**NISTIR 7888**

# SysML Extension for Dynamical System Simulation Tools

Ion Matei
Conrad Bock

http://dx.doi.org/10.6028/NIST.IR.7888

National Institute of Standards and Technology
U.S. Department of Commerce

**NISTIR 7888**

# SysML Extension for Dynamical System Simulation Tools

Ion Matei
Conrad Bock
*Systems Integration Division*
*Engineering Laboratory*

http://dx.doi.org/10.6028/NIST.IR.7888

October 2012

U.S. Department of Commerce
*Rebecca Blank, Acting Secretary*

National Institute of Standards and Technology
*Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director*

# SysML Extension for Dynamical System Simulation Tools

Ion Matei and Conrad Bock

**Abstract:** Computer-interpretable representations of system structure and behavior are at the center of designing today's complex systems. Engineers create and review such representations using graphical modeling languages that support specification, analysis, design, verification and validation of systems that include hardware, software, data, personnel, procedures, and facilities, in particular the Systems Modeling Language (SysML), an extension of the Unified Modeling Language. However, SysML's constructs are insufficient to capture all details necessary for dynamic (solver-based) simulation and must be enhanced with domain specific tools for this purpose. SysML modeling tools and simulation tools are often used separately and sequentially, which reduces the efficiency of the engineering process. As a result, there is an increasing need for integrating modeling constructs specific to simulation tools into SysML. In this report, we first analyze if SysML possesses constructs that match the constructs used in simulation tools. We conclude that such constructs exist only partially and propose extensions of SysML to accommodate modeling dynamical systems. In addition, we show through an example, how the newly proposed extensions can be used to model an electrical circuit in SysML.

## 1. Introduction

Systems engineering is an interdisciplinary field of engineering focusing on how complex engineering projects should be designed and managed over their life cycles. A system engineering process is a process for applying systems engineering techniques to the development of systems. Many different systems engineering process models have been developed over the years as shown in [1]. Recognizing the need for a standard systems engineering (SE) modeling language to support the SE process, the International Council on Systems Engineering (INCOSE) initiated an effort with the Object Management Group (OMG) to extend the Unified Modeling Language version 2 (UML 2) for full-lifecycle systems engineering. UML has proven popular with software engineers since its adoption in 1997, to the point where it is now the only widely used visual modeling language for software engineering. INCOSE saw extending it as a path to a modeling language that would address the combination of hardware and software that is characteristic of modern systems. Although UML has many useful capabilities for systems engineering, its focus on software had discouraged many system engineers from adopting it in earnest [2]. Requirements were developed for a UML-based language suitable for the analysis, specification, design, and verification of a wide range of complex systems (UML SE RFP) and issued through the OMG. Industry and government responded with the Systems Modeling Language (SysML) extension to UML 2, and with updates to UML 2 during its finalization [3]. SysML is not a methodology, nor a tool, but a modeling language intended to provide support for the systems engineering process.

SysML has strong dynamic modeling capabilities, but these do not cover all the information needed to perform dynamic (solver-based) simulation. These missing capabilities are important in the design and analysis stages of the systems engineering process, where simulations, optimizations and trade-offs are performed. Currently, simulations, optimizations and trade-offs are performed using separate and dedicated simulation tools. In this paper we focus on extending the SysML with capabilities for modeling and simulation of dynamical systems; systems that are ubiquitous in modern complex systems. Simulating dynamical systems is important since it is useful to answer questions without experimenting on the real system, experiments that may be too expensive or dangerous to perform.

Most integration efforts between SysML and simulation tools were focused on Simulink® and tools based on the Modelica® language. In [4], the authors propose an extension of SysML which enables description of continuous-time behavior. They also develop its execution tool integrated on an Eclipse-based platform by exploiting co-simulation of SysML and Matlab/ Simulink. In [5], the authors explore the usage of activities that describe continuous behavior. Another integration between SysML and Simulink was done in [6] and [7], where the authors use SysML to capture the architecture of a system, and to transform the behavior representation of the architecture into a ready-to-simulate Simulink model. Similar ideas were pursued in [8], where Simulink was integrated with UML/SysML-based Rhapsody. In [9], the author presents an approach for user-interactive simulation of system models, which are created using SysML and translated into executable Modelica models. Another SysML-Modelica transformation is described in [10], where a SysML profile is defined to represent the Modelica constructs. This transformation was standardized by the Object Management Group and the specification can be found in [11]. We would like to point out that some previous integration efforts were made with respect to the SysML version 1.2. The new specifications of SysML version 1.3 contain some differences with respect to the ports and flow properties, compared to version 1.2. In addition, the integration efforts are focused on a specific language (Matlab or Modelica). This may be problematic when models created using different modeling languages need to be integrated in SysML models.

In this paper, we propose an integration between SysML and simulation tools for dynamical systems that is independent of the particular simulation tools involved. We extend SysML with general concepts for all simulation tools for dynamical systems, rather than focusing on a specific simulation language. The advantage of this approach is that the extension is general enough to be applied to different simulation languages. To that end, we use a simulation tool abstraction proposed in one of our previous papers [12]. We identify the SysML constructs that are the best match to simulation constructs and extend some of the SysML constructs to match simulation semantics.

The structure of this paper is as follows. In Section 2 we present a short overview of SysML. Section 3 introduces the main constructs used to model dynamical systems in simulation tools. In Section 4 we identify the SysML constructs that are the closest to the simulation constructs and compare them. Section 5 contains an extension of SysML constructs to exactly match the simulation tools constructs. We end the paper with an example and some conclusions.

## 2. Overview of SysML

Systems engineers use a wide range of modeling languages, tools, and techniques on large systems projects. As outlined in Section 1, the need for a unified approach in modeling large systems led to SysML, published by OMG. SysML is designed to provide simple but powerful constructs for modeling a wide range of systems engineering problems. It is particularly effective in specifying requirements, structure, behavior, allocations, and constraints on system properties to support engineering analysis. SysML reuses a subset of UML 2 and provides additional extensions to represent requirements and constraints on the system. A high level overview of the interrelationship between SysML and UML is shown in Figure 1.

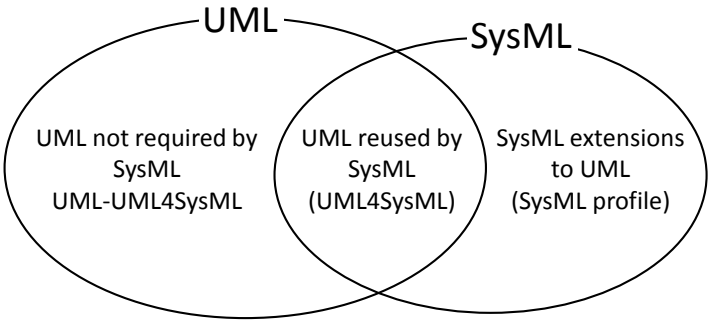



Figure 1– Overview of SysML/UML interrelationship **[3]**

SysML includes diagrams that can be used to specify system requirements, behavior, structure and parametric relationships, also known as the pillars of SysML. The system structure is represented by block definition diagrams and internal block diagrams. A block definition diagram describes the system hierarchy and system/component classifications. The internal block diagram describes the internal structure of a system in terms of its parts, ports, and connectors. The package diagram is used to organize the model. The behavior diagrams include the use case diagram, activity diagram, sequence diagram and state machine diagram. A use-case diagram provides a high-level description of the system functionality. The activity diagram represents sequences of actions and flow of items (information/energy) between them. An interaction diagram represents the interaction between collaborating parts of a system. The state machine diagram describes the state transitions and actions that a system or its parts performs in response to events. The requirement diagram captures requirement hierarchies and the derivation, satisfaction, verification and refinement relationships. The relationships provide the capability to relate requirements to one another and to relate requirements to system design models and test cases. The requirement diagram provides a bridge between typical requirements management tools and the system models. The parametric diagram represents constraints on system property values such as performance, reliability and mass properties to support engineering analysis. Figure 2 summarizes the SysML diagram taxonomy, emphasizing what is new in SysML compared with UML [2].

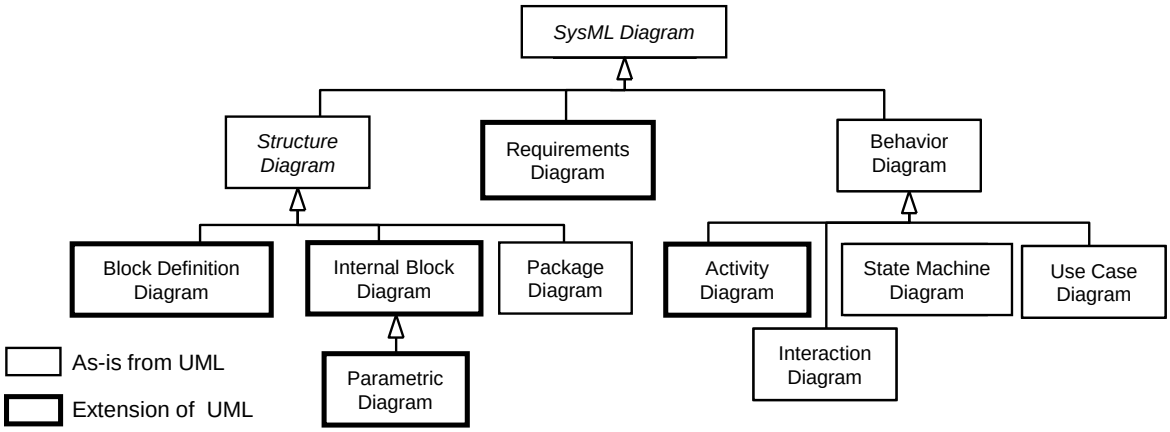



Figure 2- SysML diagram taxonomy **[3]**

## 3. Modeling dynamical systems in simulation tools

This section contains a brief description of the main modeling methodologies and constructs used in simulation tools. A more comprehensive presentation of this section can be found in [12]. The terminology of this section is suitable for understanding simulation tools, but differs from SysML terminology, including different meanings for the same terms, as explained in Section 4.

