NISTIR 7875

Modeling Methodologies and Simulation for Dynamical Systems

Ion Matei
Conrad Bock

http://dx.doi.org/10.6028/NIST.IR.7875

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

NISTIR 7875

Modeling Methodologies and Simulation for Dynamical Systems

Ion Matei
Conrad Bock
Systems Integration Division
Engineering Laboratory

http://dx.doi.org/10.6028/NIST.IR.7875

August 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

Modeling Methodologies and Simulation for Dynamical Systems

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, such as the Systems Modeling Language, an extension of the Unified Modeling Language. However, these languages usually do not support detailed enough simulation and performance analysis and must be enhanced with domain specific tools for this purpose. System modeling and simulation tools are often used separately and sequentially, which reduces the efficiency of the design process. As a result, there is an increasing need for integrating different simulation tools under a common framework for integration into system modeling tools. In this report we present results of analyzing modeling methodologies for simulating dynamical systems. We identify two primary methodologies, namely signal-flow and physical-interaction, and introduce the main concepts for building models based on these methodologies. To support the analysis, we give representative implementation examples of these concepts and methodologies in two simulation tools. We identify these concepts and methodologies in order to build an abstract model of simulation tools that facilitates integration of multiple simulation tools into a common framework.

1. Introduction

Simulation of a system is the imitation of the operation of a real-world system over time [1]. Simulation requires a model, which is a representation of a system used to answer questions, without doing experiments on the real system. Experiments may be too expensive or dangerous, the time-scale of the dynamics too long to allow performing experiments in a reasonable time interval, variables of interest may be inaccessible (may not be measurable or observable), and so on. A model can be seen as a simplified system that reflects some properties of the real system [2] [3]. Limitations and conditions under which a simulation is valid must always be considered when evaluating results. When possible, we should compare simulation results against experimental results from the real system. For simulation purposes we are mainly interested in mathematical models, as opposed to physical or other simulation media.

In this report we present results of analyzing modeling methodologies for simulating dynamical systems. We identify two primary methodologies, namely signal-flow and physical-interaction, and give representative implementation examples of these methodologies to support the analysis. In addition, we introduce the main concepts for building models based on these methodologies. Our goal is to emphasize the common methodologies across many tools, to be used as integration points with system modeling languages such as the Systems Modeling Language, an extension of the Unified Modeling Language (SysML/UML) [4].

The rest of the report is organized as follows. In Section 2 we describe two modeling methodologies used for representing dynamical systems. In Section 3 we introduce modeling concepts to be used under the methodologies. Section 4 shows how the concepts are implemented in two simulation tools, and how they are used in the methodologies. We end the paper with some conclusions in Section 5.

2. Modeling methodologies and simulation

The modeling method determines what information can be produced by simulation. The primary modeling methodologies affecting simulation are *signal-flow* and *physical-interaction modeling*.[1]

2.1. Signal-flow modeling

In signal-flow modeling, the relationship between system components is unidirectional, from outputs of one component to inputs of others. Components use inputs to compute outputs, which flow to other components. This is typical in modeling movement of information (signals), as in control engineering and signal processing. Physical conservation laws do not apply in these applications because the same information can flow (be copied) to multiple components, while physical things cannot. This is true even if the information happens to be expressed in physical quantities, such as voltage. Signal-flow modeling is useful in understanding the mathematics involved in simulating dynamical systems in part because mathematical operators such as addition, multiplication and integration appear explicitly. However, when describing physical-system behavior, signal-flow models are more work to create and are less reusable than physical-interaction models (see next section).

2.2. Physical-interaction modeling

In physical-interaction modeling, the relationship between system components is bidirectional, because actions of one component on another involve reactions back onto the acting component. For example, a component attempting to output water to another component will be affected by how much water the other component can accept and how fast. Physical conservation laws apply, because physical things, such as energy, electricity, and water, are moving, rather than information. For example, physical models of hydraulics require fluid movement between interacting components to balance out so that fluid going out of a component is balanced by the same amounts of fluid coming into other components (as compared to information that is copied in signal-flow modeling). Physical models do not specify inputs and outputs for components, because their interaction is bidirectional.

