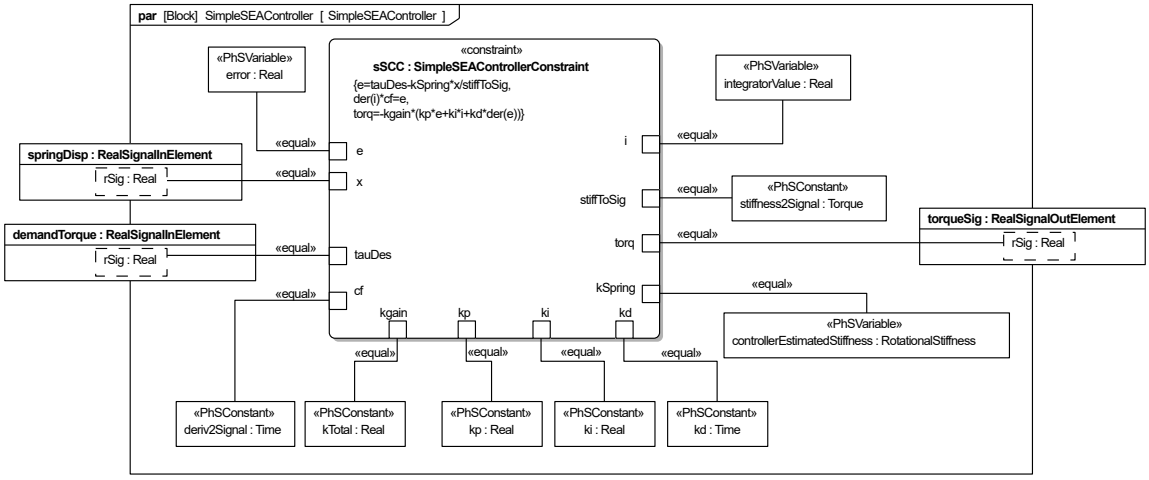

Fig. 59. Parametric diagram for SimpleSEAController in Figure 58

The electric motor in Figure 58 is modeled as a controllable torque source connected to a rotational inertia connected to an output with a rotational damper connected to ground, as shown in Figure 60. The rotational damper models the motor's internal friction, enabling it to reach a constant speed rather then accelerating forever when a constant input signal is applied.

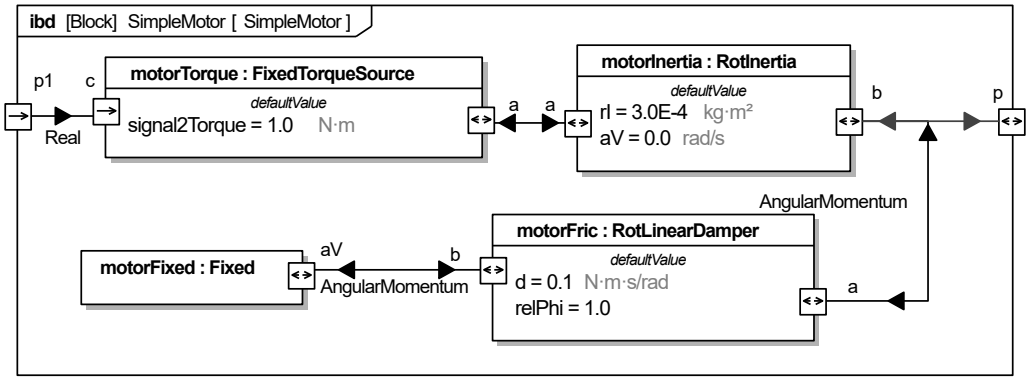

Fig. 60. SimpleMotor used in Figure 58

[17] In practice, more complicated controllers are often employed to control series elastic actuators [19].

5.1.3. Gravity Compensation

In this model the series elastic actuator is used as a means to control torque. In order to compensate for the weight at the end of the arm we need to apply torque τ given by Equation 1:

$$\tau = m * g * r * sin\theta \tag{1}$$

where m is the point mass at the end of the arm, g is acceleration due to gravity, r is distance the point mass is from the center of rotation, and θ is angle of the arm.

The block GravityCompensationController implements this equation, taking in the current angle and calculating the torque to be applied, as shown in Figure 61. The torque to be applied is multiplied by a PhSConstant gain, enabling one to apply more or less torque than necessary to balance the arm, such as applying slightly more torque to compensate for friction in the actuator.

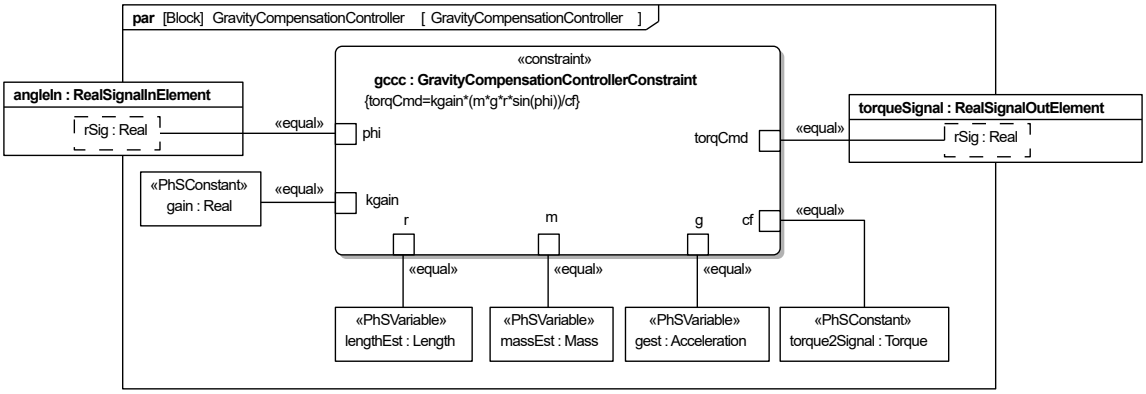

Fig. 61. Gravity compensation controller

The PhSVariables lengthEst, massEst, and gest in Figure 61 are the values for the arm's center of gravity, mass, and acceleration due to gravity. In this example, these have the same values as the arm, specified with binding connectors to PhsConstants in the test model, as shown in Figure 62, enabling the same PhSConstant values to be shared between multiple parts and changed easily. For example, the arm and controller vertical accelerations are required to have the same values as verticalAcceleration, the vertical acceleration of the operating environment of the arm. The PhSConstant pendulumLength in this environment specifies the distance to the center of gravity of the arm and pendulumMass gives the mass of the arm and payload.

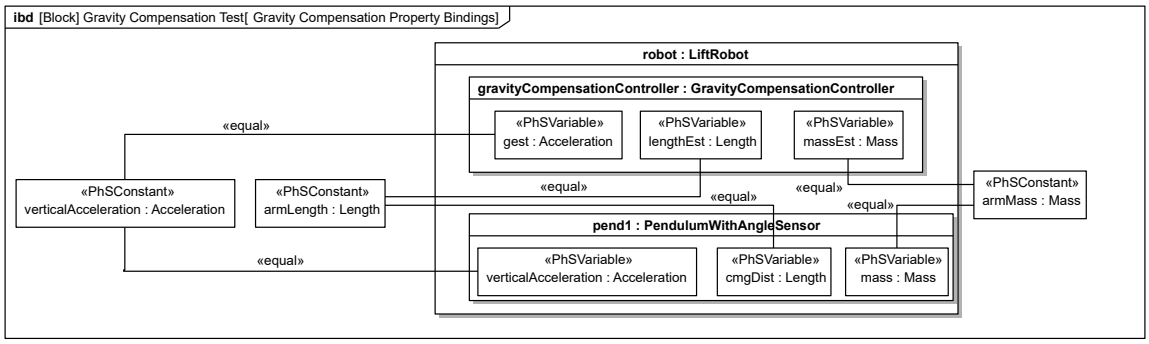

Fig. 62. Test conditions specified with binding connectors

5.1.4. Operator

The operator is modeled as an angular velocity source connected to the arm with a spring-damper system modeling the compliance between the operator and the robot, as shown in Figure 63. The connection between the operator and the robot will not be completely rigid, so some compliance is necessary. During the simulation the operator remains motionless for a set time period before moving at a constant set velocity.

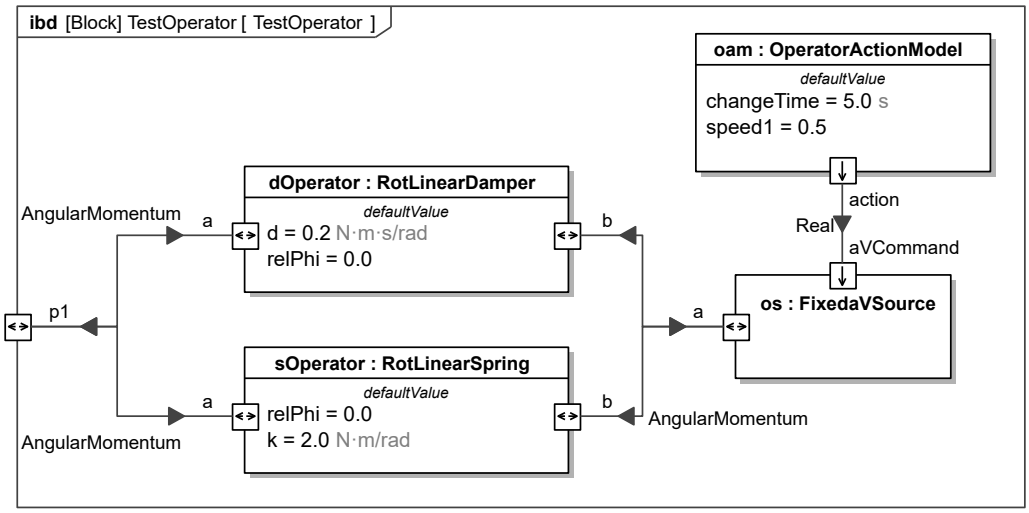

Fig. 63. Diagram of operator model

The block OperatorActionModel, shown in Figure 64, is used to model the operator remaining motionless until a set time and then proceeding at a constant speed. The block outputs a zero real signal on port 'action' until a time defined by the PhSConstant 'changeTime,' after which a real signal defined by the PhSConstant 'speed1' is output.

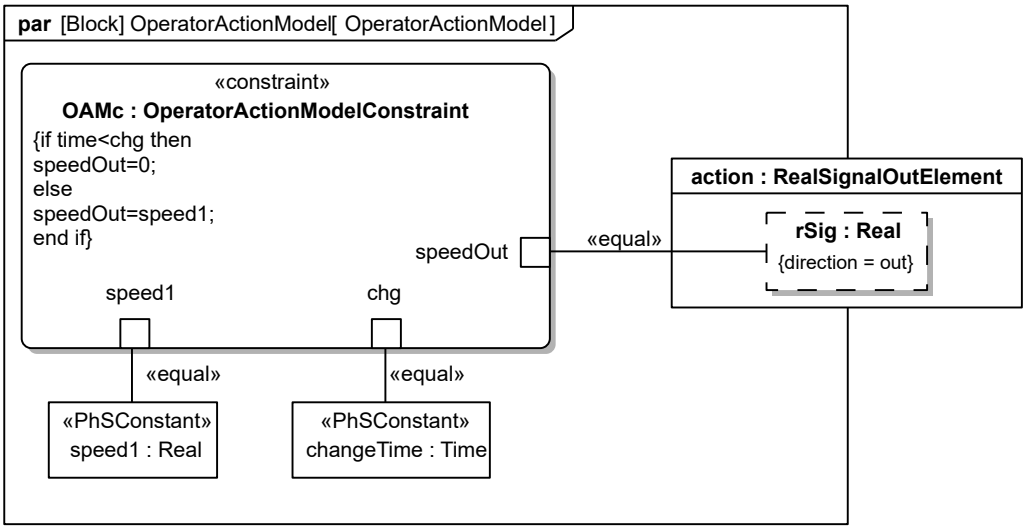

Fig. 64. Parametric diagram of operation action model

Figure 65 adds detail to Figure 53, showing the total system model for Figure 52, including operating environment. In this example, the torque on the operator determines whether the collaborative robot is operating safely and correctly by whether it is within safe bounds. This example does not define safe bounds, however, it is desirable that the force on the operator be as low as possible during operation, since the intended function of this device is to reduce torque on the operator so they can more easily move the load around.

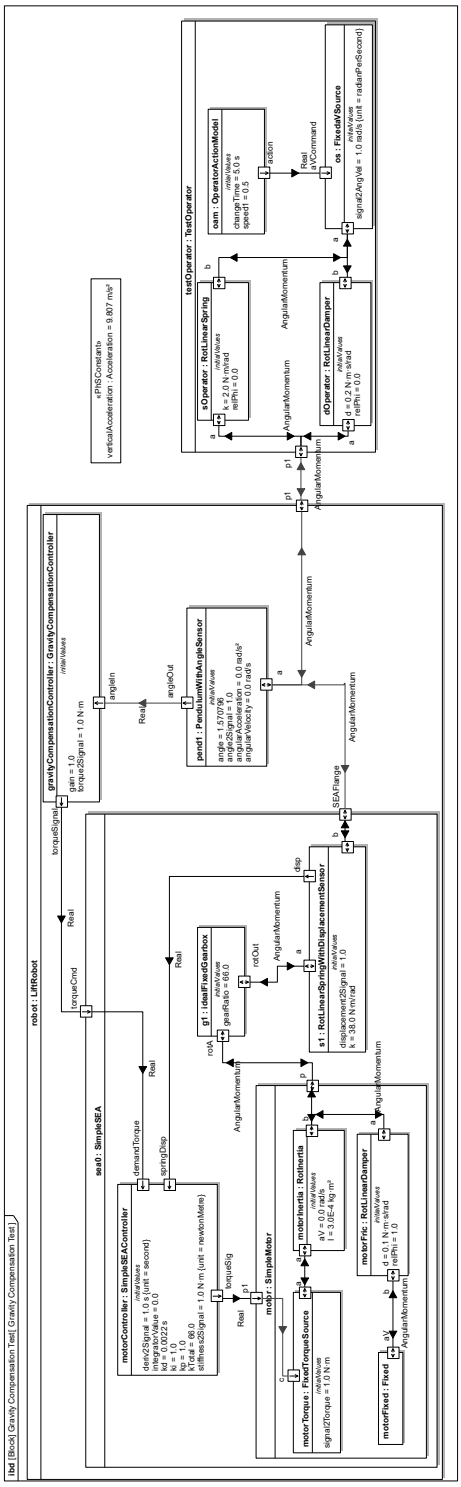

Fig. 65. Gravity compensation test internal block diagram

5.1.5. Simulation

The system was translated to modelica and simulated for 31 seconds with 0.001 second time steps in OpenModelica. The operator was set to move at 5 seconds to allow very slow movement of the arm to be detected. The rest of the time of the simulation allows the arm to complete a rotation after the operator starts moving. The arm starts out horizontal at $\pi/2$ rads, where the SEA applies a torque that keeps it in place. The SEA experiences a small initial start up transient as it accepts the arm load, because its spring starts out at zero displacement, as shown in a plot of the SEA internal motor torque in Figure 66. Internal torque being the torque produced by the torque source inside the motor.