Simulation requires a model of a system, which is used to answer questions without experimenting on the real system. A model can be seen as a simplified system that reflects some properties of the real system. The primary modeling methodologies used for simulation are *signal-flow* and *physical-interaction modeling*. In signal-flow modeling, the relationship between system components is unidirectional, from outputs of one component to inputs of others. In physical-interaction modeling physical conservation laws are used and the relationship between system components is bidirectional.

Modeling in a simulation tool is based on four basic constructs: components, ports, links and block diagrams. *Components* are the structural/behavioral elements of a system, which through their interaction create the model of the system. *Ports* are a way for components to interact (exchange energy/information) with other components. *Links* are connections between ports of components, across which energy/information is exchanged. A *subsystem* is a component formed of other components and/or subsystems. Subsystems show the hierarchical nature of a system or simplify the model, when the number of components is significantly large. Components that cannot be decomposed are called *atomic*. A *model* can be viewed as a component formed of components and/or subsystems. *Block diagrams* contain components, subsystems, ports and links, and are used to describe the model of a system, that is, its components and the interactions between them.

The behavior of a system is determined by the behavior of its components and the interactions between them. The behavior of atomic components is represented using mathematical models, and in particular, differential equations.

## Components

Components consist of subcomponents, variables and parameters. Subcomponents can be other components or ports. Variables represent quantities that can change in time, while parameters represent quantities that remain constant during each simulation.

The behavior of a component that is not a subsystem is given by a mathematical model describing the mathematical relationships (equations) between the variables and the parameters of the component. Mathematical models can be dynamic,[1] stationary, continuous,[2] discrete[3] or hybrid. Dynamic mathematical models implicitly or explicitly include time, causing some variables of the model to vary. Static models are used to represent steady-state or equilibrium regimes. Variables of continuous time models evolve continuously while variables of discrete-time models change only at discrete time instants. Models with both continuous and discrete components are called hybrid.

The most general mathematical models that simulation tools use are differential algebraic equations (DAEs). Formally, a continuous-time DAE can be expressed as

$$f(\dot{x}(t), x(t), u(t), y(t), t, \theta) = 0, \qquad x(0) = x_0$$

where $u(t), y(t)$ are the input and output vectors, respectively, $x(t) = \big(x_1(t), x_2(t), \dots, x_n(t)\big)$ is a vector of dependent variables also known as state vector, $f = (f_1, f_2, \dots, f_m)$ is a function-valued vector and $\dot{x}(t)$ is the derivative of the state vector. In the above equations time is denoted by $t$ and $\theta$ refers to the (constant) parameters of the model. DAEs differ from ordinary differential equations (ODE) in that a DAE is not completely solvable in closed form for the derivatives of all components of the vector of variables $x$.

We can have discrete versions of the above equations that can be expressed as

$$g(x(k+1), x(k), u(k), y(k), k, \theta) = 0, \qquad x(0) = x_0$$

where $k$ is the discrete time, $u(k), y(k)$ are the input, output vectors, respectively, $x(k) = \big(x_1(k), x_2(k), \dots, x_n(k)\big)$, $g = (g_1, g_2, \dots, g_m)$ and $x(k)$ is updated at discrete time instants. More generally, by combining the continuous and discrete equations, we obtain hybrid mathematical models for describing systems. In first principle modeling methodology, we find useful to identify the *conserved* variables and the *non-conserved* variables of a system.[4] Generally speaking, the *non-conserved* variables represent the driving force in the system and the *conserved* variables represent rates of flow. Note that the product of the conserved variables and non-conserved variables typically has the units of power.

[1] The state of a dynamical system changes with time, unlike the case of stationary systems.
[2] A variable $x$ (depending on time) is continuous if for every time instant $t_0$, the limit of $x(t)$ as $t$ approaches $t_0$ exists and is equal to $x(t_0)$.
[3] The time evolution of a discrete variable is piecewise constant.
[4] Conserved variables are also known as through variables. Non-conserved variables are also known as across variables.

## Ports

Components interact with other components through *ports*. Ports have variables describing aspects of the energy or information exchanged with other components, with exactly one kind of energy or information for each port. For example, variables on a port for electrical flow might give voltage and current (the driving force and rate of flow, respectively). Each electrical flow is specified by a separate port, because the voltage and current might be different for each flow. In the case of signal-flow modeling, ports specify which variables are inputs and which variable are outputs, and are commonly referred to as *input* or *output ports*, and collectively as *directional*.[5] In the case of physical-interaction modeling, ports still specify which variables interact with other blocks, but not whether they are inputs or outputs, and are referred to as *bidirectional*.[6]

## Links

As mentioned in the previous section, the first step in building a model is to identify the main components of the system and how they interact. The interactions between components induce the behavior of the model. Connecting components actually means connecting the interfaces (ports) of the components using links. Depending on the modeling methodology, we distinguish two types of links: signal-flow links[7] and physical interaction links.[8]

In signal-flow links [12], the variables belonging to the ports are designated as inputs or outputs. Interconnecting components means connecting the outputs of some components to the inputs of other components, under some constraints: an input can only be connected to one output; an output cannot be directly connected to the input of the same component; an output can be connected to several inputs. From the semantic point of view, when an output is connected to an input, the output value assigns the same value to the connected input. Therefore the output and the input to which it is connected must have compatible type and the same dimension in case they are vectors.

Physical-interaction links are used in physical-interaction modeling, where components are connected bidirectionally. We distinguish two types of variables present on ports of bidirectionally-connected components: *conserved* and *non-conserved* variables. Bidirectional links between ports have the following semantics: conserved variables sum up to zero, and non-conserved variables must be equal. Obviously, the variables must match, that is, they must have compatible types and the same dimension if they are vectors.

## Block diagram

Block diagram is used to represent the model of a system, that is, its composing components, ports and links. The behavior of a system is dictated by the behavior of its components and by the interconnections. Therefore, in addition to the structural aspect, a block diagram describes behavioral aspects of a system, as well.

[5] Directional ports are also known as causal ports.
[6] Bidirectional ports are also known as acausal ports.
[7] Signal-flow links are also known as causal links.
[8] Physical-interaction links are also known as acausal links.

## 4. SysML constructs and simulation tools constructs compared

In this section we give a brief description of a set of relevant SysML constructs and analyze if and how they can be used to represent simulation tools constructs [3]. We perform the analysis from both structural and behavioral perspectives.

### 4.1 Structural constructs

Among the structural constructs of SysML relevant for representation of simulation tools constructs, we enumerate: block, connector and internal block diagram.

SysML blocks extend the UML structured classes and represent any level of the system hierarchy including the top-level system, a subsystem, or logical or physical component of a system or environment. SysML blocks provide a general-purpose capability to describe the architecture of a system. They provide the ability to represent a system hierarchy, in which a system at one level is composed of systems at a more basic level. They can describe not only the connectivity relationships between the systems at any level, but also quantitative values or other information about a system. Using the properties of a SysML block we can specify the subcomponents of a block, while Internal Block Diagrams (IBDs) are used to describe how the subcomponents interact with each other (*internal structure*). The features of blocks can be represented using properties. Among the most important properties we describe below: parts, references, values, port and constraint properties.

It naturally follows that the correspondent of a simulation tool model in SysML is a block with internal structure (a SysML block that contains part properties, port properties and connectors, in addition to the properties pertinent to the behavior description). The Internal Block Diagram corresponds to the block diagram in a simulation tool and depicts the internal structure of the block. Similarly, a subsystem corresponds to a SysML block with internal structure. An atomic block corresponds to a SysML block without internal structure, but with properties relevant for the description of the block's behavior.

Parts are properties typed by SysML blocks and describe the decomposition of a block into its constituent elements. Hence it follows that the blocks typing these properties correspond to the components of a model. Reference properties are used on a block definition diagram to capture relationships between blocks other than decomposition. Value properties describe the quantifiable characteristics of a block, such as weight or velocity, and therefore we may be inclined to consider them to represent variables and parameters of a simulation tool component.

In SysML the word "port" is used to designate a set of any kind of features available to other system components, rather than a specification of a single information or energy exchange as it is the case in simulation tools. SysML port properties are properties with a type that specifies features available to the external entities via connectors to the ports. Ports are points at which external entities can interact with a block in different or more limited ways than interacting directly to the block itself. Figure 3 depicts SysML block diagram presenting several blocks with examples of properties. Figure 4 shows an internal block diagram with examples of ports on blocks.