2.3. Dangers of incorrect modeling methodology

Physical-interaction methodology supports both conserved and non-conserved quantities, while signal-flow methodology directly supports only non-conserved quantities, and requires more complicated modeling to support conserved quantities (see Section 4.3.2). Consequently, it is easier to making modeling mistakes when using the signal-flow methodology on physical applications than using the physical-interaction methodology. However, it is easier for simulators to support signal-flow modeling, and engineers may be tempted to choose the signal-flow

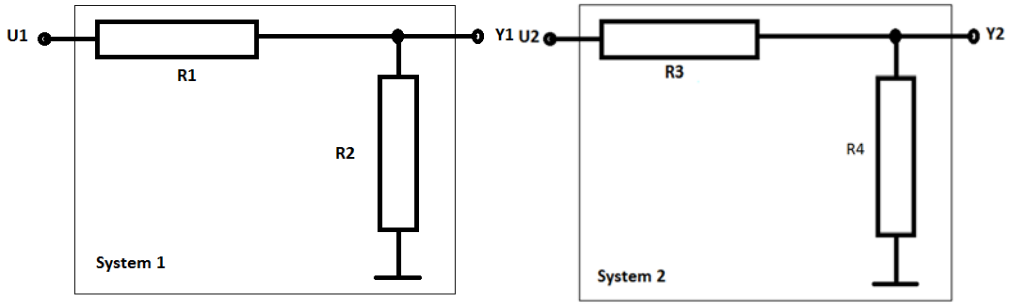
[1] Also known as *causal* or *block diagram* modeling, and *acausal* or *first principles* or *conserved* modeling, respectively.

methodology due to wide availability of tools. Unfortunately, the information provided by simulating the model may prove to be incorrect. Consider the two electrical circuits given in Figure 1. We use a signal-flow model to represent them, where the signal is represented by *voltage*. A simple exercise shows that the input-output relation between voltages Y1 and U1 of System 1 (where U1 is considered input and Y1 is considered output) is given by

$$Y1 = \frac{R2}{R1 + R2} U1$$

Similarly, the input-output relation of the second system is given by

$$Y2 = \frac{R4}{R3 + R4} U2$$



a) b)

Figure 1: a) Depiction of Electrical Circuit 1; b) Depiction of Electrical Circuit 2.

An important characteristic of signal-flow modeling is that when the above two systems are connected in series (as showed in Figure 2), the input-output relation of the resulting system should be given by

$$Y = \frac{R3}{R3 + R4} U2 = \frac{R3}{R3 + R4} Y1 = \frac{R1R3}{(R1 + R2)(R3 + R4)} U$$

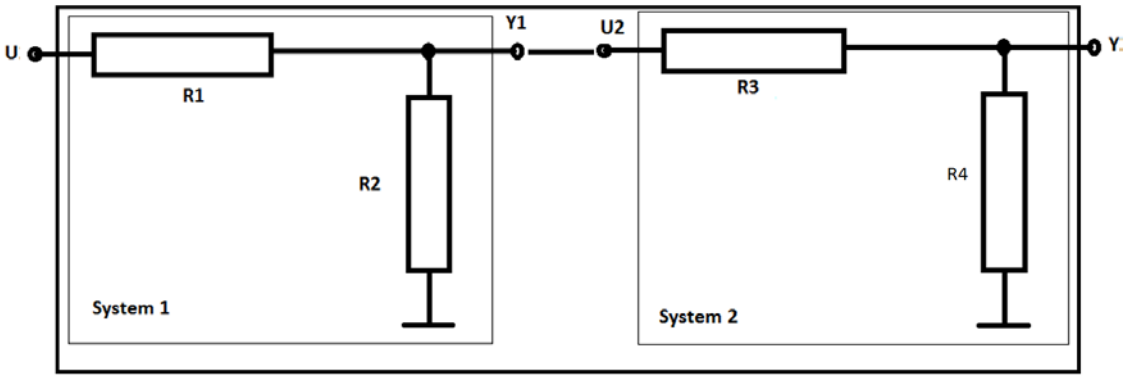


Figure 2: Series connection between Electrical Circuit 1 and Electrical Circuit 2

However, the above input-output relation for the electrical system shown above is incorrect, since the correct expression is given by

$$Y = \frac{R1R3R4}{(R1 + R2)(R3 + R4) + R1R3} U$$

The mistake is that our model ignored the currents (rate of electricity movement) at the connection points.

However, there are cases where signal-flow modeling is equivalent to physical modeling based on conservation laws. For example, a signal-flow model of two electrical circuits with infinite (very large) input impedances and zero (very small) output impedances produces the correct mathematical expression for input-output relation of two systems in series.

3. Building models

To build a model of a system, the first step is to identify the main components of the system and how they interact. Each component is broken down into subcomponents until a desired level of granularity is attained. The granularity level is usually determined by the availability of models belonging to a library of reusable subcomponents. The next step is to specify the interfaces of the subcomponents and mathematically express the interactions between the (sub)components of the model.

Modeling in a simulation tool is based on three basic concepts: components, ports, and links. *Components* are system elements that process and exchange information or physical things. *Ports* are interfaces through which components exchange information or physical things with other components. *Links* are connections between ports of components across which information or physical things are exchanged. A *subsystem* is a component composed of other components, which may be other subsystems. Subsystems show the hierarchical nature of a system or simplify the model, when the number of components is significantly large. A *model* can be viewed as a component composed of other components, which might be subsystems.