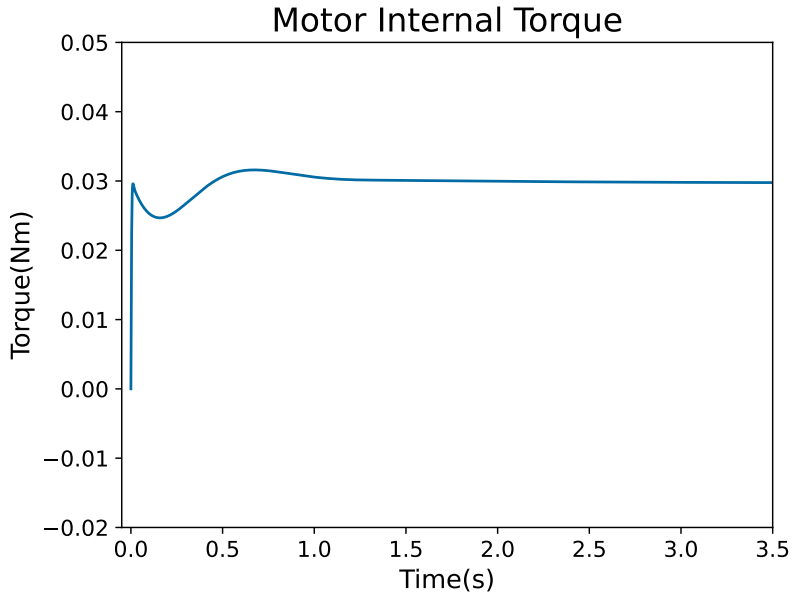

Fig. 66. Motor torque in series elastic actuator

The arm experiences negligible change in angle until the operator starts moving at 5 seconds and the arm follows. Figure 67 shows the arm's movement (flat line, then increasing linearly). The arm stays at $\pi/2$ radians until the operator starts moving at 5 seconds at rad/s. Movement while collaborating with the operator is smooth even as the arm completes multiple rotations.

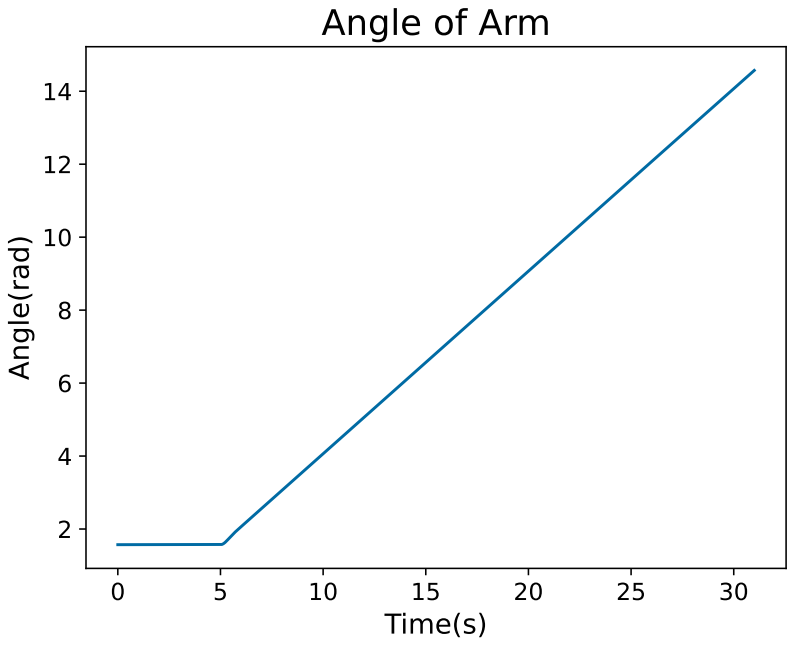

Fig. 67. Angle of the arm

The operator experiences a torque that peaks when the arm starts moving, as shown in Figure 68. This maximum torque is fairly low at 0.12 Nm.

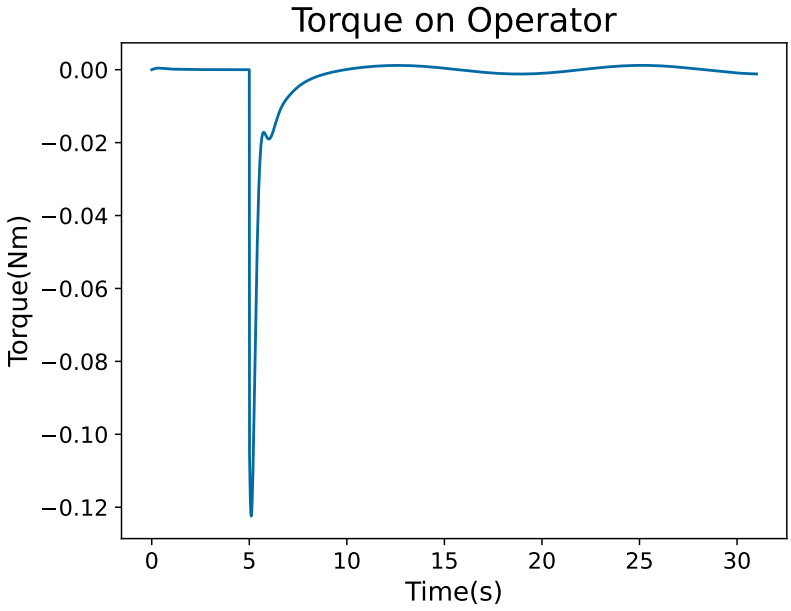

Fig. 68. Torque experienced by operator

Figure 69 shows the torque applied by the SEA (curved line) is much more than the torque the operator experiences (flat line) as it compensates for the weight.

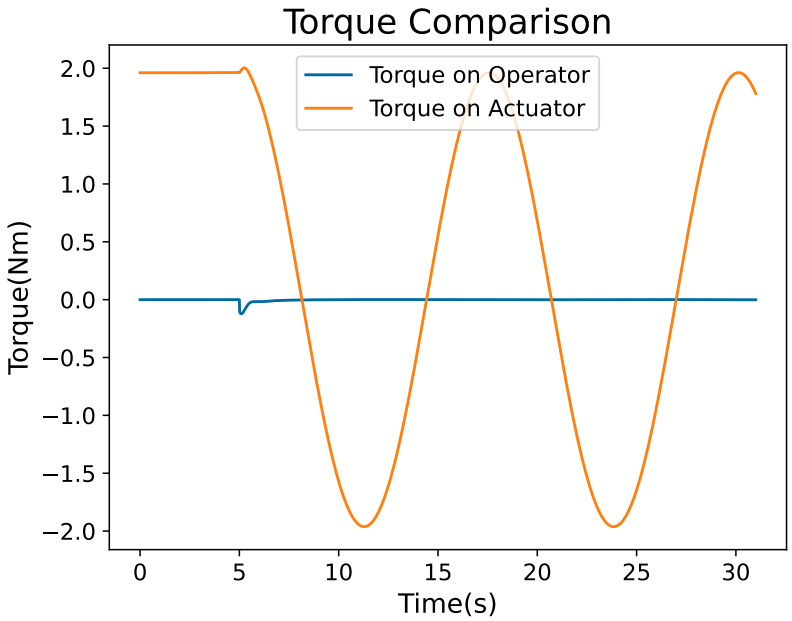

Fig. 69. Torque on operator compared to torque applied to the arm

In systems modelling is it useful to test different configurations of the system. One way to do this is by making generalizations of the model with different parameters and initial conditions. By generalizing the model, the same model structure may be used and only certain parameters and initial conditions may be changed. Figure 70 shows how this model is generalized to two different testing scenarios, one in which the payload mass has been increased(TestCaseHeavierPayload) and another in which the gravity has been decreased(TestCaseMoon).

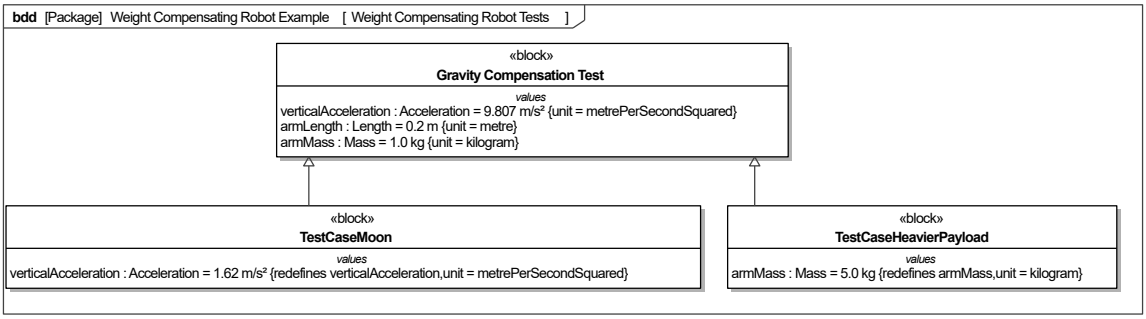

Fig. 70. Different test cases by generalizing the original model

In TestCaseHeavierPayload, the payload on the arm has been increased from 1 kg to 10 kg. In TestCaseMoon, gravitational acceleration has been decreased from 9.807 m/s^2 to 1.62 m/s^2. Figure 71 shows a comparison a comparison of these test cases and the original model. Operating the system in an environment with less gravity(orange), appears to have less of an effect on the system than increasing the payload(grey) compared to the original

case(blue). The lower gravity test case and original case appear to have nearly the same trajectory. This may imply that the system will need to be redesigned if it is to handle more payload.

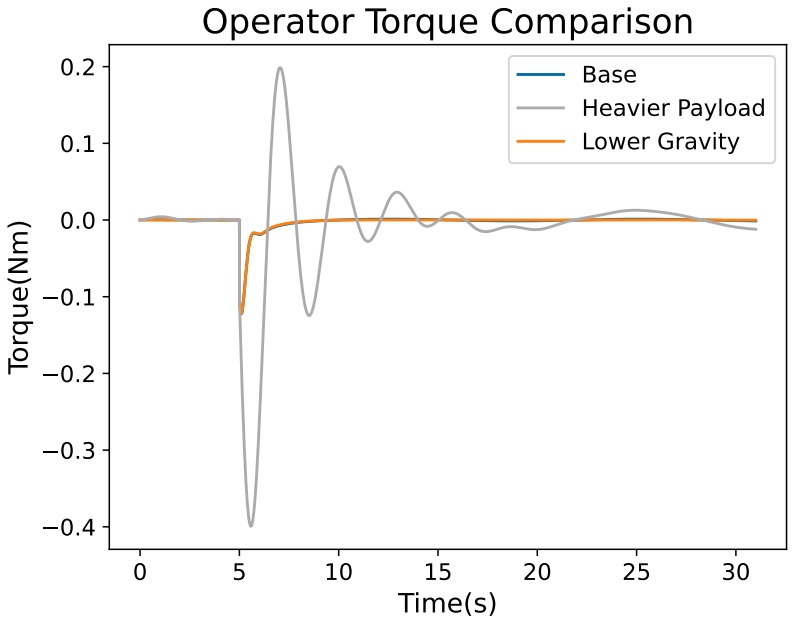

Fig. 71. Comparison of torque on operator with different parameter changes

these simulation results show that the system works roughly as intended. The robot arm can hold an object at a set position and allow the operator to move the object with much less torque than is needed to support the object. Various cases were tested and it was found that the system is sensitive to changes in payload mass. Whether this requires redesign is beyond the scope of this publication. These examples are intended to demonstrate the use of the libraries in the context of manufacturing systems and do not model real systems.

5.2. Fused Deposition Modeling 3D Printer

Most 3D printers use fused deposition modeling (FDM), a process that melts a plastic filament in an extruder and deposits it on a surface, layer by layer, to make 3D shapes. The printer must move the extruder back and forth rapidly as it traces out the part, requiring simulation to determine whether it can carry out such maneuvers. Figure 72 illustrates the mechanics of the 3D printer modeled in this example (the controller and thermal aspects are not shown). Three perpendicular motor drive axes move a tool that extrudes plastic to produce a part.

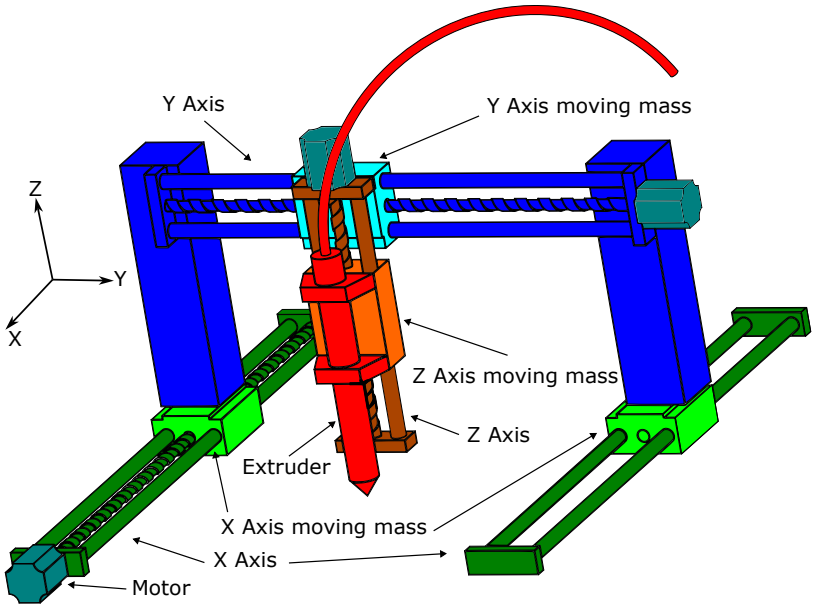

Fig. 72. FDM example (cartesian robot and extruder)

This system is modeled in two parts, one for printer mechanics and another for thermal aspects of the heated bed. The printer model includes a cartesian robot that moves an extruder to trace out a cylinder. Requirements on a 3D printer might include that it produce parts varying in dimension only within a specific range (tolerances), printing speed is higher than some value, and peak power usage is below some value. The printer model enables testing whether a design with given parameters will meet these requirements. The main output is the trajectory of the extruder as it is moved to produce a part, a cylinder in this example. The example trajectory traces first in the X and Y axes, then up in the Z direction, repeatedly. The model does not account for motion limits, motor saturation, and the extruder does not model viscoelasticity of the plastic.