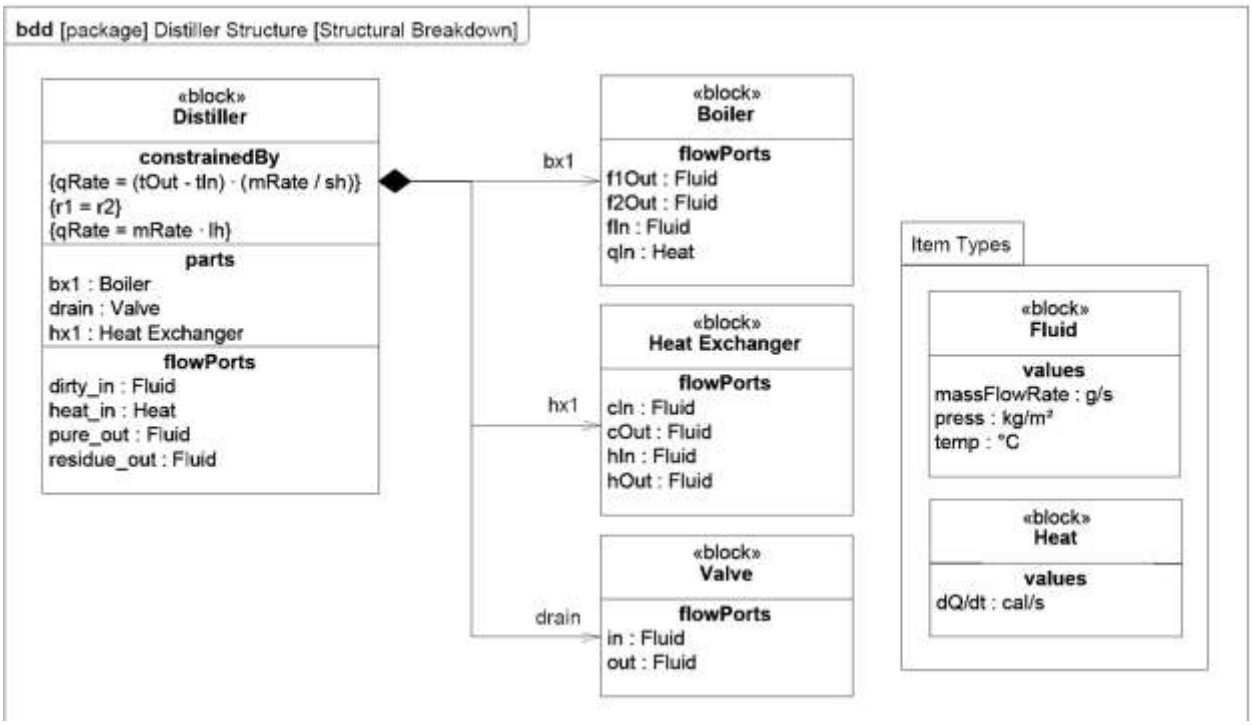



Figure 3 – SysML Block Definition Diagram **[2]**

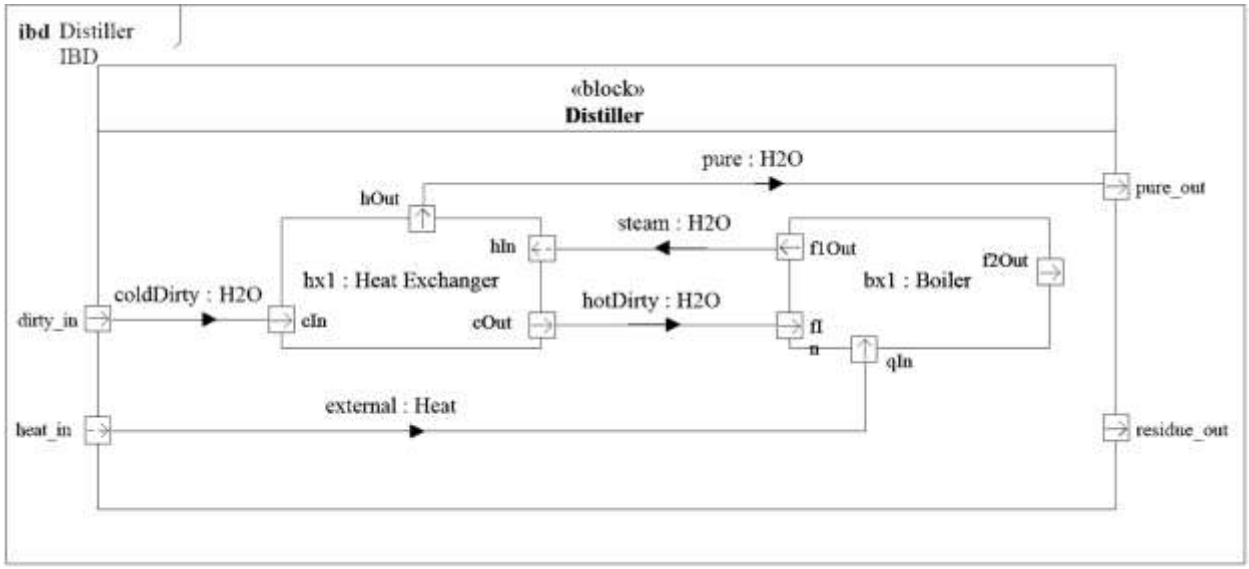



Figure 4 – SysML Internal Block Diagram **[2]**

Connecting two SysML port properties implies that there exists some relation between the ports. In simulation tools, links have additional mathematical constraints. We can use SysML connectors to represents links with mathematical constraints added for the ports being connected. As previously mentioned, a SysML block can have value properties that describe the quantifiable characteristics of a block, such as weight or velocity. However, SysML does not directly

distinguish between parameters and variables on the one hand, and between continuous, discrete, conserved and non-conserved variables on the other hand (SysML uses the terms "continuous" and "discrete" in activity semantics, but not in with the same meanings as simulation tools, see Section 3). Therefore, we need new constructs that can distinguish between these kinds of value properties.

Blocks typing port properties of a SysML block represent components of that block available to external entities. They specify properties that are accessible to other blocks. Some of these properties might be *flow properties*, which specify the kinds of items that might flow between instances of a block and its environment, whether it is information, material, or energy. Ports specify the kind of item flowing with the property type, which means the values of flow properties are the portions of information, material, or energy flowing in or out of an instance of the block at a single point in time. Flow properties can have direction (in, out or inout) and if all flow properties defined on a port have the same direction, then the port can be considered to be an input, output or input and output port. Note however, that we may want to specify that a current is an input for a simulation block describing an electrical circuit. But the items that flow between a SysML block would be in this case the electrical charge and not the current (current is the rate of flow of electrical charge). Therefore, using only flow properties to specify simulation variables is insufficient.

Connectors in SysML are used to specify relationships between parts of the same containing block, and therefore can be used to represent a system hierarchy. Figure 5 shows a connector connecting the ports of the blocks WaterSupply and WaterClient as they are used in the internal structure of a house. The connector is typed by the association waterDelivery. Flow can occur between connected ports if flow properties at each end match in their type and direction. For example, flow properties with direction out and type Water match flow properties with direction in and type Fluid. Links in simulation tools have the same role as SysML connectors. However, SysML cannot completely characterize the type of connections found in signal-flow and physical interaction methodologies, which depend of the type of variables defined on ports. A special kind of connector, called binding connector, is used to specify that the properties at both ends of the connector have equal values. A similar phenomenon happens in the case of directional links, but a binding connector does not give any information about the direction of the ends of the connector, that is, whether they are inputs or outputs.

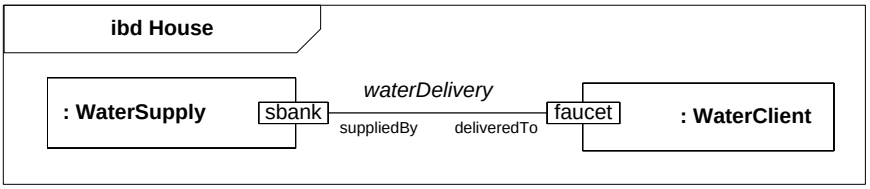



Figure 5 – Connector typed by the "waterDelivery" association **[3]**

A constraint property is a property of a block that is typed by a constraint block. The main advantage of using constraint blocks comes from reusability, that is, constraint blocks define generic forms of constraints that can be used in multiple contexts. A constraint block includes the constraint, such as {F=m*a}**,** and the parameters of the constraint such as F, m, and a (SysML

uses the term "parameter" in a different sense than simulation tools). SysML constraint parameters are properties of constraint blocks. They are bound to properties of other blocks in a surrounding context where the constraint is used. Parametric diagrams include usages of constraint blocks to constrain the properties of another block. The usage of a constraint binds the parameters of the constraint, such as F, m, and a, to specific properties of a block, such as a mass, that provide values for the parameters.

Since differential equations modeling the behavior of simulation tools components can be seen as mathematical constraints, constraint properties are good candidates to represent them in SysML.

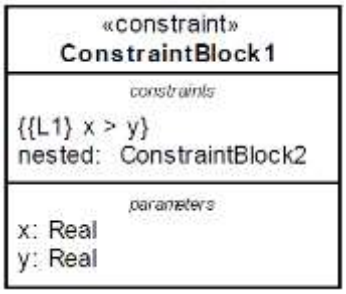



Figure 6 –SysML Constraint Block **[3]**

Constraint blocks are used in the context of Parametric Diagrams (Figure 7), which are particular types of SysML diagrams used to express mathematical relationships between constraint parameters. They contain a particular type of blocks, called Constraints Blocks, and connections between them. From this point of view they serve the same purpose as block diagrams in simulation tools.