The behavior of a model is specified by describing the behavior of each component in it and linking them together. The interaction between components together with their behavior dictates the behavior of the model. The behavior of components that are not subsystems is defined using mathematical models.

3.1. Components

Components consist of subcomponents, variables, and parameters. Subcomponents are other components or ports internal to the component. 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, static, continuous, discrete 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 where properties are constant. 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.

3.1.1. Mathematical equations to express component behavior

As mentioned earlier, we use mathematical models to represent component behavior. We distinguish two types of properties in specifying the behavior of a model: variables and parameters. *Variables* are properties whose values change during the simulation. *Parameters* are properties that are set at the beginning of the simulation but stay constant during the simulation.

The most general mathematical models that simulation tools use are differential algebraic equations (DAEs) [5]. Formally, a continuous-time DAE can be expressed as

$$f_1(\dot{x}(t), x(t), t, \theta) = 0$$
$$\dots$$
$$f_m(\dot{x}(t), x(t), t, \theta) = 0$$

where $x(t) = \big(x_1(t), x_2(t), \dots, x_n(t)\big)$ is a vector of dependent variables, $(f_1, f_2, \dots, f_m)$ is a set of m functions and $\dot{x}(t)$ is the derivative of the state vector. In the above equations, time is denoted by t and θ refers to the (constant) parameters of the model. DAEs are very useful for modeling constrained mechanical systems, electrical circuits, and chemical reaction kinetics.

Example 1: DAE for representing the dynamics of a planar pendulum [6]

Figure 3 presents a mathematical pendulum, where a pendulum of mass m is connected is attached to a rod of length l. The potential energy in terms of the position (x,y) of the pendulum is given by

$$U(x, y) = mgl - mgy$$

and the kinetic energy is

$$T(\dot{x}, \dot{y}) = \frac{1}{2}(\dot{x}^2 + \dot{y}^2)$$

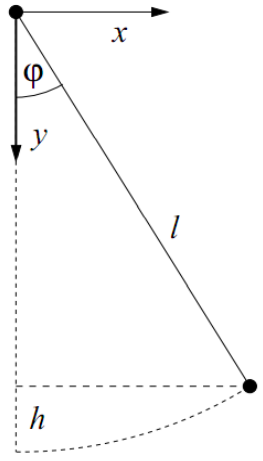


Figure 3: The mathematical pendulum

The mechanical constraint of the pendulum is given by

$$g(x,y) = x^2 + y^2 - l^2 = 0$$

Using Euler's equations, the motion of the pendulum is described by

$$m\ddot{x} + 2\lambda x = 0$$

$$m\ddot{y} - mg + 2\lambda y = 0$$

$$g(x,y) = 0$$

By introducing the additional notations $\varphi_1 = x$, $\varphi_2 = \dot{x}, \varphi_3 = y$ and $\varphi_4 = \dot{y}$ we obtain

$$m\dot{\varphi}_2 + 2\lambda\varphi_1 = 0$$

$$m\dot{\varphi}_4 - mg + 2\lambda\varphi_3 = 0$$

$$\varphi_2 = \dot{\varphi}_1$$

$$\varphi_4 = \dot{\varphi}_3$$

$${\varphi_1}^2 + {\varphi_3}^2 - l^2 = 0$$

which is the DAE for the mathematical pendulum.

We can observe from the previous example that the motion equations were obtained using energy conservation laws, which requires physical-interaction modeling.

A discrete time equivalent of a DAE can be expressed as

$$f(x(k+1), x(k), k, \theta) = 0$$

where $x(k) = \big(x_1(k), x_2(k), \dots, x_n(k)\big)$ and k is the discrete time.

A particular type of DAEs are the ordinary differential equations (ODEs), which can be generally described by

$$\dot{x}(t) = f(x(t), t, \theta)$$

$$0 = h(x(t), t, \theta)$$

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. ODEs are the most common mathematical models used by simulation tools to describe behavior of components. Note that ODEs are DAEs solved for all variables' derivatives. Since ODEs are a kind of DAE they can be simulated using physical-interaction modeling.

ODEs are used for signal-flow modeling, where the relationships between components are in one direction, from output of one component to inputs of another. This is typical in modeling

movement of information, rather than physical things (see Section 2.1). In particular, the (continuous-time) mathematical models used in the signal-flow methodology are

$$\dot{x}(t) = f(x(t), u(t), t, \theta)$$

$$y(t) = h(x(t), u(t), t, \theta)$$

where $u(t)$ represent the inputs of the systems and $y(t)$ are the outputs of the systems. The vector of variables $x(t)$ is usually referred to as the state vector.