The heated bed model is described in Section 4.3, with thermal effects modeled in Figure 83 in Section 5.2.4. This determines how long it takes to warm up the bed to a set temperature and whether any unsafe temperatures occur in doing this. Thermal effects are not extended to printer mechanics, because thermal transients occur over a much longer time scale than mechanical dynamics (minutes vs. seconds). The mechanics model also does not include collision with the part which has already been printed.

Figure 73 shows the internal block diagram of this system. The control is applied to a cartesian robot that moves an extruder around. The controller sends signals to the cartesian robot and extruder and receives position signals from each axis of the cartesian robot. The extruder is coupled to the cartesian robot and the extruder nozzle is coupled to a zero pressure boundary condition[18]. In this model, the interaction of the extruder with the printed

[18]The zero pressure boundary condition is implemented as a trivial block CnstPressureSource that has a single VolumeFlow port with pressure set to a specified value, in this case zero.

part are not modeled.

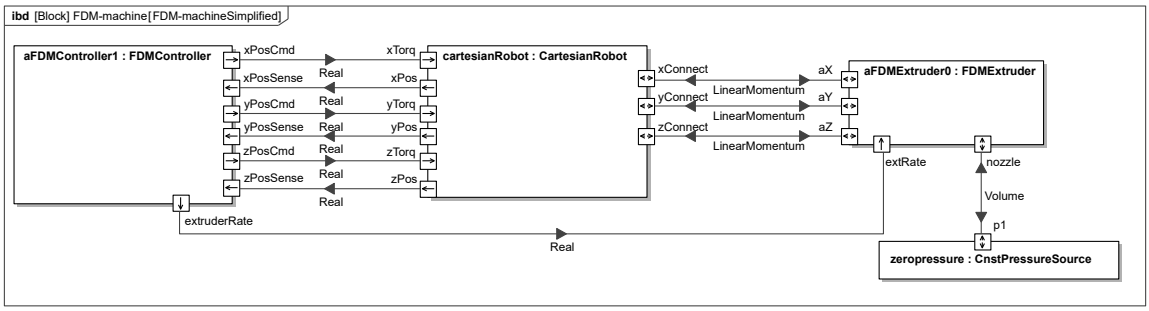

Fig. 73. Total system structure of FDM machine

5.2.1. Cartesian Robot

A key part of the printer is a cartesian robot, consisting of linear actuators arranged to move the extruder along three perpendicular axes. Figure 74 shows detail of an axis actuator. They includes a motor that drives conversion of rotation to translation via a rotating screw going through a moving mass guided by rails and other components which do not move along the axis which are expected to have significant mass. The rotary to translation conversion is taken as ideal even though screws are not typically very efficient, like belts or gear racks are.

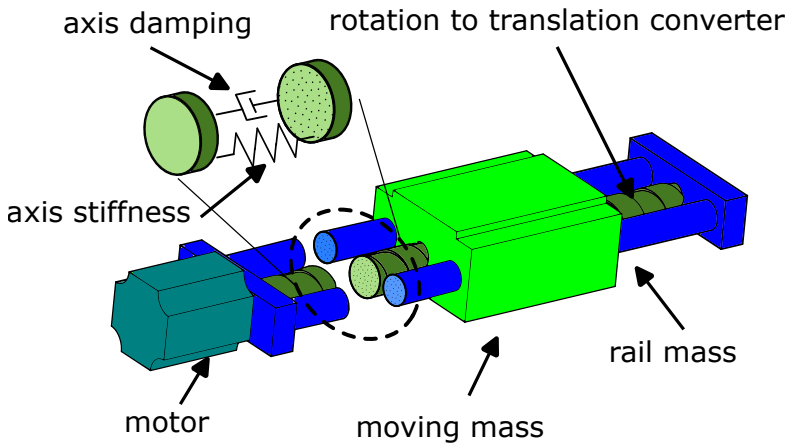

Fig. 74. Components of a single axis actuator

The axis actuator in Figure 74 is modeled by the block AxisLinearMotion, as shown in Figure 75. The motor from Figure 60 in Section 5.1.2 is coupled to an ideal rotary to translational converter (see Figure 34 in Section 4.2), which connects to a spring and damper,

then to port p1, for connection to the mass being moved. Masses of the actuator parts are modeled separately, to enable AxisLinearMotion to be reused on all axes, see Figure 76.

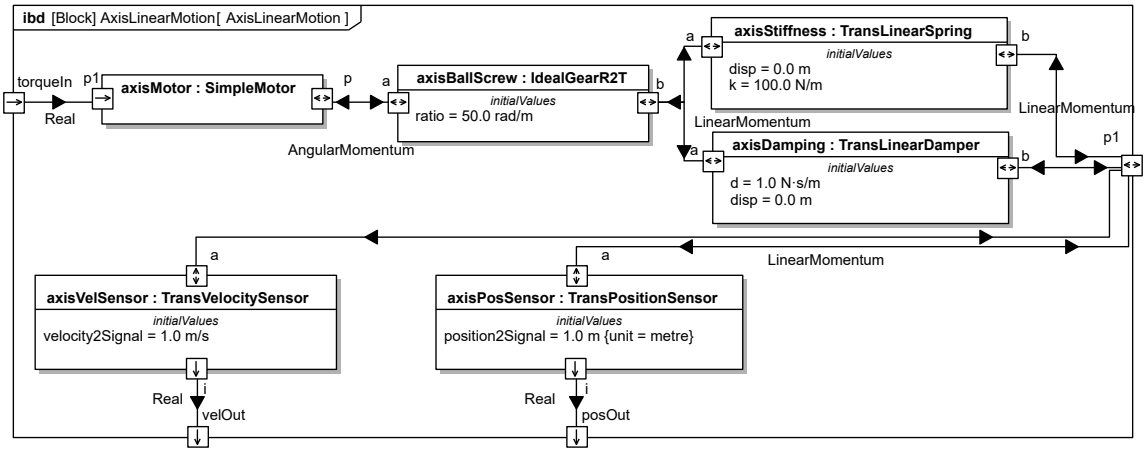

Fig. 75. Figure 74 modeled in SysPhS

One factor contributing to speed, accuracy, and cost of the 3D printer is the stiffness of components between the point at which material is extruded and the surface on which it is extruded. Stiffer designs deform less, and conversely may produce more accurate parts or operate faster, but cost more. It is assumed here that the most significant deformation is in the linear actuators. This is modeled by including a spring in AxisLinearMotion for the stiffness of an axis (axisStiffness) in between the translation-to-rotation converter and port p1. The stiffness could correspond to that of the screw (as shown in Figure 74), or all components of the machine movement axis, or some combination of them. Some damping is also included between port p1 and the translation to rotation converter, which is present in real systems and prevents oscillation of the spring from growing forever during simulation. AxisLinearMotion includes a velocity sensor and position sensor that output the velocity and position of the spring as real signals, on the velOut and posOut ports, respectively.[19] The motor accepts a control signal for desired motor torque from the port torqueIn. The TransPositionSensor (axisPosSensor) integrates velocity on the port it is attached to, so the initial position of the sensor on the axis should be specified as an initial value of the sensor's linearPosition.[20]

Each of the axis actuators is mounted on another actuator in series to move a tool, such as an extruder, through 3 dimensional space. The tool is moved along the X axis by the X axis actuator, which is moved along the Y axis by the Y axis actuator, which is moved along the Z axis by the Z axis actuator. The actuators are expected to have significant mass and each

[19] Both sensors could correspond to a single absolute linear encoder, for example as found in milling machines.

[20] This hypothetical printer uses linear servo motors, rather open loop stepper motors as in most 3D printers, for simplicity. These have sensors and continuous motors, and are found in some 2D printers and in motion control systems.

must take into account the total inertia being moved on its axis, including other actuators and the tool. For example, the Z axis actuator moves the Y and X axis actuators, as well as the tool, all contributing to inertia on the Z degree of freedom. This is modeled in Figure 76 with each axis (cXAxis, cYAxis, and cZAxis) combining an AxisLinearMotion (xAxis, yAxis, and zAxis) with corresponding inertial elements.[21] The inertial chain begins with the tool connected to the robot via momentum ports (xConnect, yConnect, and zConnect). The X actuator (cXAxis) moves the tool by its connections to these ports, and is moved via its own ports by the Y actuator (aY, aZ), which in turn has ports for being moved by the Z actuator (aZ).

The inertia of each of the axes is modeled in Figure 76 by Trans3DInertia blocks (see Section 4.1), distinguished by whether they represent the part of the actuator that moves with respect to it (moving mass in Figure 74) or the parts that do not (rail mass in Figure 74). The X and Y actuators have both of these, moving (xMoveMass and yMoveMass, respectively) and not (xRailMass and yRailMass, respectively). The Z axis actuator only models its moving mass (zMoveMass), because its rail is assumed to be rigidly fixed to the ground, not affecting dynamics. All three degrees of freedom of xMoveMass connect to the tool via ports (bX, bY, bZ) while xMoveMass is moved along the X degree of freedom by the xAxis actuator and the Y and Z degrees of freedom by the moving part of the Y axis yMoveMas. The Y and Z freedoms of yMoveMass connect to the corresponding ports of the X actuator (aY and aZ via bY and bZ), because the Y actuator is directly connected to the X, moving it only in Y and Z. The Z degree of freedom of zMoveMass connects to the corresponding port of the Y actuator (aZ via bZ), moving it only in Z.

[21] The inertias could be modeled in AxisLinearMotion to reflect the mechanical structure of the actuator, see Section 7.

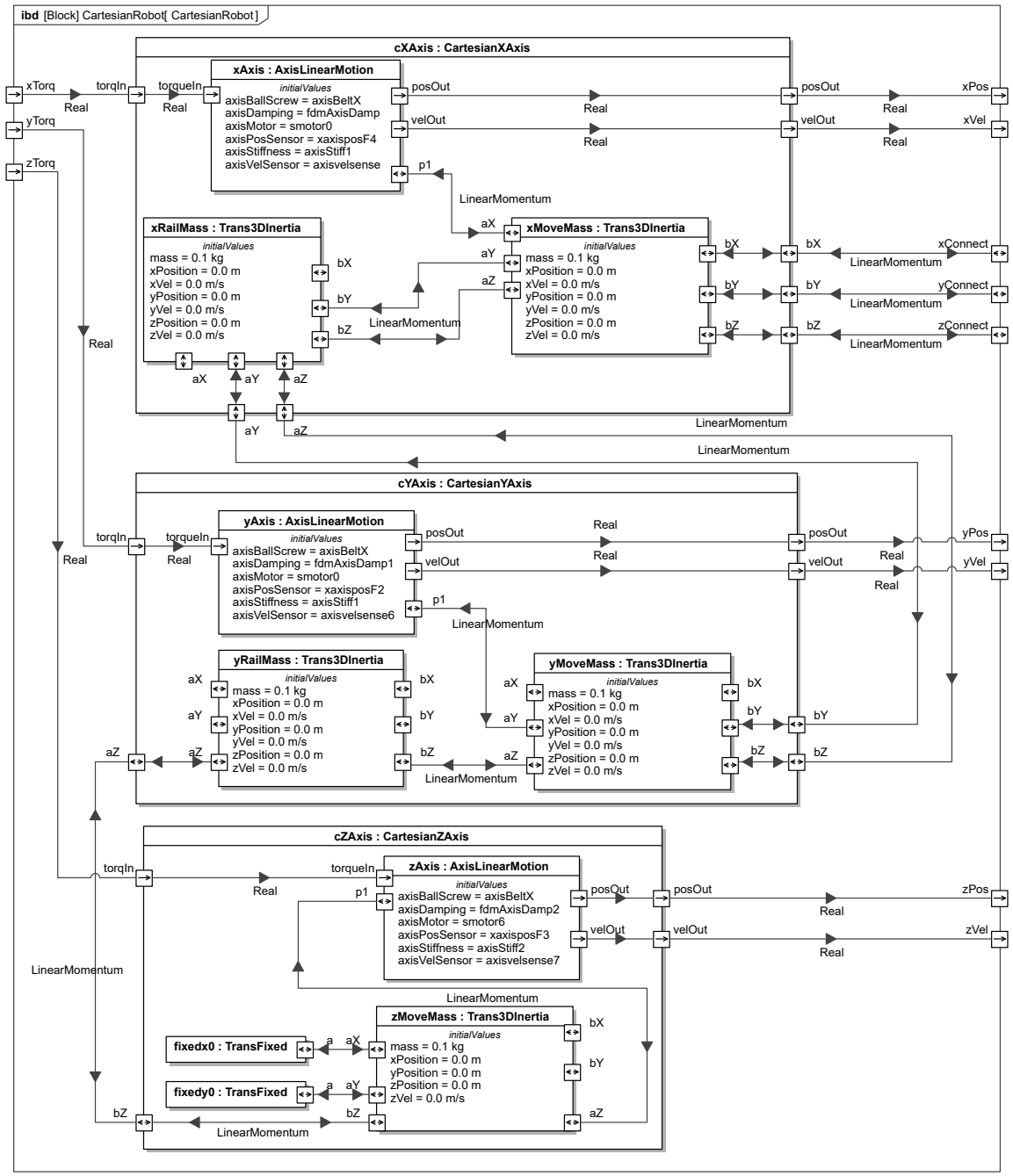

Fig. 76. Cartesian robot

The momentum port of each AxisLinearMotion (p1) connects to the port of its moving mass that corresponds to the axis (aX, aY, aZ in cXAxis, xYAxis, and cZAxis, respectively). The remaining ports of the moving mass are connected to the corresponding ones on the rail mass, but only for the axes the rail mass moves on. For the X axis, the rail mass only moves along Y and Z, modeled by connecting those ports (bY and bZ) to the corresponding ones

on the the moving mass (aY and aZ). The X degree of freedom (bX) is not connected to anything, because X rail mass does not move along the X axis. For the Y axis, the rail mass only moves along Z, modeled by connecting that port (bZ) to the corresponding one on the the moving mass (aZ). The X and Y degrees of freedom (bX and bY) do not connect to anything, because they do not move along those axes. The Z axis has no rail mass because it is rigidly fixed to ground, modeled by connecting TranstlFixed (see Section 4.1) to the moving mass for the X and Y degrees of freedom (aX and aY).