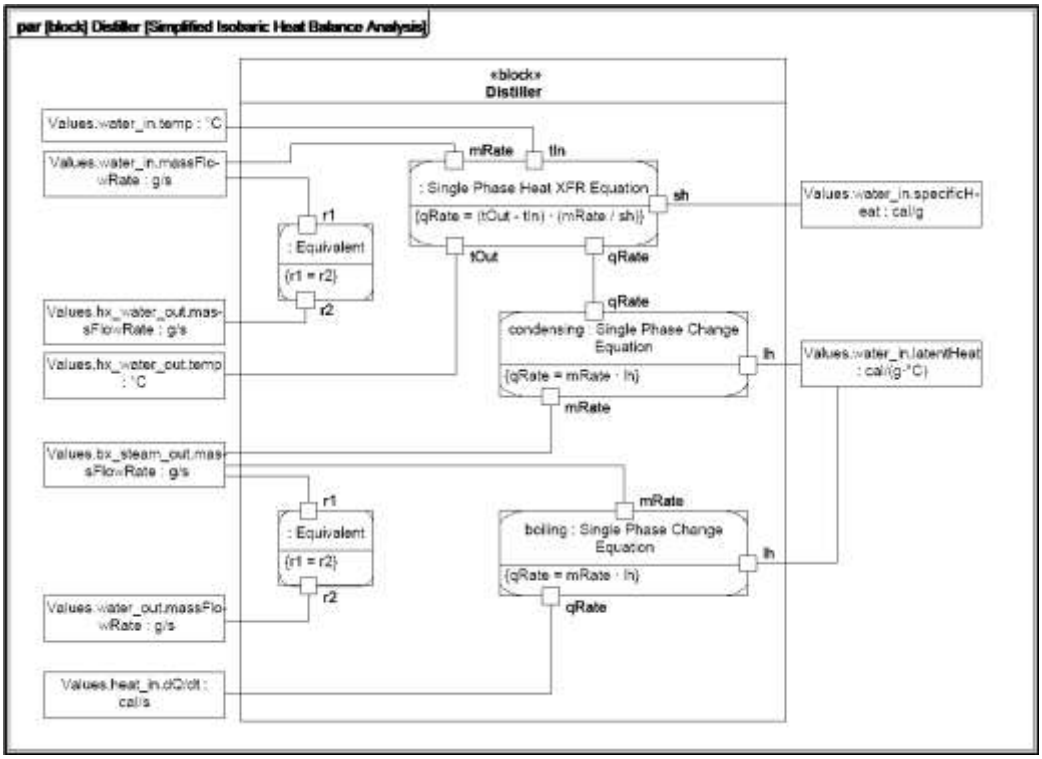


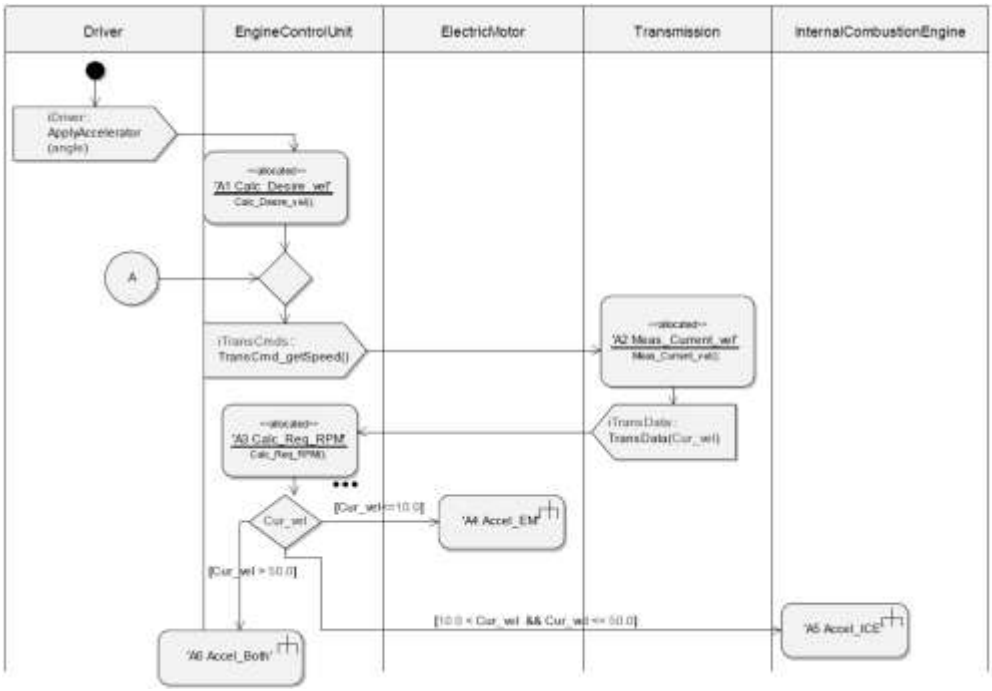

Figure 7 – Example of Parametric Diagram **[2]**

However, there are several drawbacks with this approach. As pointed out in [10], in Parametric Diagrams, there is no notion of ports. The closest thing that graphically mimics ports is the constraint parameter (the Reals x and y in Figure 6). However, Constraint Parameters are not actual ports in the sense defined in SysML or simulation tools, but graphical representations (rectangles) of constraint parameters. Additionally, constraint blocks accept only parameters as properties and hence we cannot distinguish between variables and parameters of a simulation tool component.

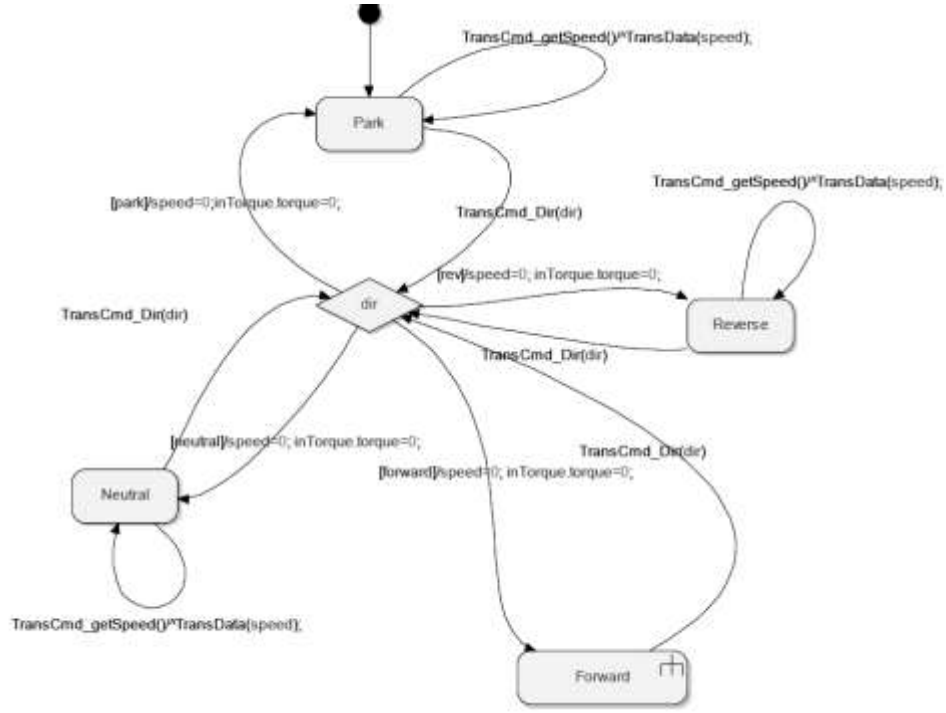
Binding connectors are used to create complex mathematical relationships by connecting constraint parameters. The semantics of binding connectors is mathematical equality, similar to the semantics of links used to connect non-conserved variables in simulation tools block diagrams. Unfortunately, binding connectors do not support the semantics for connecting conserved variables, such as the Kirchhoff law for currents, and do not include direction (inputs/outputs), as needed for the signal-flow modeling methodology.