The same rail mass degrees of freedom that connect to the moving mass in each axis also connect to another axis that moves it via ports for that purpose (aY and aZ for the X axis, just aZ for the Y axis), which are connected to the other axis on ports for the things it moves (bY and bZ for the Y axis, just bZ for the Z axis). This enables the inertia of the X axis to be propagated to the Y axis, and the inertia of the Y axis to the Z, an example of multiple components contributing inertia on each axis. The inertial chain begins with tool's contribution to inertia (see Section 5.2.2) along all axes, as shown by its connections to the cartesian robot in Figure 73, with the X portion affecting the X actuator, the Y portion propagated to the Y actuator along with moving an rail masses of the X actuator, because all these elements move along the Y axis, and the Z portion propagated to the Z actuator along with the moving and rail masses of the X and Y actuators, because all these elements move along the Z axis.

5.2.2. Extruder

FDM printers use an extruder to melt plastic filament and deposit it on the part being made. The extruder model in Figure 77 includes a Trans3DInertia block (see Section 4.1) to represent its mass and another block for fluid flow from the extruder, contains a simplified model of fluid flow, defined in Figure 78, that covers some fluid effects, but ignores thermal ones. It is for the kind of extruder where a motor, typically with a high gear reduction, turns a screw or gear that pushes the filament through a hot nozzle to melt it. The motor with a high reduction gear is modeled as an angular velocity source (extDrive). It is coupled to an ideal rotary to translation converter (see Figure 34 in Section 4.2) that drives two hydraulic components modeling how a solid filament interacts with the molten plastic in the extruder, see next. The output of the extruder is coupled to a zero pressure boundary condition, see Figure 73.

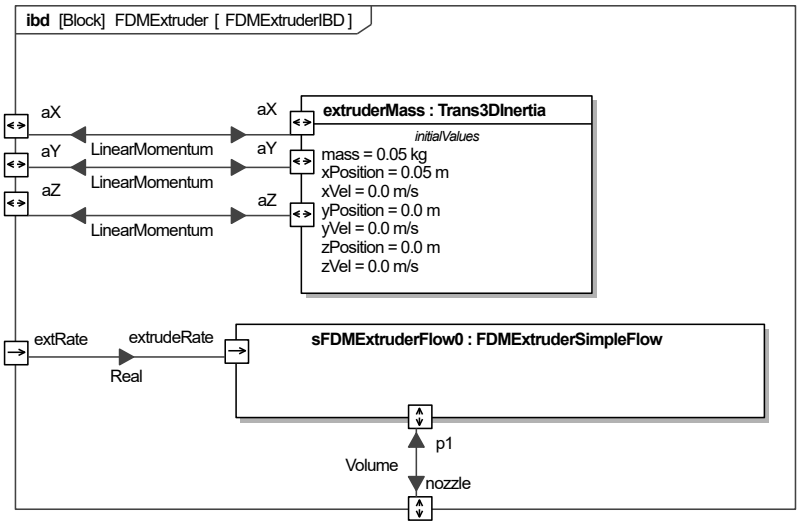

Fig. 77. Extruder internal block diagram

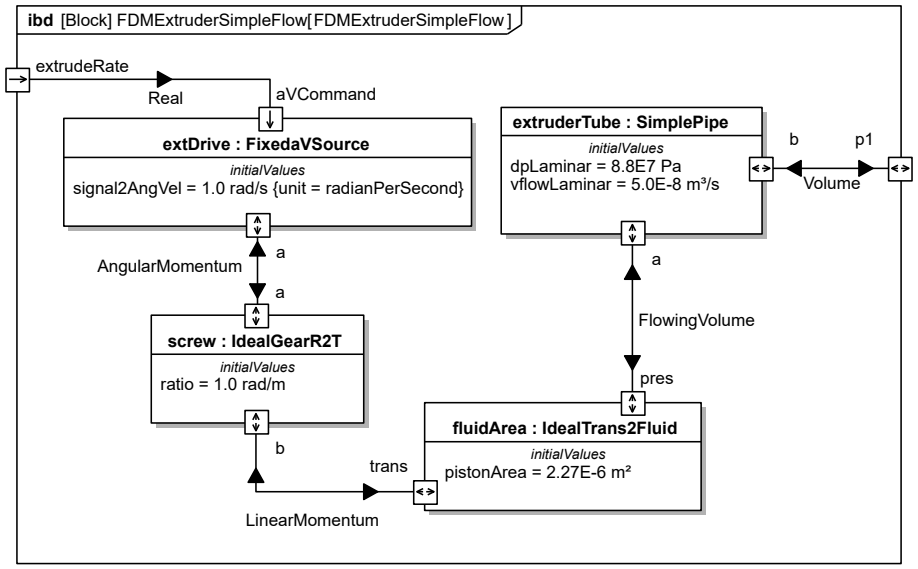

Fig. 78. Extruder flow model

Figure 79 defines the translation to volume flow converter used in Figure 78 (IdealTrans2Fluid). It models a device that converts translational movement to a volume flow through an area, such as a piston, except it enables unlimited translational displacement. The element is ideal without elasticity, friction loss, or fluid loss.

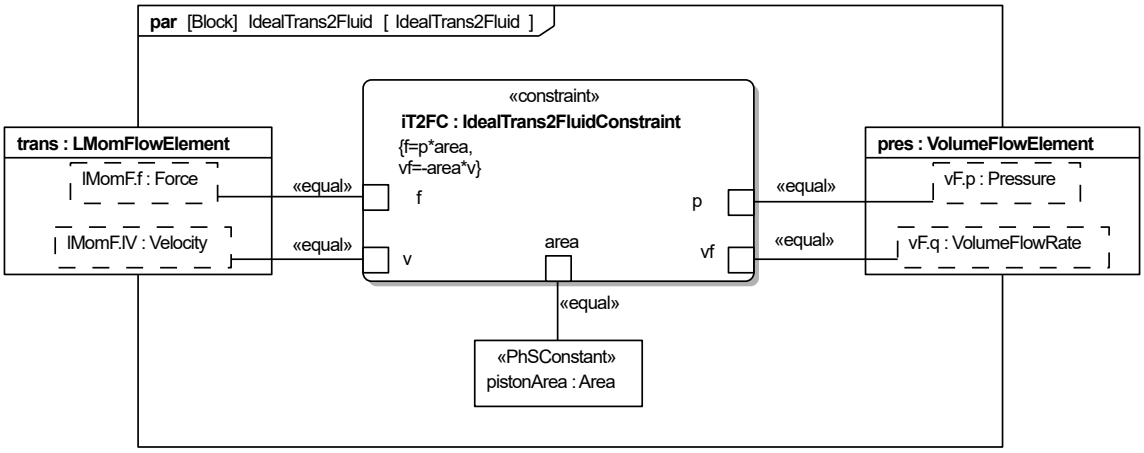

Fig. 79. Translational to volume flow converter

Figure 80 models the hydraulic resistance to forcing molten filament through a nozzle (SimplePipe) used in Figure 78. The difference in pressure across it (between its ports) is proportional to flow rate multiplied by some resistance factor. This is expressed as the ratio of pressure difference (dpC) and volume flow rate (VflowC) that results from the pressure difference, as shown in Figure 80.[22] The element does not account for non-newtonian behavior of molten plastic or changes in resistance due to changes in nozzle height.

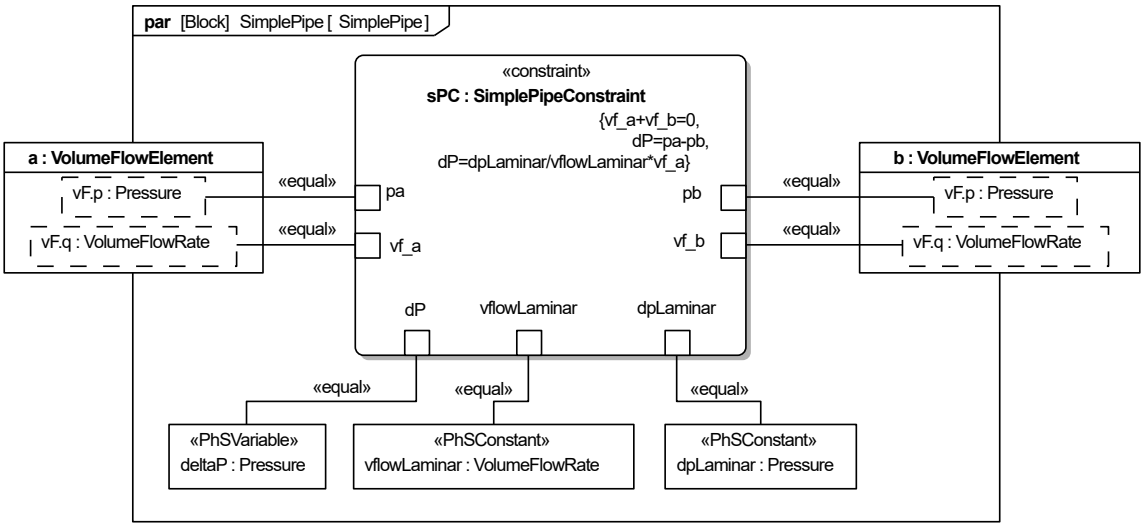

Fig. 80. Hydraulic resistance

5.2.3. Controller

The FDM controller directs the axis actuators to move the extruder through a trajectory tracing out a cylindrical part. It starts with a circle in the X and Y dimensions, then moves up in

[22] This model is based on SimpleFriction in the Modelica Standard Library FluidHeatFlow [20].

the Z dimension, and repeats indefinitely. The controller consists of two parts, a trajectory generator that outputs positions over time and a PID position controller for each axis, as shown in Figure 81. The block CylinderMakerFDM outputs positions on each axis for the extruder to move through a cylinderical trajectory (xPosTarget,yPosTarget,zPosTarget). It also turns the extruder on and off (extruderRate), on when tracing a circle and off when moving up. The position signals are sent to three PID position controllers (see Figure 11 in Section 3), one for each axis. Each axis PID controller takes in the desired position from the trajectory generator (setPoint), the present position of the machine axis (curValue) and outputs a control signal for the corresponding axis (outSig).

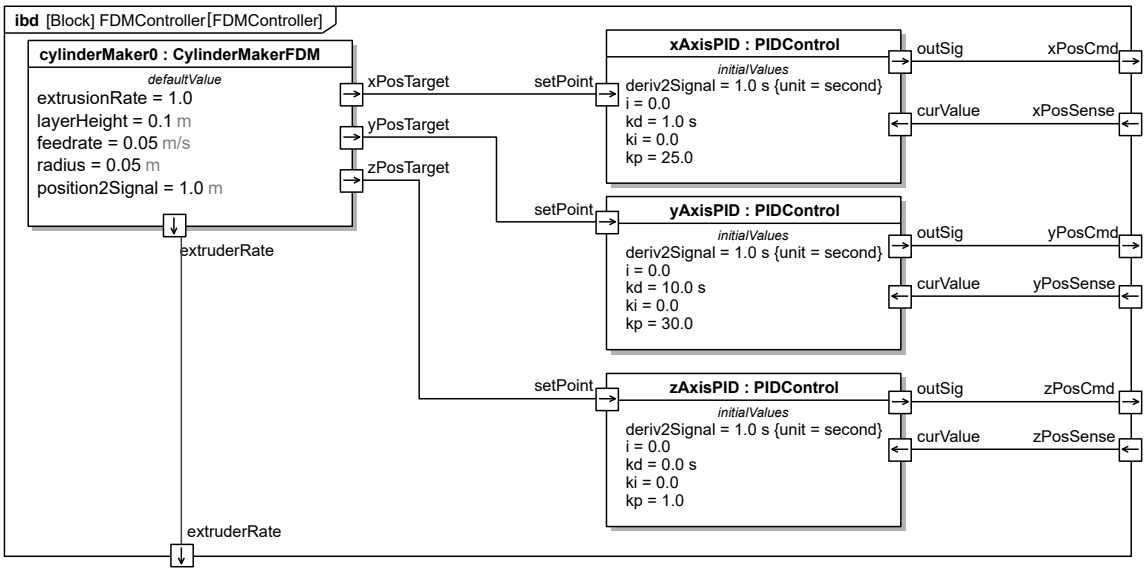

Fig. 81. FDM controller

CylinderMakerFDM generates a trajectory that traces out a circle in X and Y, moves a set distances up in Z, repeatedly. This circle is centered at x=0,y=0 and tracing starts and stops at x=1,y=1. CylinderMakerFDM has properties that are constant for each simulation run:

- feedrate is the speed at which the extruder is intended to move.
- radius is the radius of the circle to be traced.
- layerHeight is the Z distance the extruder is intended to move up each cycle. In the FDM process, this can also correspond to the thickness of an extruded layer.
- position2Signal is the inverse conversion factor between signal output and position calculated.
- extrusionRate is the real signal that is output to the extruder when it is to be turned on. This may be used for motor rotation rate as in this model or just to indicate that the extruder is on.

Figure 82 shows the parametric diagram for the block CylinderMakerFDM. It includes variables for intermediate calculations:

- circleTime is the time it takes the extruder to trace out a circle.
- cycleTime is the time it takes to complete a cycle, that is tracing a circle and moving the extruder up.
- layerNumber is number of completed layers.
- cyclePosition is amount of time spent so far towards the completion of a cycle.

CylinderMakerFDM generates positions for a circular trajectory, then for moving up a set distance in Z repeatedly. The time spent towards completing the current cycle is determined by $b = mod(time, cycleTime)$. If this is less than the time needed to trace out a circle, the block uses the parametric equations for a circle to trace a circle. Otherwise, it outputs a linear increase in Z with time, corresponding to moving the extruder up. It is not recommended that any of the PhS variables be given initial values, as they are intermediate variables constrained by the equations. The output might be discontinuous as events are generated for mod and if statements. This is explained in Section 5.2.5.