### 4.2 Behavior constructs

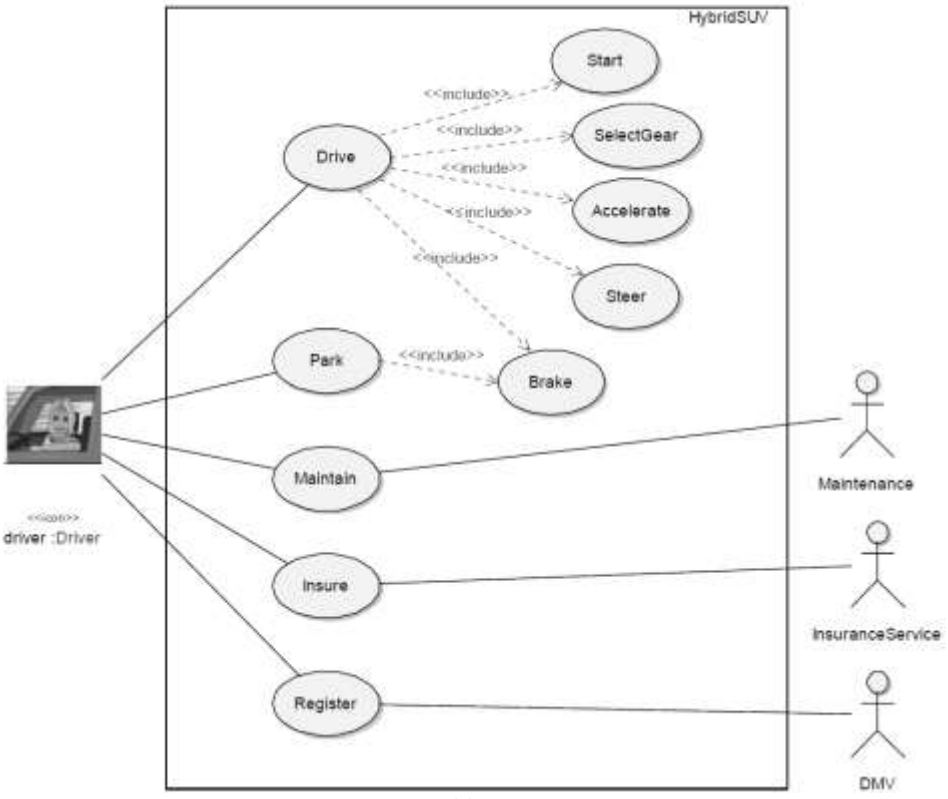
The behavior of a SysML block can be specified using activities, interactions, state machines and use cases (see Figure 8). Activities describe the sequences of actions and flow of inputs and outputs among actions. Interactions define message-based behaviors. State Machines specify state-based behavior in terms of system states and transitions between them, while use cases describe behavior in terms of the high-level functionality and uses of a system, that are further specified in the other behavioral diagrams referred to above. We study if these constructs for describing behavior are suitable to describe the behavior of blocks used in simulation tools.



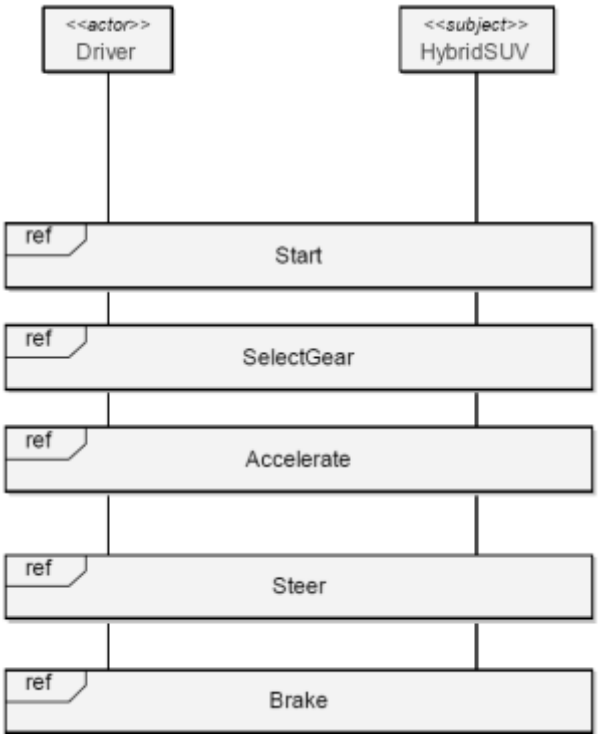

(a)


TransCmd_getSpeed()/*TransData(speed);
Park
[park]/speed=0;inTorque.torque=0;
TransCmd_Dir(dir)
TransCmd_getSpeed()/*TransData(speed);
[rev]/speed=0; inTorque.torque=0;
dir
Reverse
TransCmd_Dir(dir)
TransCmd_Dir(dir)
[neutral]/speed=0; inTorque.torque=0;
TransCmd_Dir(dir)
[forward]/speed=0; inTorque.torque=0;
Neutral
TransCmd_getSpeed()/*TransData(speed);
Forward

(b)


HybridSUV
Start
<<include>>
<<include>>
Drive
SelectGear
<<include>>
Accelerate
<<include>>
<<include>>
Steer
Park
<<include>>
Brake
Maintain
Insure
Register
driver :Driver
Maintenance
InsuranceService
DMV

(c)




(d)

Figure 8 – (a) Activity diagram; (b) State machine; (c) Use case; (d) Interaction diagram **[6]**

Use cases in SysML are the behavior construct farthest from modeling in simulation tools because they only represent the breakdown of interactions between systems and their operators. For example, in Figure 8c, the interaction between a person and car for driving is broken down into interactions for starting, steering, and so on. Any detail about these interactions is specified by interaction diagrams, see next.

Interactions in SysML specify the time-ordering of items passing between components in a system (or between systems and their operators, as in use cases above). For example, in Figure 8d, in the interaction between a person and car for driving, starting occurs before selecting the gear, which happens before accelerating, and so on. Interaction models are limited to discrete and finite flows between components, because each flow is separate element in the model, shown graphically as a line between the components. They cannot represent the behavior of typical continuous dynamical systems. Interactions can only represent inputs and outputs between two components in the "request-reply" pattern, not inputs from multiple components that result in outputs to multiple other components as often occurs in dynamical systems.

States machines in SysML describe the state dependent behavior of a block in terms of its states and the transitions between them. It is apparent that SysML state machines are suitable for modeling systems with discrete states. Unfortunately, the number of discrete states must be finite and therefore SysML state machines are not suitable even for the simplest discrete linear systems that can have an infinite number of states. We recall that the most general dynamic systems that a simulation tool block can represent are hybrid systems. Hybrid systems are characterized by continuous and discrete states. For simplicity let us assume that the number of discrete states is finite so that it can correspond to a state of a SysML state machine. In each of

these discrete states the continuous states evolve according to some continuous time dynamics. SysML state machines do not provide direct ways to represent continuous dynamics in a particular state. The discrete state may change as a result of an event generated by the continuous dynamics corresponding to that specific state. SysML can model this with ChangeEvents, which detect changes at any time, but there's no standard way to specify the change to be detected in the case of general mathematical models needed for dynamic systems.

Activities in SysML are probably the behavior construct closest to modeling in simulation tools. This is because an activity specifies the transformation of inputs to outputs through a controlled sequence of actions, which can be continuously flowing. The activity diagram is the primary representation for modeling flow-based behavior and is analogous to the functional flow diagram used for modeling systems [5].

As in the case of the rest of SysML behavior constructs, an activity (or action) does not have direct means to describe a mathematical model associated with simulation tool blocks. As we know, a mathematical model is characterized by variables and parameters and therefore we would need a mechanism to specify them in an activity.

Activities are built using actions, which describe how activities execute. Each action can accept inputs and produce outputs called tokens. The token are placed on input or output buffers called pins, until they are ready to be consumed. Streaming pins can continue to accept tokens while nonstreaming pins only accept and produce tokens at the beginning and end of the execution.

The activities (and their usages as actions) work with inputs and outputs to describe what tokens are received and produced. Therefore, activities are not appropriate to model behavior in physical-interaction modeling methodology, with direct implication in modeling the connection semantics of simulation tools blocks.

### 4.3 Conclusions on mapping between SysML and simulation tool constructs

From the analysis presented in the previous sections we note that SysML provides constructs that partially correspond to the constructs used in simulation tools. These constructs are summarized in Table 1. However, there are significant semantic differences between them. Among the main differences we found are:

- SysML value properties cannot distinguish between variables and parameters, and between continuous, discrete (in the sense simulation tools use these terms), non-conserved and conserved variables.
- SysML can use flow properties to specify direction of simulation tool ports. However, the semantics of flow properties do not completely match the semantics of properties of simulation tool ports, since the former specify the kinds of things that flow, while the latter do not, and the latter specify other characteristics of flow, such as driving force and flow rate, while the former do not.
- The semantics of SysML connectors provide some of the semantics for links used in signal-flow modeling (with binding connectors), but not the direction of signal-flow, and none of the semantics for links used in physical-interaction modeling.
- The SysML behavioral constructs do not match the behavior constructs in simulation tools, but using constraints properties may be a solution.

| Simulation tools constructs | | SysML constructs |
|---|---|---|
| Model | | Block with internal structure, but not a component of other blocks |
| Component | Atomic component | Block without internal structure |
| | Subsystem | Block with internal structure |
| Links | | Connectors |
| Ports | | Flow properties |
| Dynamic equations | | Constraint blocks typing constraint properties |

Table 1 – SysML and simulation tools constructs mapping

In conclusion, we need to extend the aforementioned SysML constructs with new semantics such that they can match the semantics of simulation tools constructs.

## 5. Extension of SysML constructs for modeling dynamical systems

As emphasized in the previous section, the semantics of the closest SysML constructs that correspond to the constructs used in simulation tools do not match exactly. In the following we introduce extensions of SysML constructs, so that dynamical systems can be modeled.

We emphasized earlier that SysML does not distinguish between parameters and variables, and between continuous and discrete variables (in the sense simulation tools use these terms, see Section 3). As a consequence we introduce a new stereotype named SimConstant which extends Property from UML4SysML (the SysML constructs that are from UML, as shown in Figure 1). This stereotype is used to specify that the value remains constant during each simulation execution (the stereotype name does not use the term "parameter" because it has different meanings in SysML than simulation).[9] The abstract syntax of this stereotype is shown in Figure 9.

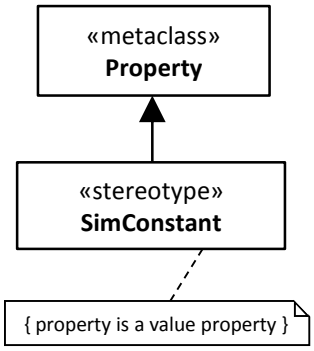



Figure 9 - SimConstant abstract syntax

[9] The term "constant" in simulation tools refers to values that are the same across all simulation executions, such as the physical constant for gravitational acceleration, rather than just during each simulation execution separately ("parameter"). Simulation constants can be modeled in SysML as read-only properties with default values.

In addition, we introduce SimVariable, that again extends Property from UML4SysML. We use this stereotype to specify that a property is a simulation tool variable (that is, its value will be used by simulation tools and might change over time). The abstract syntax of this stereotype is presented in Figure 10, and has the following attributes:

- isContinuous : Boolean = true
  If the attribute is true the variable is *continuous*. If the attribute is false, the variable is *discrete*.[10] The default value is true.

- isConserved : Boolean = false
  If the attribute is false the variable is *non-conserved*. If the attribute is true the variable is *conserved*. The default value is false.

- changeCycle: Real = 0
  The attribute specifies the time cycle at which a discrete variable (isContinuous = false) might change values. During each cycle, the value must be constant. The value might change at the end of each cycle, but not necessarily. In the case of continuous variables, the attribute takes value zero, which means the value might change at any time.

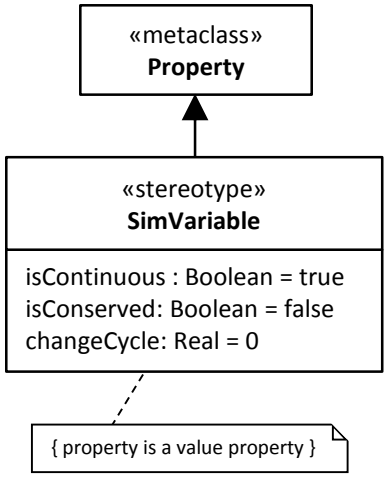



Figure 10 – SimVariable abstract syntax

SimConstants and SimVariables can be applied to any property to indicate it is relevant for simulation tools, but SimVariables can characterize interactions between components, while SimConstants cannot, because interactions always have the potential to change property values of components. As a consequence, we propose a special kind of SysML Block named SimBlock that has properties that are all SimVariables. This kind of block has the constraint that all properties have the SimVariable stereotype applied. SimBlocks are particularly useful for describing interactions between components from the viewpoint of simulation tools, but can also be used to describe components themselves if needed. The abstract syntax of this block is shown in Figure 11.

[10] This is continuous and discrete in the sense given in footnotes 2 and 3 (Section 3 under Components), respectively, rather than continuous in the SysML sense of zero time between items flowing in activity models, see Section 1.4.2.

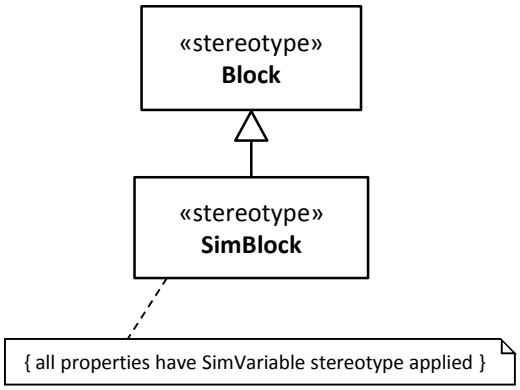



Figure 11 – SimBlock abstract syntax

As described in Section 4.1, SysML has a special type of property indicating direction of flow (flow properties). However, these properties only designate the kind of items that flow, rather than the characteristics of the flow, such as driving force or flow rate, whereas simulation tool ports can contain variables representing non-conserved and conserved quantities describing these aspects of the flow. Still we argue that an extension of flow properties can be used to specify simulation tool ports. The reason is that in any type of modeling (signal-flow or physical-interaction) there are items that flow between ports, although these items are not directly represented by variables at the level of simulation tool ports. For example, in the case of physical-interaction modeling when currents and voltages are port variables, we can think at electrons as the items flowing on links, or more generally, we can think of electricity. Similarly, in the case of signal-flow modeling we can think of electrical signals or information flowing on links connecting ports.

Next, we propose a stereotype named SimProperty, extending Property from UML4SysML. The abstract syntax of this stereotype is shown in Figure 12. Properties with this stereotype applied are always typed by a UML Class with the SimBlock stereotype applied. The SimProperty stereotype has an attribute *referTo* that identifies a flow property used to specify the direction of the flow. The value of properties that have SimProperty applied are the instantaneous flows through the flow properties they refer to (see Section 4.1 about the values of flow properties). For simplicity, if the direction of the flow property given by the attribute referTo is in or out, we say that the usage of the SimBlock typing the SimProperty has input, output direction respectively. In the case the direction is inout, we say that the usage of the SimBlock typing the SimProperty is bidirectional. We have the following constraints on the SimProperties of SimBlocks:

a) A SimBlock with direction input or output can have only non-conserved variables.

b) A bidirectional SimBlock can have both non-conserved and conserved variables.

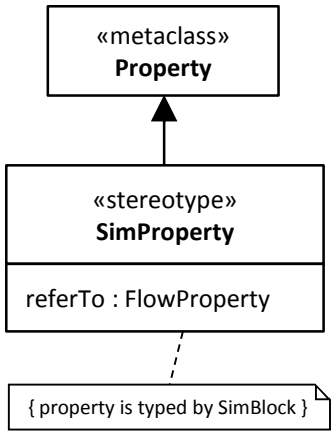



Figure 12 – Abstract syntax for SimProperty

SimProperties can be used as properties of blocks typing port properties, in which case they refer to flow properties of the block typing the port property. We can connect the ports using SysML connectors. For simplicity, if flow properties on port types at each end of connector match (see Section 4.1 about matching flow properties), we say SimBlocks are connected if they type SimProperties referring to matching flow properties on the types of the connected ports, and we say SimBlocks have a direction in their usage as types of SimProperties referring to flow properties that have direction. We have the following constraints on connected SimBlocks:[11]

a) The SimVariables of connected SimBlocks must match, that is, they must have the same type, the same name, and their number must be the same.

b) A SimBlock with output direction can be connected to one or several ports matching SimBlocks with input direction. However, a SimBlock with input direction can be connected to only one matching SimBlock with output direction. During simulation, the variables of the block with input direction will be assigned the values of the matching variables in the SimBlock with output direction.

c) A bidirectional SimBlock can be connected to one or several matching bidirectional SimBlocks, for both input and output direction. During simulation, the values of the matching non-conserved variables of the connected blocks must be equal, while the values of all matching conserved variables must sum up to zero.

SimVariables on the same SimBlock are typically typed in non-overlapping ways to avoid redundancy. For example, if one SimProperty is typed by voltage, there won't be another SimProperty on the same SimBlock typed by voltage, because they would have the same value for the flows they describe. Simulation ports sometimes have multiple variables with the same type, where each variable is interpreted as a different flow. In the extension proposed here, this would be modeled in SysML with multiple SimBlocks, one for each of the variables of the same

[11] These apply to connectors between any property typed by block containing SimProperties, but this case is not covered here.

type, and each of SimBlocks would describe a separate flow, through separate flow properties (see next about relating SimBlocks to flow properties).

For clarity, we present the following example. For visual clarity, we introduced a new compartment named SimProperties that contains properties typed by SimBlocks.

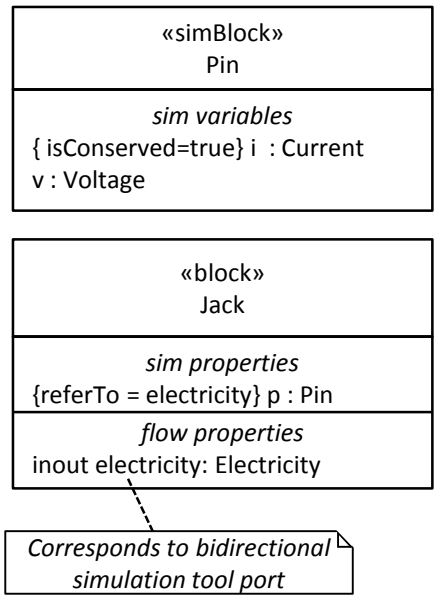



Figure 13 – Example of modeling bidirectional simulation tool ports in SysML

Figure 13 shows how we can use the SysML constructs defined above to model a simulation tool bidirectional port. We first define the SimBlock Pin containing the conserved variable i and the non-conserved variable v. A usage of Pin is in the block Port as the type of SimProperty p, which in addition contains the flow property electricity. Note that the attribute referTo of the SimProperty is the flow property electricity, through which direction of p and its usage of Pin is determined. In this example, the direction of p is inout and the Pin SimBlock is bidirectional.

Figure 14 shows how we can model an output port using the extended SysML constructs. As in the previous example, we use a flow property with direction out. Note that in this case, the SimBlock contains only non-conserved variables.