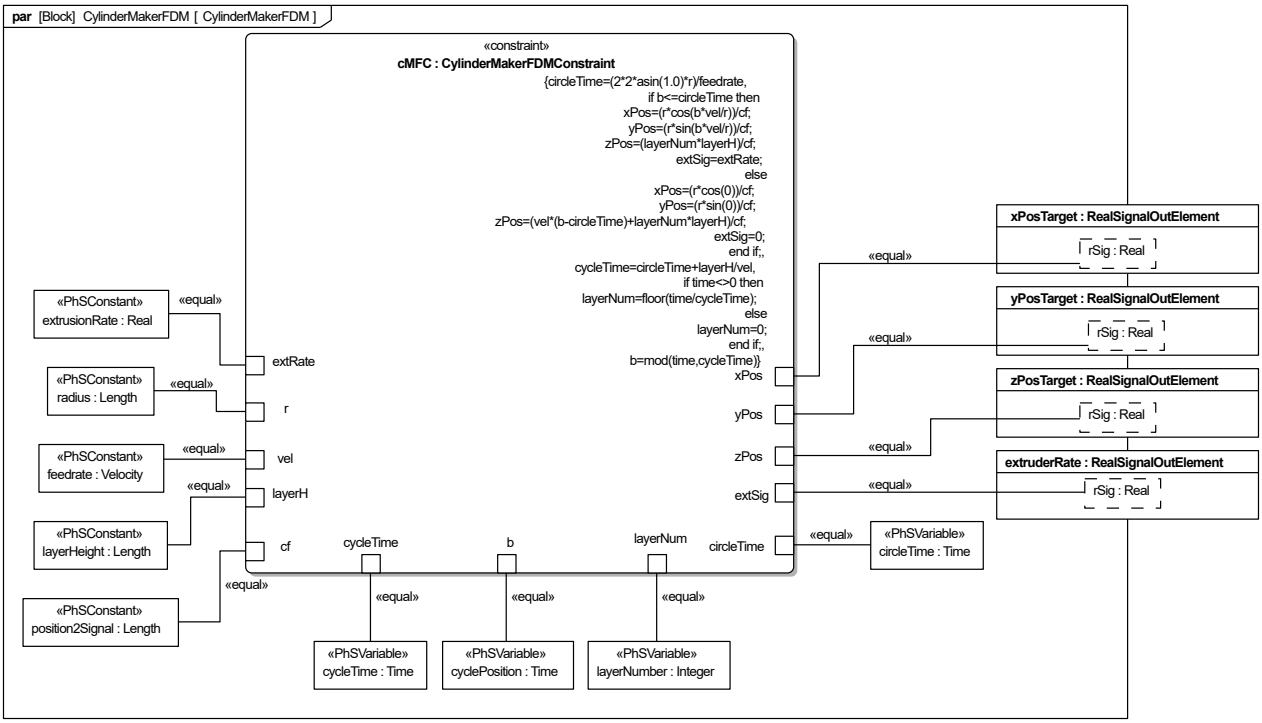

Fig. 82. CylinderMakerFDM parametric diagram

5.2.4. Heated Bed

Fused deposition modeling machines often extrude material onto a heated plate, to slow down cooling of the part being printed. This reduces warping and provides better adhesion of the filament. It is useful to determine how long it takes for the heated bed to warm up and whether it will reach dangerous temperatures. This is modeled separately from the cartesian robot because the dynamics of the heated bed occur over a much longer time scale and are not coupled to the dynamics of the robot model (effects of the hot filament on bed temperature are excluded). It is expected that the FDM machine starts up cold, the bed is raised to operating temperature, and the printing process begins.

Figure 83 shows the application of the HeatedBed block defined in Figure 40 of Section 4.3, and explained in that section. In this test the heated bed starts out at 293 K in a 293 K environment. The bed needs to reach a temperature of 373 K in order to operate. Figure 83 connects the heated bed to a FixedTemperature block (see Section 4.3) at 293 K and a constant real signal of 373.

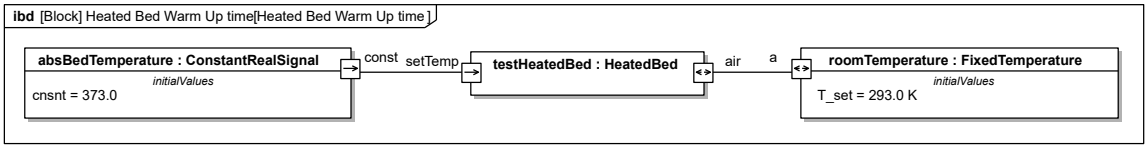

Fig. 83. Heated bed warm up time model

5.2.5. Simulation

The cartesian robot and heated bed are simulated separately because the robot operates on a much shorter time scale than the heated bed.

5.2.5.1. Cartesian robot

Simulations of the cartesian robot, controller, and extruder models in Sections 5.2.1 through 5.2.3 predict how closely the extruder will be from its intended position, to check whether the resulting part will be out of tolerance. Figures 84, 85, and 86 show how the trajectory of the extruder(actual) lags behind the intended postion(target). They were produced in OpenModelica using the integration algorithm DASSL, running for 25 seconds with a 0.001 second step size. This was configured to trace a cylinder with a 5cm radius.

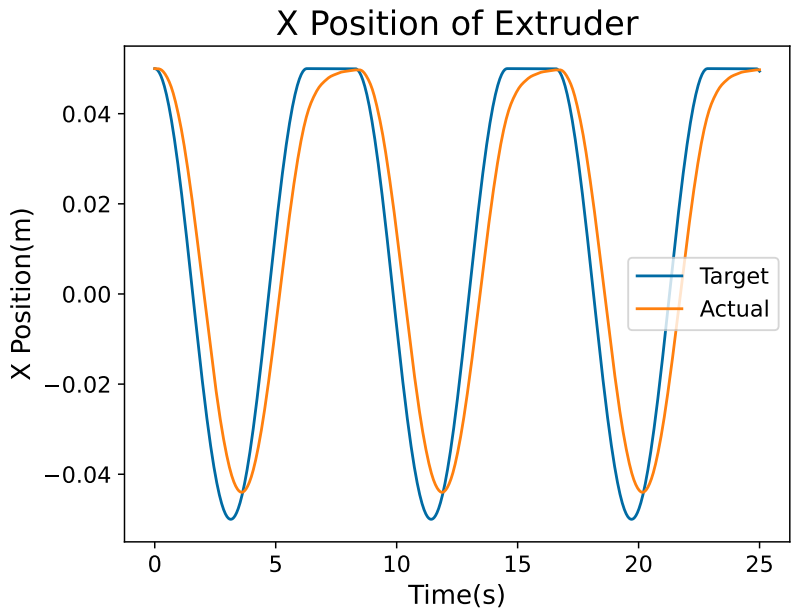

Fig. 84. X position of the extruder compared to intended position

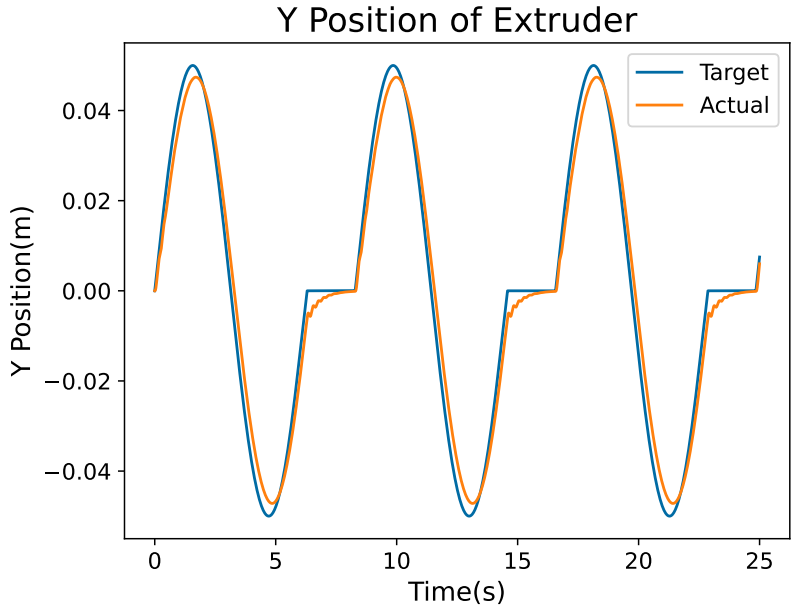

Fig. 85. Y position of the extruder compared to intended position

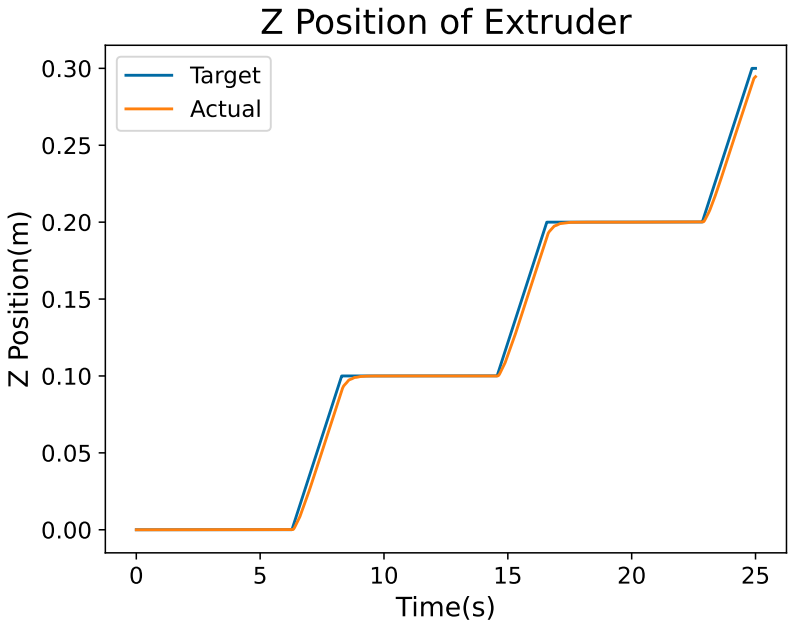

Fig. 86. Z position of the extruder compared to intended position

In the case of the X position, shown in Figure 84, there is a fairly significant deviation from the intended trajectory. The extruder does not reach the maximum radius in the -X direction and is too slow to reach the maximum radius in the +X direction. The Y position is also unable to reach the intended radii and is off from the intended position by a couple of millimeters. This will result in parts being constructed with incorrect dimensions. The extruder might collide with the part being built in some cases, because it cannot reach the intended X position before completing a circle and its Z position lags behind the intended position. This indicates that the controller should use different parameters or be redesigned to better trace out the intended trajectory.

5.2.5.2. Heated bed

This model in Section 5.2.4 was simulated for 400 seconds with 0.01 second time steps on OpenModelica. This is longer time period than the cartesian robot model, because heat dynamics occur over longer timescales.

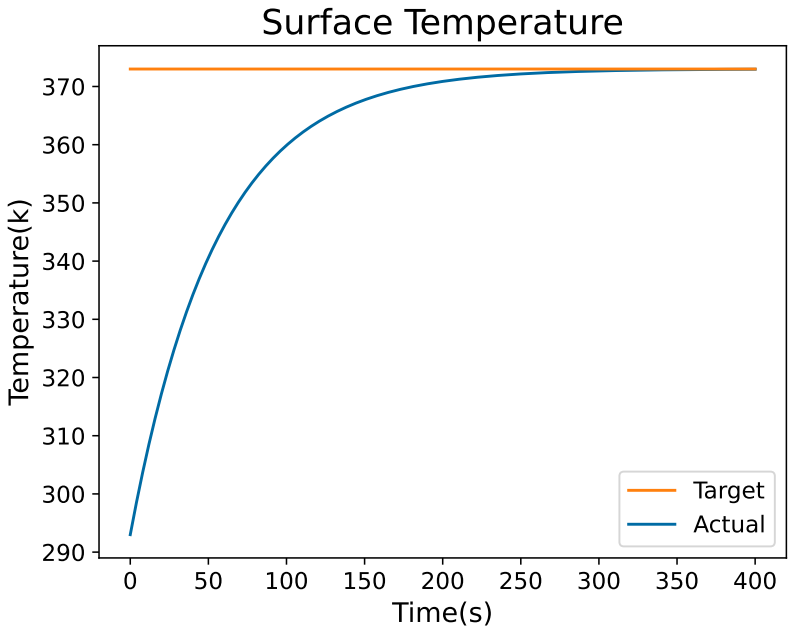

Fig. 87. Temperature of heated bed surface with time

Figure 87 shows how the surface temperature of the heated bed, shown in blue(actual), approaches the set temperature, shown in red(flat line, target). Convergence takes around 300 seconds. Convergence to the intended temperature indicates the controller is functioning as intended as there is no overshoot that may be harmful to the heated bed. This convergence time might be used as an estimate for how long it takes the bed to warm up.

5.3. Polishing Machine

One task manufacturing robots can perform is polishing and deburring complicated parts. The robot moves an abrasive tool (rotary or belt) over a part while applying a constant normal force to it. Variations in applied force are undesirable as they might result in nonuniform deburring or finish. Active control is necessary to apply a constant normal force and keep the robot on the desired trajectory in spite of reactions from the abrasive tool and workpiece.

Figure 88 illustrates a simplified polishing machine that moves a rotating polishing wheel through a predefined trajectory over a circular work piece, maintaining a constant normal force on it. Circular workpieces are simpler than most manufactured parts, but have some similarities with more complicated workpieces. The contact between two rigid disks will be unstable when one disk is held in place and force is applied to the other, the other disk will diverge from the fixed disk if the force does not point exactly to the center of the fixed disk. Two force controlled axis actuators move the polishing wheel in X and Y, while applying force to it. The axes in this examples are series elastic actuators (see Section 5.1.2) connected to rotation to translation converters. The polishing wheel is connected to

a motor (see Figure 60 in Section 5.1.2) that is given a constant torque signal. The circular workpiece is fixed in space, while the circular polishing wheel moves along the X and Y axes around it. Friction between the polishing wheel and workpiece is assumed to be viscous (linearly dependent on normal force and velocity difference at the contact point). A force is applied normal to the contacting surfaces based on how much the two circles interpenetrate and their combined stiffness.

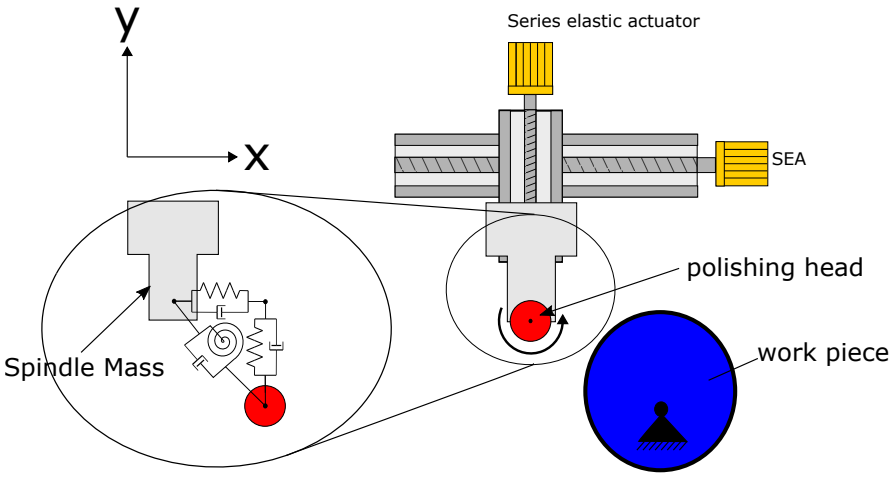

Fig. 88. Polishing machine

5.3.1. Machine

The polishing machine in Figure 88 consists of two force controlled axes, a spindle, and a polishing wheel, as model in Figure 89. Each force controlled axis has a series elastic actuator coupled to a rotary to translational converter. Each axis has a 2D Inertia block connected to the ForceControlledAxis block output modeling the inertia of the moving part of the machine axis. The Y axis is moved by the X axis, so it has an additional 2D inertia block representing the inertia of the part of the Y axis that does not experience motion in the Y direction. These two axes move the block spindle, which contains a 2D inertia representing the mass of the tool spindle and a motor. This connects to the polishing wheel with linear momentum flow elements for X and Y translation of the polishing wheel and an angular momentum flow element for the rotary degree of freedom of the polishing wheel.