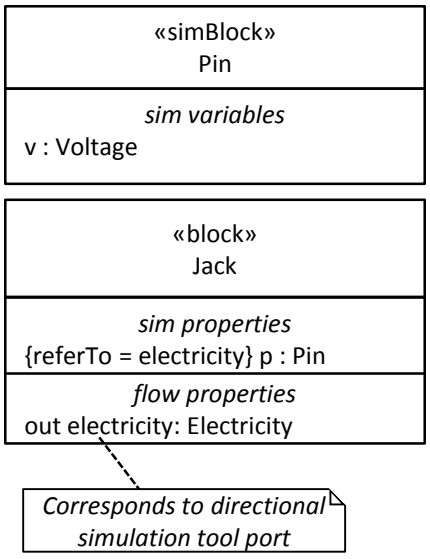



Figure 14 – Example of modeling in SysML a simulation tool output port

In both examples, the block Port can be used to type port properties.

## 6. Modeling the behavior of simulation tools blocks

Since differential/difference equations can be interpreted as mathematical constraints, using constraint blocks to represent them seems natural. Note that using constraint blocks has another advantage: it permits reusability. We can define constraints blocks describing differential equations that can be used to express behavior for several simulation blocks.

In Figure 15 we show how we can use constraint blocks to model the following differential and difference equations:

$$dx/dt = a*x+b*u$$

$$x_{n+1} = a*x_n+b*u_n$$

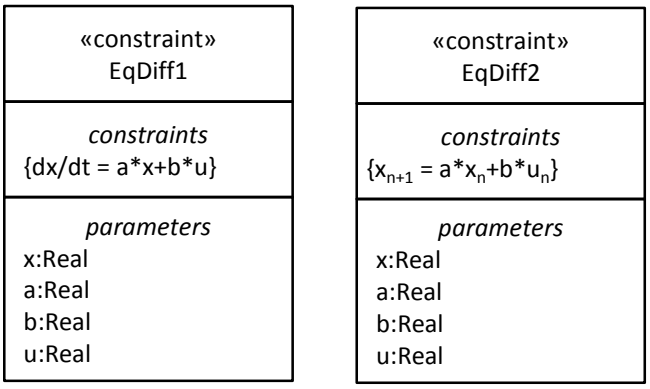



Figure 15 – Example of using constraint blocks to model differential and difference equations

The parameters of these constraint blocks can be bound to SimVariables of SimBlock using parametric diagrams, see example in Section 7.

If reusability is not a priority, we can use UML Constraint directly to represent equations. A UML constraint is shown as a text string in curly braces according to the following syntax:

***constraint*** ::= '{' [ ***name*** ':' ] ***boolean-expression*** '}'

UML specification does not restrict languages which can be used to describe constraints, but the Object Constraint Language is predefined in UML. If we choose to use UML constraints, care must be taken to use the variable names in the constraint.

## 7. Modeling example

In the following we show how we can use the newly introduced SysML constructs to represent the electrical circuit in Figure 16.

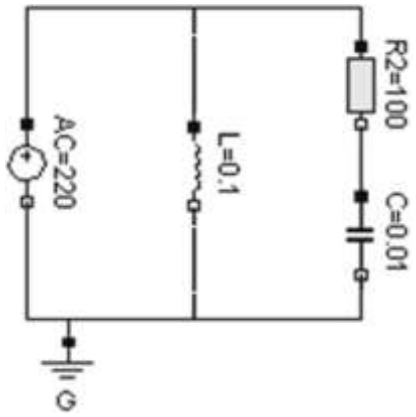



Figure 16 – Electrical circuits

Figure 17 shows a block definition diagram describing the structure of the electrical circuit while In Figure 18 and Figure 19 we show a more detailed description of the components of the electrical circuit reflecting their constants, variables, and constraints for describing behavior. In addition, Figure 20 presents an internal block diagram that shows connections between the components of the electrical circuit, while Figure 21 shows a usage of a constraint block that describes the behavior of the component Resistor.