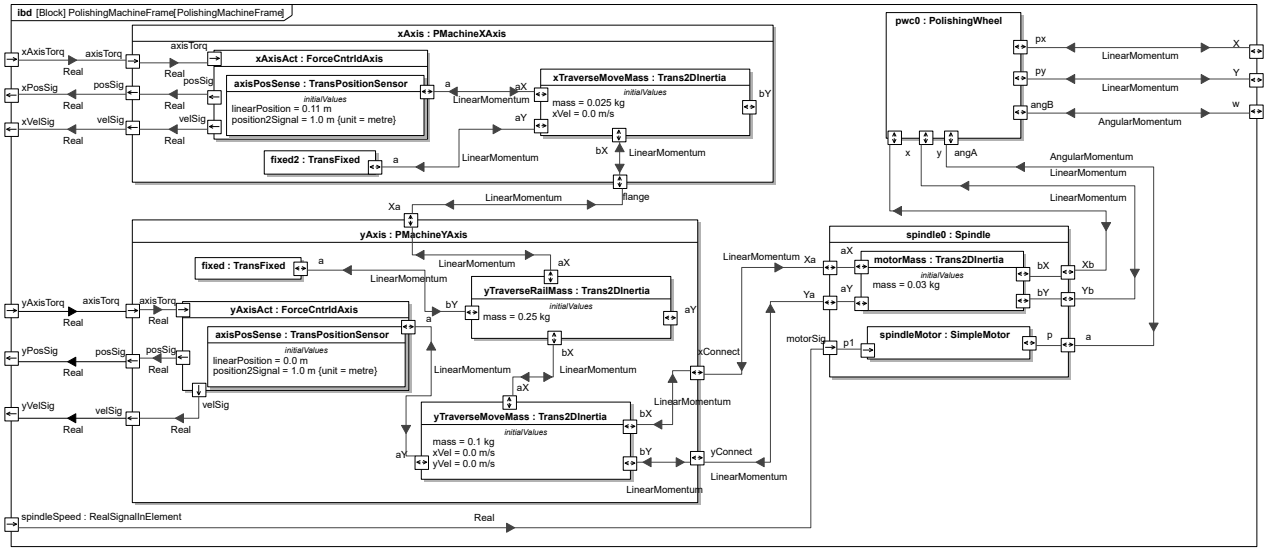

Fig. 89. Polishing machine axes and spindle

Figure 90 shows a force controlled axis model, which has translational position and velocity sensors connected to translational output. These are used to provide real signals for the current position and velocity of the axis. The initial position of the translational position sensor should be set to define the initial position of the axis. The axes are connected in a similar manner to the axes in the FDM machine.

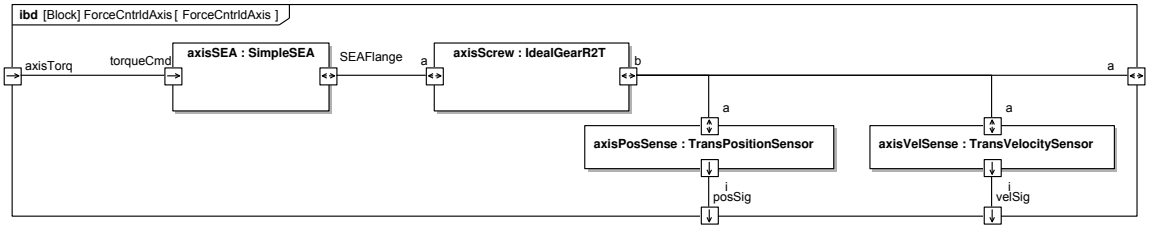

Fig. 90. Force controlled axis IBD

The block PolishingWheel, shown in Figure 91, represents the inertia of the polishing wheel and the elastic properties of the mechanical connection between the polishing wheel and motor. The Linear Momentum port x,y and the angular momentum flow port angA, represent the mechanical connection to the spindle. The ports px,py, and angB respresent the connection to the contact model/workpiece. The polishing wheel is expected to have mass and angular inertia that cannot be ignored, so it contains a 2D translational inertia and angular inertia. The connection between the motor and the polishing wheel that is expected to be somewhat thin may have a stiffness that cannot be approximated as infinite. In addition, the polishing wheel may be made of a somewhat deformable material. So springs and dampers are connected between the X and Y linear momentum ports and the translational inertia representing the polishing head mass to model this possible deformation.

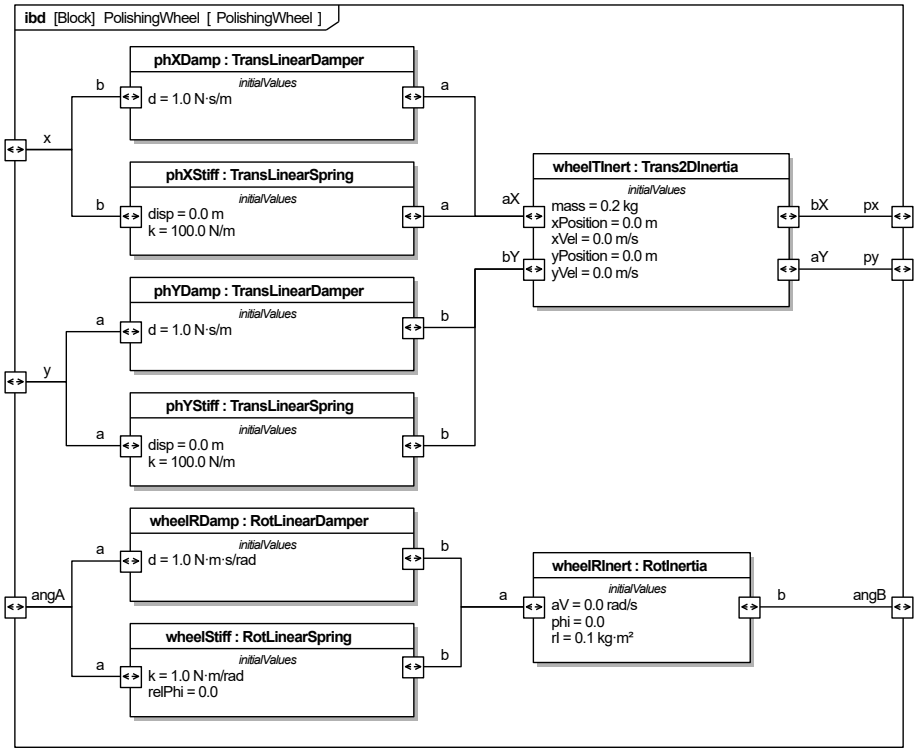

Fig. 91. Polishing wheel IBD

The block CircleCircleContact, parametric diagram shown in Figure 92, is used to describe the contact forces and torques between the polishing wheel and the workpiece. In this model we assume both the polishing wheel and the work piece are circular. The polishing wheel is able to translate in X, Y and rotate, while the workpiece is assumed to be rigidly fixed. Contact can be determined by checking if the distance between the center of the two circles is less than the sum of their radii. Contact normal force is presumed to be proportional to the distance the two circles are interpenetrating. If there is no contact, the forces and torques on the polishing wheel are zero. Tangential forces are calculated using the relative velocity of the contact point and viscous friction.

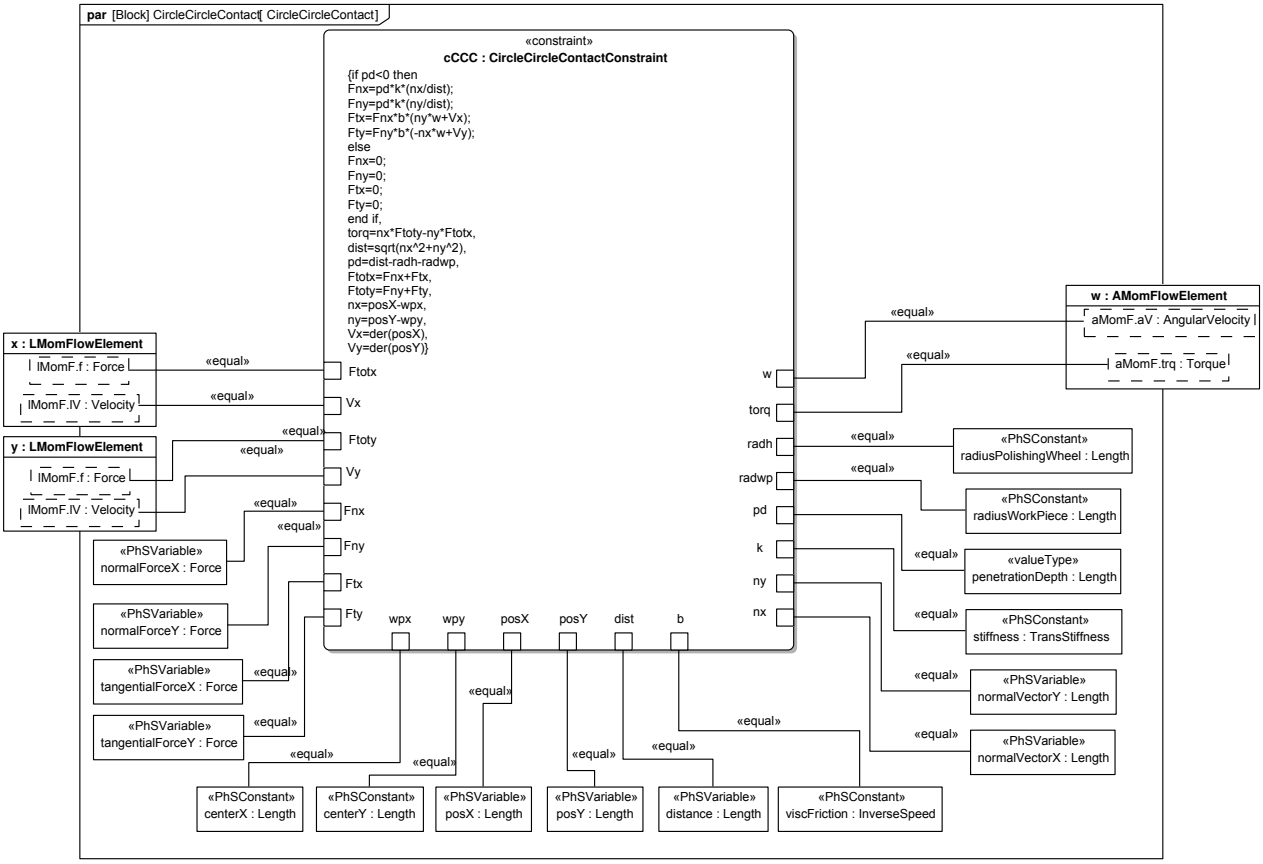

Fig. 92. Contact model

5.3.2. Controller

The controller determines the forces that axes should apply to move the polishing head over a part, while applying a constant normal force to it. Controlling both the force an end effector applies and its position may be done with a technique known as hybrid position force control [21]. This works by mixing the force to be applied with a force that moves the end effector along the trajectory. The controller shown in Figure 93 uses CircularPathGenerator to generate a circular trajectory from time, CircleNormalForceGenerator to calculate the normal force from the current position of the polishing head, and blocks that implement hybrid position force control take the intended trajectory and force to determine control force. The controller also outputs a constant signal to the polishing wheel motor.

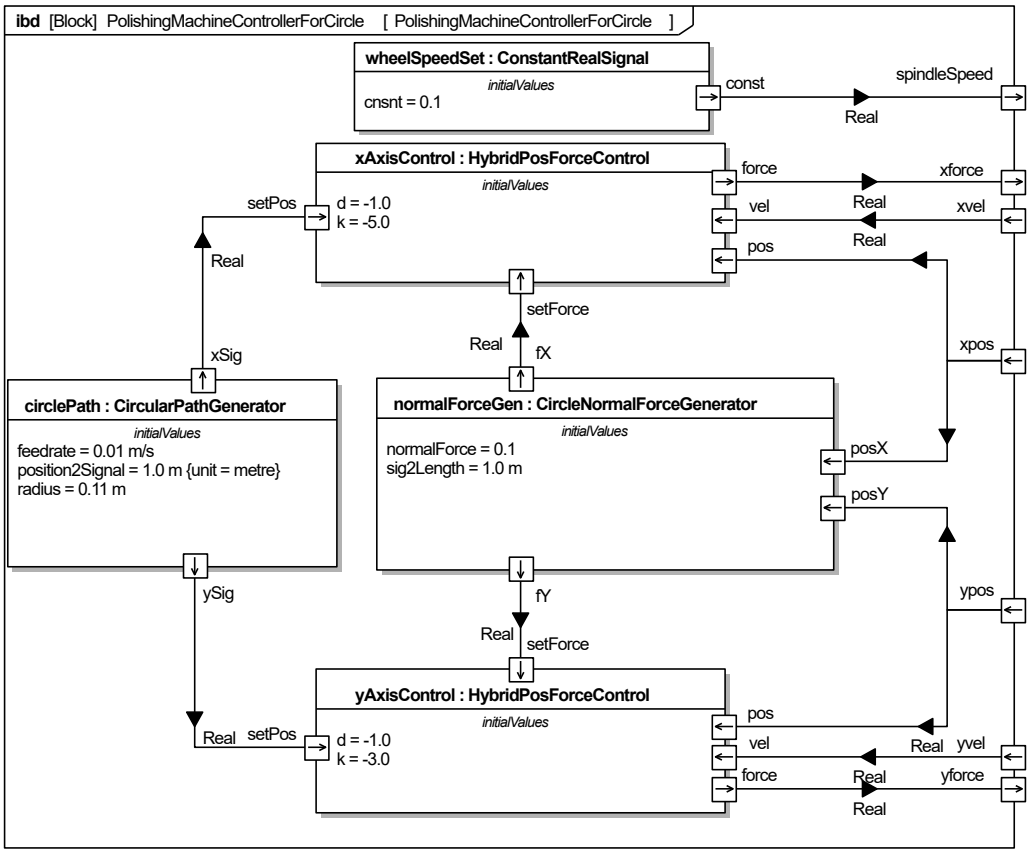

Fig. 93. Polishing machine controller

Hybrid position force control in this example mixes the normal force applied to a part with a force that pulls the polishing head along a path around it, as calculated in Figure 94. SEAHybridPositionForceControl takes in the current position along one axis, the target position along that axis, velocity along that axis, desired force to be applied along that axis, and outputs a control force. This is calculated by summing the normal force to be applied to the part with a force proportional to the deviance from the intended position and a force proportional to present speed. That is we apply force $F = k * e_pos + F_{target} - d * V$, where k is proportionality coefficient, e_{pos} is position error, F_{target} is target force, V is velocity, and d is damping coefficient.

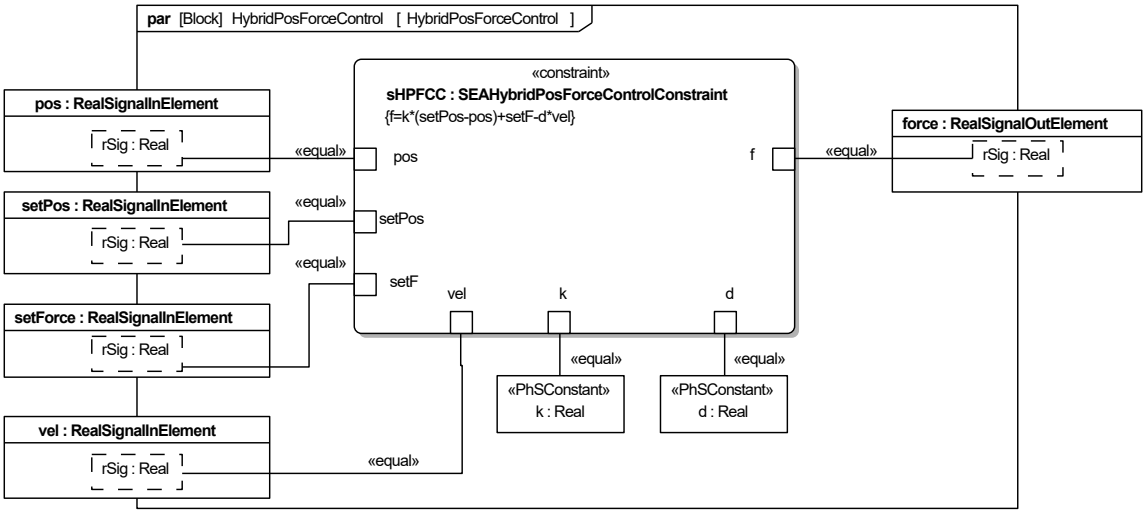

Fig. 94. Hybrid position force control

CirclularPathGenerator(parametric diagram shown in Figure 95) outputs x and y coordinates as a function of time, to trace out a circle of specified radius and center position at a specified rate. The position at time zero is x=1 and y=0, with respect to the center position. The parameters centerX and centerY are the center X and Y coordinates, respectively, while radius is the radius of the circular path and feedrate the velocity at which the path is to be traced. The parameter position2Signal specifies the inverse conversion factor between signal output and position calculated.

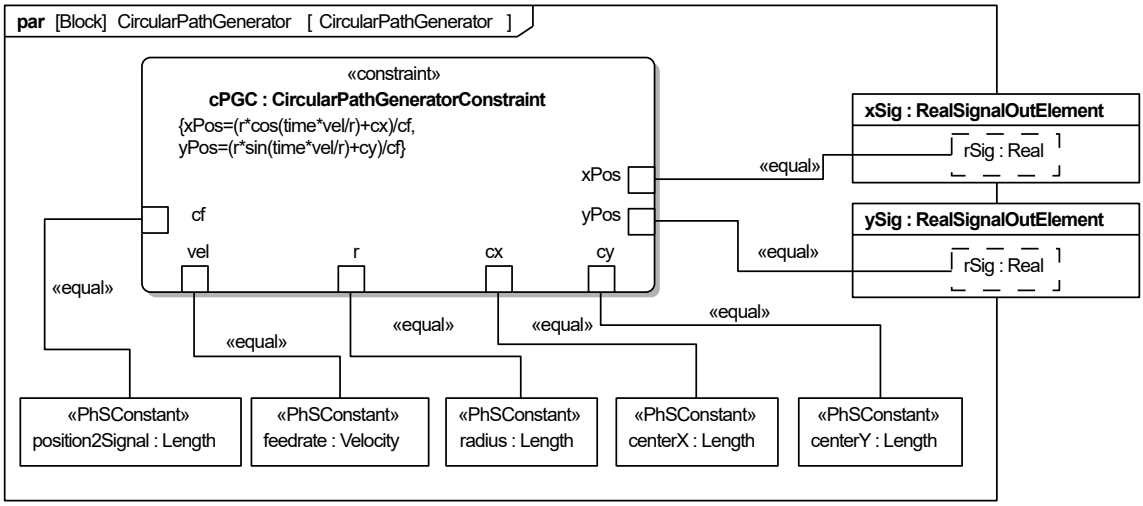

Fig. 95. Circular path generator

CircleNormalForceGenerator calculates a constant normal force to be applied to the circular workpiece, which is towards its center, as shown in Figure 96. The normal vector is determined by the position of the tool (given by two real input signals posX, posY) and

center of the workpiece (two real internal constants workpieceX and workpieceY). The constant normalForce specifies the magnitude of the normal force. From these CircleNormalForceGenerator outputs the X and Y components of the normal force as real signals fX and fY.[23]

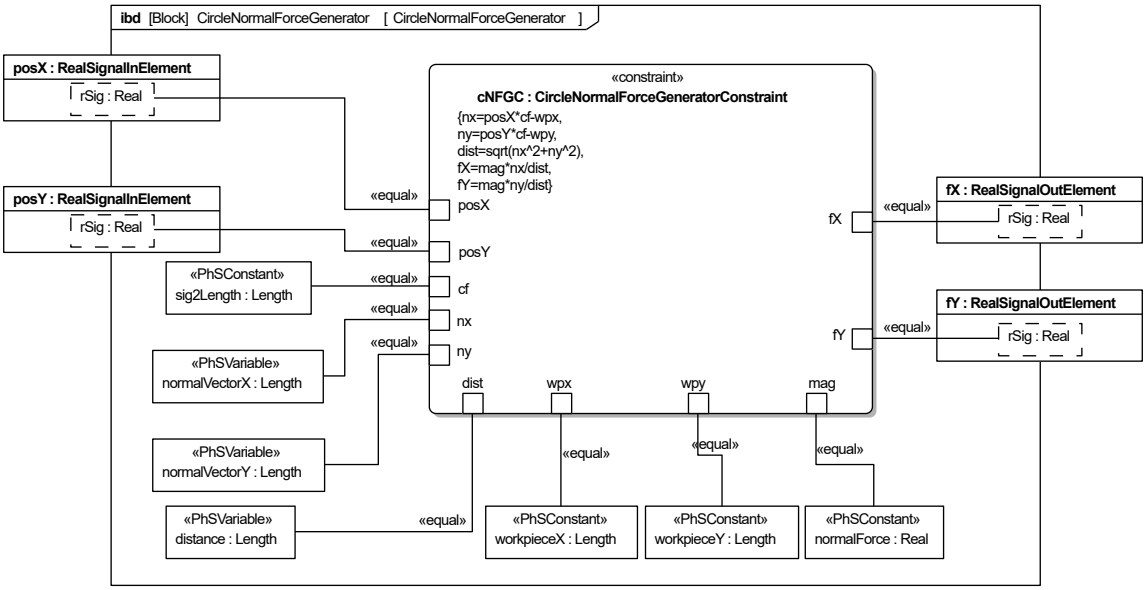

Fig. 96. Circle normal force calculation

5.3.3. Simulation

The polishing machine was simulated on OpenModelica for 70 seconds with a 0.001 second step size. The work piece was set to a larger radius than the polishing wheel, with radii 0.1 m and 0.01 m, respectively. The series elastic actuators are the same as the ones in the weight compensating robot example, see Section 5.1.2. The polishing wheel starts out in contact with the workpiece at position x=0.11 m and y=0.0 m, but with no applied normal force, and zero angular and translational velocities. The normal force to be applied was set to 0.1 N.

Figure 97 shows the magnitudes of the forces on the polishing wheel. The X component of the normal force applied to the workpiece (blue line with oscillation which starts at the bottom) oscillates some at the beginning, due to polishing wheel spinning up and force being applied to the work piece. Both X and Y (orange line which starts in the middle) components settle to tracing out a negative cosine, negative sine trajectory, as would be expected for a vector pointing to the center of a circle. Overall the magnitude of the normal force applied to the work piece (grey line near the top of the figure), is slightly under 0.1 N the intended value of the normal force.

[23]The calculation assumes the position of the tool and workpiece are never the same (posX and posY never equal workpieceX and workpieceY, respectively).

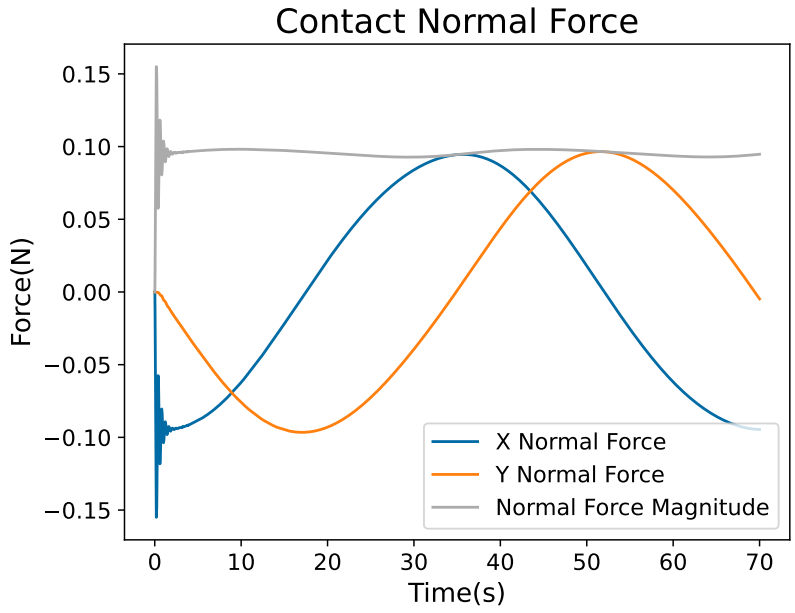

Fig. 97. Forces applied by the polishing wheel on the workpiece

The downward sloping lines in Figure 98 are the position of the actual X position of polishing wheel over time (blue) and the position output by the circular trajectory generator (grey). The polishing wheel is never far off from the intended position. The upward sloping lines are the intended Y position of the polishing wheel (dark grey) and the actual position of the polishing wheel (orange), which nearly overlap.

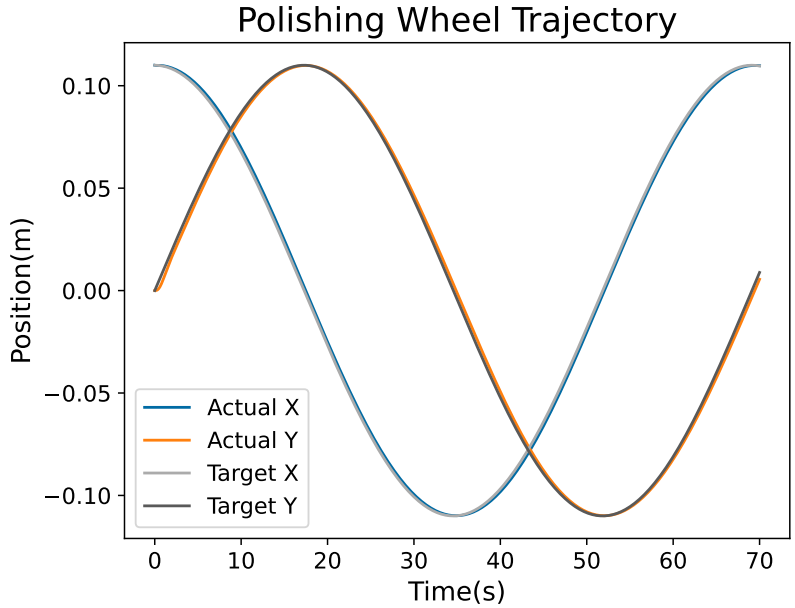

Fig. 98. Trajectory of polishing wheel and intended position of polishing wheel

6. Evaluating Interoperability

The standard translations of SysPhS to simulation platforms are expected to provide the same simulation results on all of them. To evaluate this, the models in Sections 5.1 through 5.3 [22] were translated to Simscape and Modelica by an open implementation of the standard [7][8],[24] then simulated on Simscape 10.4 and OpenModelica v1.18.0 with OMSimulator v2.1.1 [3], respectively. The same constant time step size of 0.001 seconds was used in all cases. The OpenModelica integration method was DASSL with tolerance 1e-6. The Simscape solver was local, of type backward Euler. Consistency tolerance was 1e-09. Its option for "Use fixed cost runtime consistency iterations" was set to true and 3 nonlinear iterations were used. The difference between OpenModelica and Simscape simulation results was found to be relatively negligible for the models simulated and for the time period simulated. Figure 99 shows a comparison between the arm angle with time for Simscape and OpenModelica. As can be seen in the figure there is negligible difference as there only appears to be a single line. Figure 100 shows the X position of the extruder with time for both OpenModelica and Simscape. In the figure there is no discernible difference between the two trajectories.