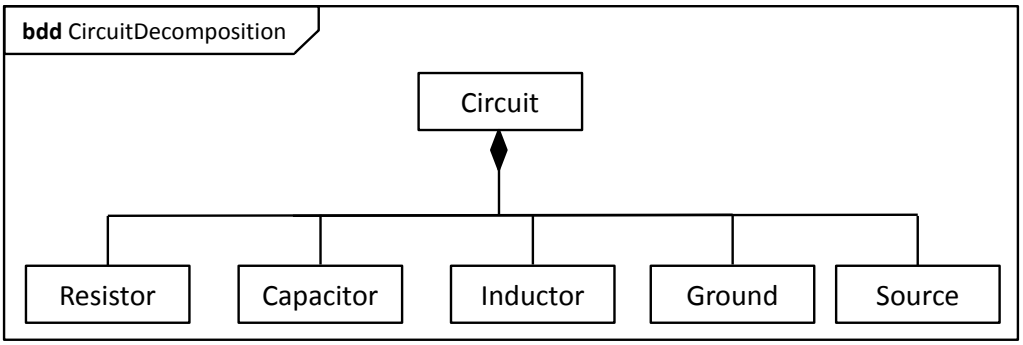



Figure 17 - Electrical circuit structure

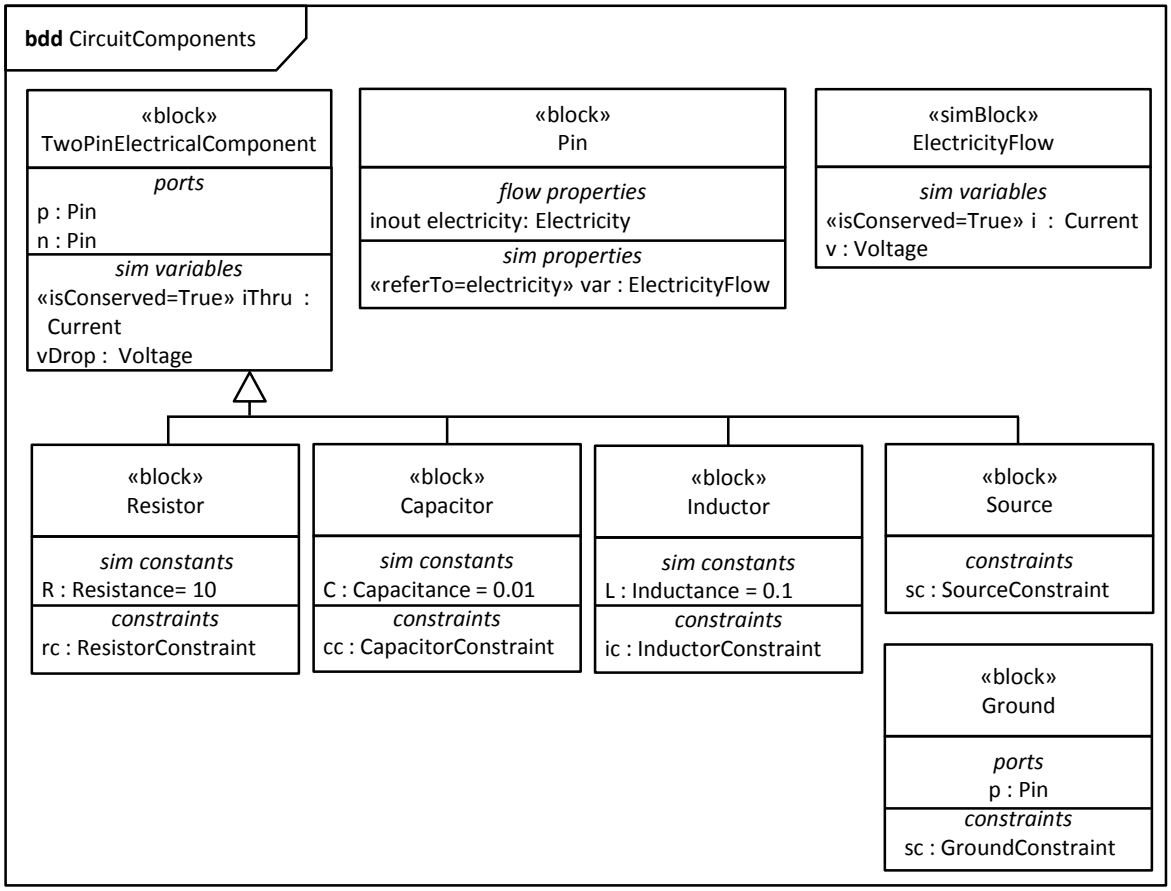



Figure 18 – Description of the electrical circuit components with port modeling

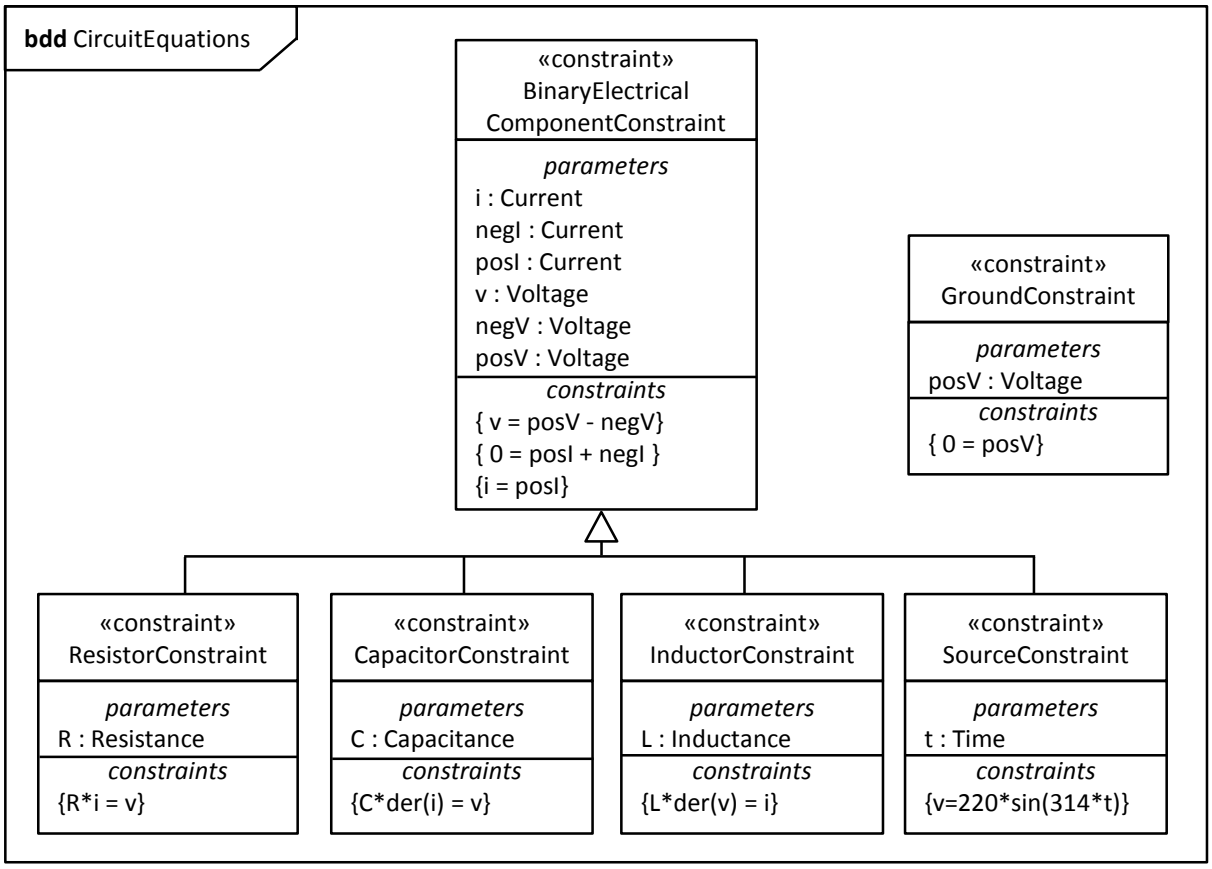



Figure 19 – Block definition diagram to represent the behavior of the electrical circuit components

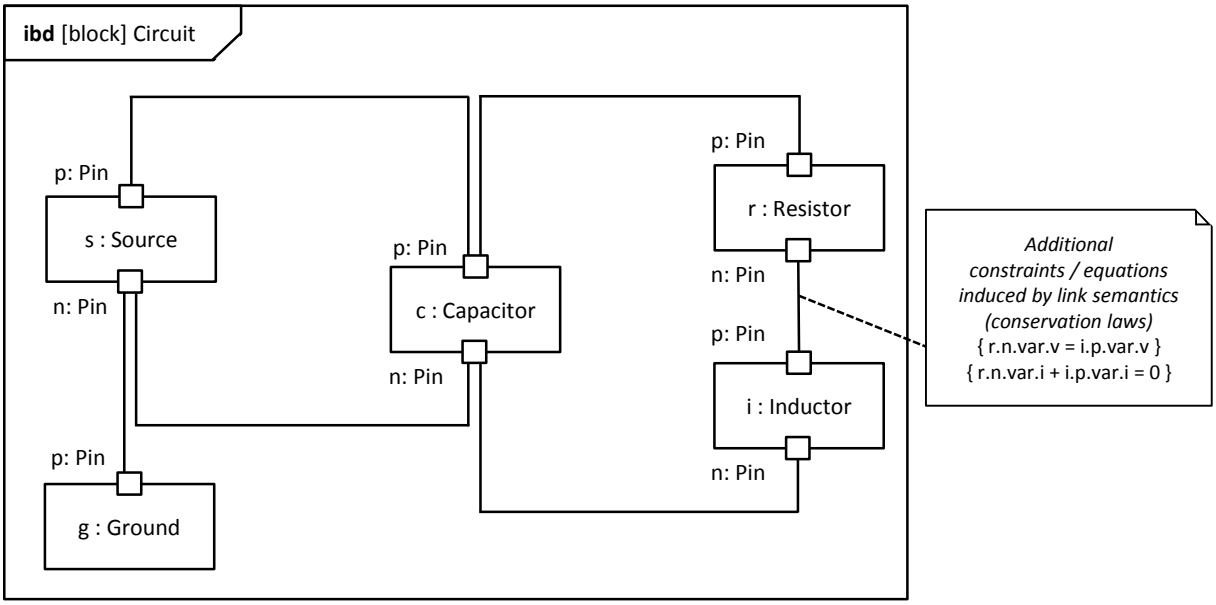



Figure 20 – Internal block diagram to represent the internal structure of the electrical circuit

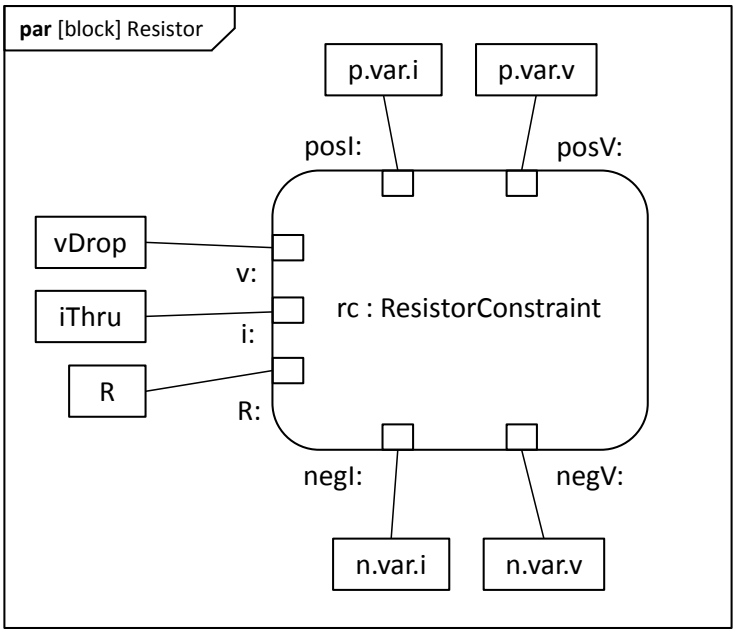



Figure 21 - Parametric diagram for connecting the variables of the Resistor block with the parameters of the constraint block

## 8. Conclusions

In this report we first investigate if SysML has constructs to model dynamical systems. We conclude that SysML has a set of constructs that partially match the constructs used by simulation tools to model dynamical systems. However, we need to extend some of these constructs to harmonize with the constructs used in simulation tools. The extended constructs allow for representation of simulation tool variables and parameters and for setting direction of simulation tool ports. In addition we show through an example how we can use the extended SysML constructs to model an electrical circuit.

**Disclaimer**: Commercial products and services are identified to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products and services identified are necessarily the best available for the purpose.

## References

[1] J. Estefan, "Survey of Candidate Model-Based Systems Engineering (MBSE) Methodologies," International Council on Systems Engineering (INCOSE), INCOSE-TD-2007-003-02, 2008.

[2] M. Hause, "The SysML Modelling Language," in *Fifth European Systems Engineering Conference*, 2006.

[3] Object Management Group, "OMG Systems Modeling Language," www.omg.org/spec/SysML/1.3/, 2012.

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

[5] C. Bock, "SysML and UML 2 Support for Activity Modeling," *Systems Engineering,* vol. 9, no. 2, pp. 160-185, 2006.

[6] R. Snyder, D. Bocktaels and X. Feigenbaum, "Functional validation with a practical SysML / Simulink transformation," in *NEPTUNE Workshop*, 2010.

[7] R. Snyder, D. Bocktaels and X. Feigenbaum, "Validation fonctionnelle a l'aide d'une transformation SysML/Simulink," *Genie Logiciel,* no. Juin, pp. 49-53, 2010.

[8] R. Boldt, "Combining the Power of MathWorks Simulink and Telelogic UML/SysML- based Rhapsody to Redefine the Model Driven Development Experience," Telelogic White paper, 2007.

[9] P. Vasaiely, "Interactive Simulation of SysML Models using Modelica," Hamburg University of Applied Sciences, 2009.

[10] C. Paredis, "An Overview of the SysML-Modelica Transformation Specification," in *20th Anniversary INCOSE International Symposium*, 2010.

[11] Object Management Group, "SysML-Modelica Transformation Specification," http://www.omg.org/spec/SyM/1.0/Beta3/, 2012.

[12] I. Matei and C. Bock, "Modeling Methodologies and Simulation for Dynamical Systems," NIST interagency Report 7875, http://dx.doi.org/10.6028/NIST.IR.7875, 2012.