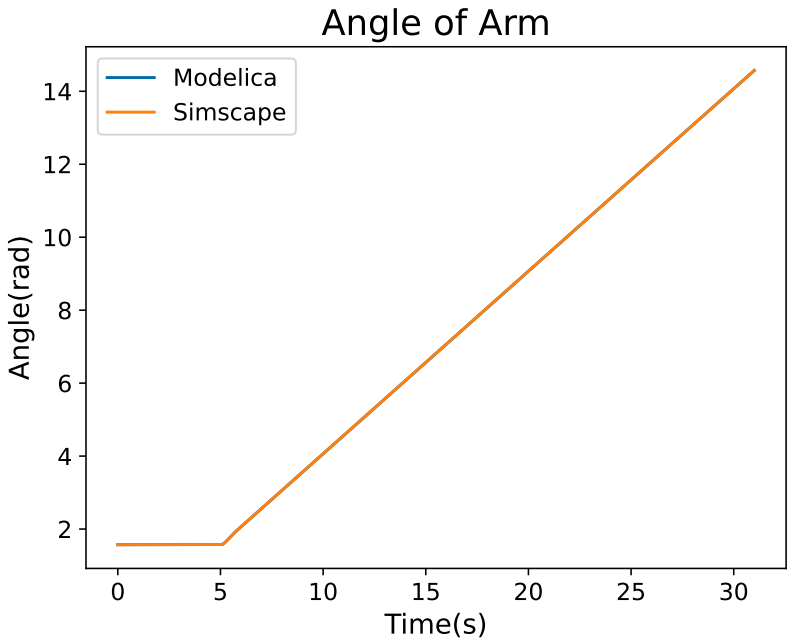

Fig. 99. Comparison of arm angle for collaborative robot

[24]The implementation required some minor fixes to work on these models and is included with them [22]. The translator takes in a SysML xmi of the system(which references the libraries), and outputs a modelica mo file for translation to modelica and outputs a simscape slx file for the system, another .slx file for a library of components and a folder for build simscape libraries in .ssc.

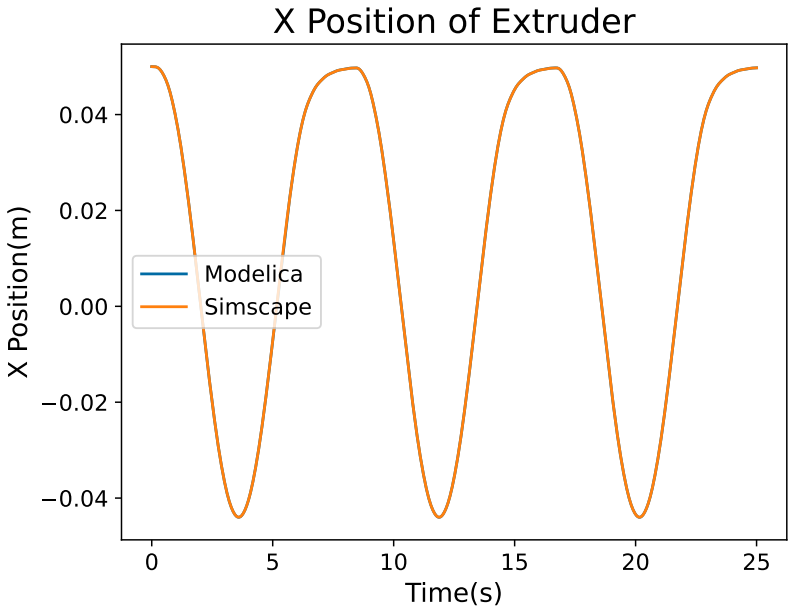

Fig. 100. Comparison of FDM extruder X position

Figure 101 shows the difference between the two trajectories with time, with Simscape's subtracted from OpenModelica. The difference between the two trajectories is in micrometers. If a smaller difference between OpenModelica and Simscape Simulations is needed, the simulation tolerance and/or timestep should be decreased.

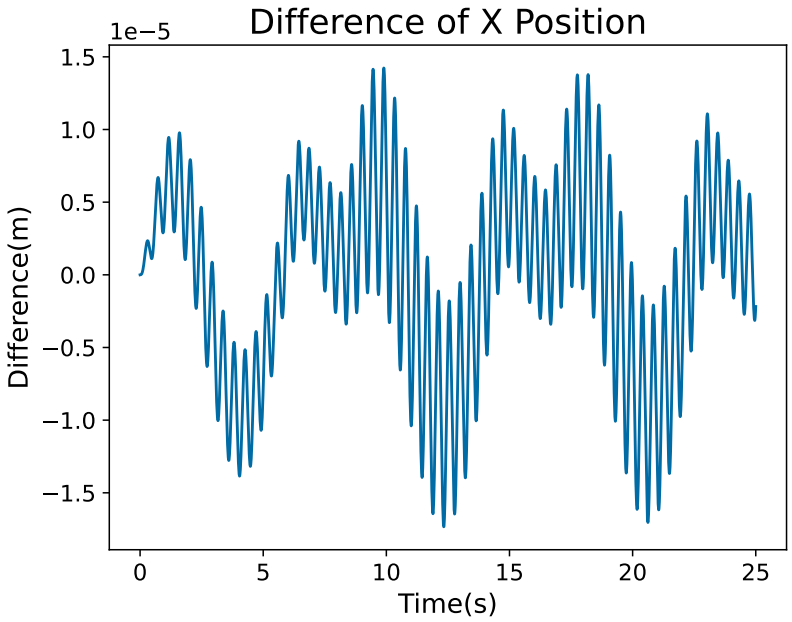

Fig. 101. Difference between trajectories of extruder in X

Figure 102 shows a comparison of the the X trajectory of the polishing head in the polishing machine produced using Open Modelica and Simscape. As can be seen in the diagram, there appears to only be one trajectory as the difference in trajectories is negligible.

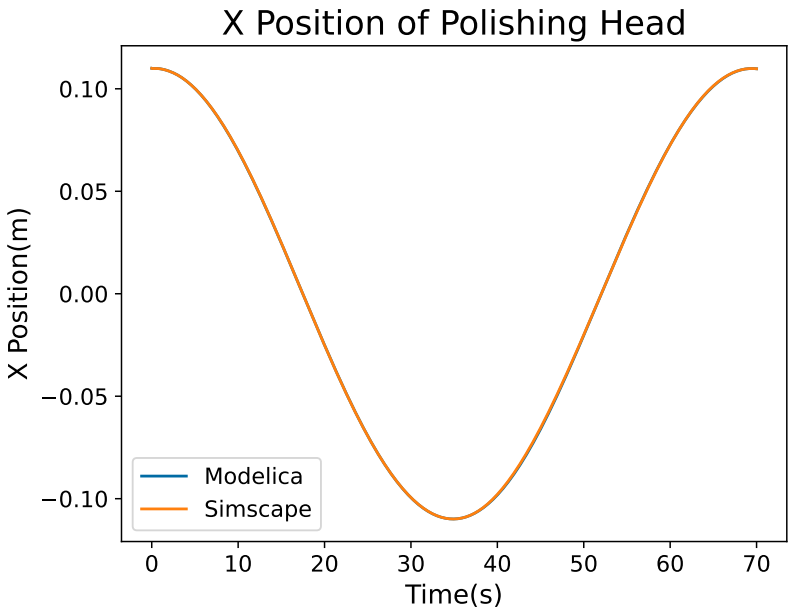

Fig. 102. Comparison of polishing wheel x position

Figure 103 shows the difference between the trajectories, with the simscape trajectory subtracted from the OpenModelica trajectory. The difference is in micrometers.

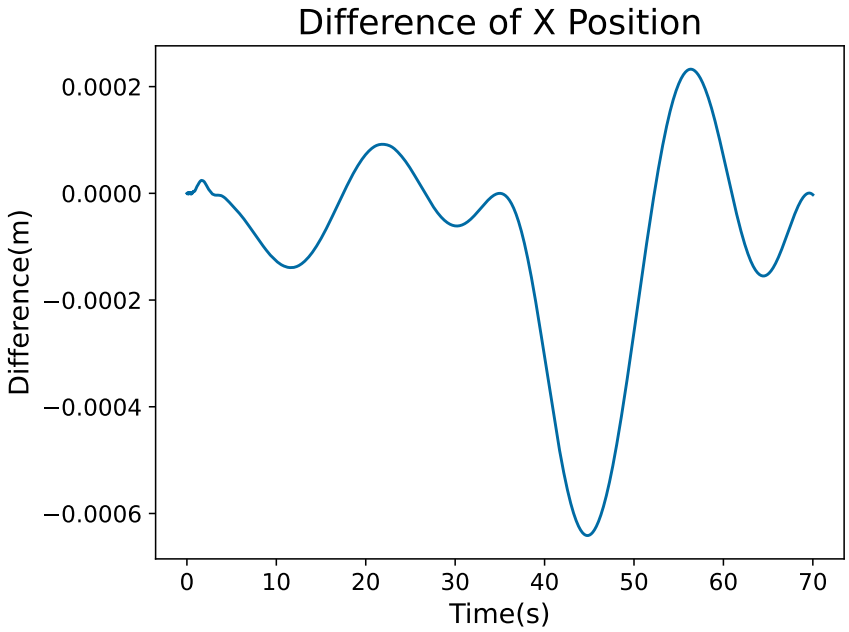

Fig. 103. Difference between trajectories for polishing machine

All models show negligible difference between the two platforms for these examples for the time period simulated.

7. Summary and Future Work

This work presents several model libraries for physical interaction modeling not currently included in SysPhs (Section 4), as well as examples of their application to manufacturing (Section 5). Libraries for signal flow, translational mechanics, rotational mechanics, and heat flow were applied to model a weight compensating robot, the mechanics and heated bed of a 3D printer, and a polishing machine. All of these libraries use SysPhs conventions for conserved substances. These examples are shown to have the same behavior on multiple platforms.

Some areas of future work on these libraries and examples are:

- Update elements of the translational and rotational library to always conserve translational or angular momentum. This could improve extensibility of the models and help users find issues in them. For example, a rotating component in a satellite might cause the satellite to point off target, due to gyroscopic effects, which is more difficult to pinpoint when momentum is not conserved. A second port could be added to source elements with an equal and opposite force applied. Gearboxes could have a third port for reaction torque.
- Related to the above, update the examples to avoid modeling translational and rotational fixed boundary conditions inside parts used under those conditions, such as an

internal component of a device that rigidly holds the device in place. For the device to remain in place it must have a physical connection to something external to it.[25]

- Update some of the example models to better reflect mechanical structure of the system being modeled. For example the inertia of axis actuators in the cartesian robot could be modeled in the same component as the rest of the actuator, instead of where the actuators are used.
- Add
  - Coulomb friction for rotational and translational libraries.
  - Lossy rotary transformers for the rotational library.
  - Control elements for the real signal library, such as high/low pass filters.
  - Detail to the manufacturing examples. The motor model could include a torque-speed curve. Nonlinear fluid flow could be added to the extruder flow model, to account for viscoelastic effects.
  - Prismatic joints in the translational library. Some aspects of the FDM and polishing machine could be simplified with prismatic joints, which enable relative translation between two components along a direction defined by a vector.
- Reduce the size of the real signal library by defining mappings of SysPhS Component Behavior Blocks to equivalent components in Simscape.

Acknowledgments

The authors thank Thomas Roth and Marcus Richardson for their helpful comments.

References

[1] Object Management Group (2019) OMG Systems Modeling Language Specification, version 1.6. Available at https://www.omg.org/spec/SysML/1.6.
[2] The MathWorks,Inc (2016) Simulink® Documentation. Available at https://www.mathworks.com/help/releases/R2016a/simulink/.
[3] Open Source Modelica Consortium (OSMC) (2022) Openmodelica users guide. Available at https://www.openmodelica.org/doc/OpenModelicaUsersGuide/latest/.
[4] The MathWorks I (2016) Simscape™ Documentation. Available at https://www.mathworks.com/help/releases/R2016a/physmod/simscape/.
[5] Object Management Group (2021) SysML Extension for Physical Interaction and Signal Flow Simulation. Available at https://www.omg.org/spec/SysPhS.

[25] In fact, zero velocity boundary conditions cannot exist, they are only approximations of a connection to an inertia massive enough such the momentum the system exchanges with it negligibly changes its velocity.

[6] Modelica Association (2021) Modelica, A Unified Object Oriented Language for Systems Modeling, Language Specification, Version 3.5. Available at https://specification.modelica.org/maint/3.5/MLS.pdf.
[7] Barbau R, Bock C, Dadfarnia M (2021) Translator from Extended SysML to Physical Interaction and Signal Flow Simulation Platforms, Version 1.1. *Journal of Research of the National Institute of Standards and Technology* 126. https://doi.org/10.6028/jres.126.027
[8] Bock C, Barbau R, Matei I, Dadfarnia M (2018) Extension of the systems modeling language for physical interaction and signal flow simulation. *Systems Engineering* 20(5):395–431. https://doi.org/10.1002/sys.21380
[9] Dadfarnia M, Bock C, Barbau R, et al. (2016) An improved method of physical interaction and signal flow modeling for systems engineering. *Conference on Systems Engineering Research (CSER 2016). Available at https://www.nist.gov/publications/improved-method-physical-interaction-and-signal-flow-modeling-systems-engineering*.
[10] Raven F (1995) *Automatic Control Engineering* (McGraw-Hill, New York, New York), 5th Ed.
[11] Object Management Group (2017) OMG Unified Modeling Language Specification, version 2.5.1. Available at https://www.omg.org/spec/UML/2.5.1/.
[12] International Organization for Standardization (2022) ISO 80000-1:2022 Quantities and units — Part 1: General. Available at https://www.iso.org/standard/76921.html.
[13] Modelica Associates and Contributors (2023) Modelica.mechanics.translational. Available at https://build.openmodelica.org/Documentation/Modelica.Mechanics.Translational.html.
[14] Karnopp DC, Margolis DL, Rosenberg RC (2006) System dynamics-modeling and simulation of mechatronic systems, john willey & sons. *Inc, Hoboken, New Jersey* .
[15] Modelica Associates and Contributors (2022) Modelica.mechanics.rotational. Available at https://build.openmodelica.org/Documentation/Modelica.Mechanics.Rotational.html.
[16] Modelica Associates and Contributors (2023) Modelica.Mechanics.Translational. Available at https://build.openmodelica.org/Documentation/Modelica.Thermal.HeatTransfer.html.
[17] Thoma JU (1975) Entropy and mass flow for energy conversion. *Journal of the Franklin Institute* 299(2):89–96.
[18] Pratt GA, Williamson MM (1995) Series elastic actuators. *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots* (IEEE), Vol. 1, pp 399–406.
[19] Paine N, Oh S, Sentis L (2013) Design and control considerations for high-performance series elastic actuators. *IEEE/ASME Transactions on Mechatronics* 19(3):1080–1091.
[20] Modelica Associates and Contributors (2023) Modelica.mechanics.translational. Available at https://build.openmodelica.org/Documentation/Modelica.Thermal.

FluidHeatFlow.html.

[21] Raibert MH, Craig JJ (1981) Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement, and Control* .

[22] Manion C, Bock C, Barbau R (2023) SysPhS Models for Physical Interaction Simulation in Manufacturing. Available at https://github.com/usnistgov/saismo/releases/tag/sysphslibs.