A discrete version of the above mathematical model is

$$x(k+1) = f(x(k), u(k), k, \theta)$$

$$y(k) = h(x(k), u(k), k, \theta)$$

where, as before, k denotes the discrete time. Combinations of components with continuous and discrete mathematical models generate *hybrid models*, which are also supported by simulation tools implementing signal-flow modeling.

3.2. Ports

Components interact with other components through interfaces, which are called *ports*. They specify which variables are shared or exchanged with other components. For example, in the case of signal-flow modeling, ports have variables that are either all inputs or all outputs, and are commonly referred to as input or output ports. In the case of physical-interaction modeling, ports still specify which variables interact with other components, but their variables are neither inputs nor outputs.

3.3. Links

As mentioned at the beginning of Section 3, the first step in building a model is to identify the main components of the system and how they interact. Interaction between components combines with the internal behavior of the components to produce the behavior of the entire model. Connecting components actually means connecting their ports using links. When components are connected through links, mathematical formulas are generated. These formulas become part of the behavior of the model and their expressions depend on the type of the modeling methodology used.

3.3.1. Signal-flow link semantics

This type of link is found in signal-flow modeling, where the variables belonging to the ports are designated as inputs or outputs. Interconnecting components means connecting the output ports of some components to the input ports 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 port is connected to an input port, setting a variable of the output port assigns the same value to the same variable in the connected input port. Therefore, the variables that are the same on connected output and input ports must have compatible types and the same dimension if they are vectors. Figure 4 shows equivalent graphical depictions of an output of a component connected to the inputs of two other components via two links.

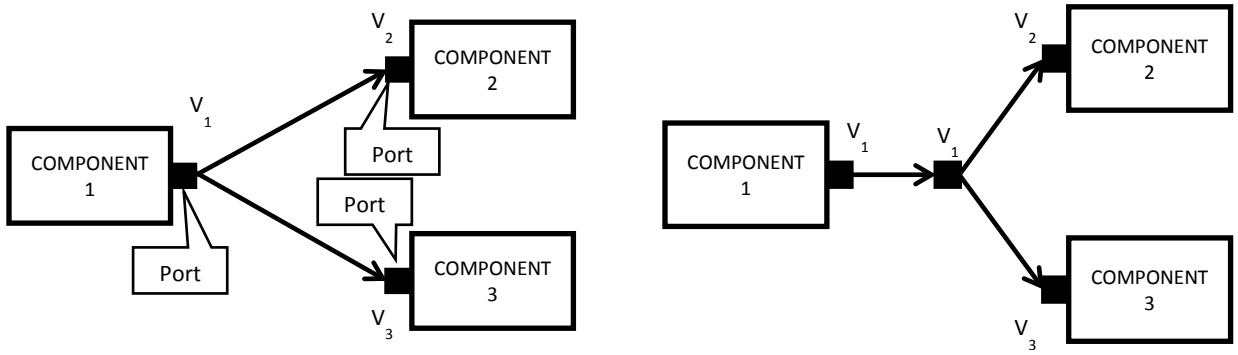


Figure 4: Equivalent graphical depictions of two links in signal-flow modeling

The output port V_1 is connected to the inputs ports V_2 and V_3, where the arrows specify the direction. The signal-flow semantics induces the following assignments

$$V_2 := V_1$$

$$V_3 := V_1$$

where := is an *assignment* operator. Assignment means when the value of V_1 is changed, the values of V_2 and V_3 are changed to the same value. The value of V_1 is not affected by changes in V_2 and V_3, but since V_2 and V_3 are on input ports, their values cannot be changed by their components, only by other components linked to them. This reflects the directional nature of signal-flow modeling.

3.3.2. Physical-interaction link semantics

This type of link is found in physical-interaction modeling, where variables belonging to ports interact bidirectionally. We distinguish two types of variables present on ports in physical-interaction models: *conserved* and *non-conserved*.[2] Conserved variables give the rate of movement of physical things that are not created or destroyed by the system, for example current and power are rates of movement of charge and energy, respectively. Non-conserved variables describe physical things in other ways than their rate of movement. For example, voltage is a non-conserved variable for the exchange of electricity, because it measures electric potential, rather than movement of charge. Physical-interaction links between ports have the following semantics: the values of conserved variables sum up to zero, and values of non-conserved variables must be equal.

Figure 5 depicts two equivalent graphical depictions showing two links from one component to two others. The ports of these components have conserved (I_1, I_2, I_3) and non-conserved (V_1, V_2, V_3) properties.

[2] Simulation tools call these “through” and “across” variables, respectively.

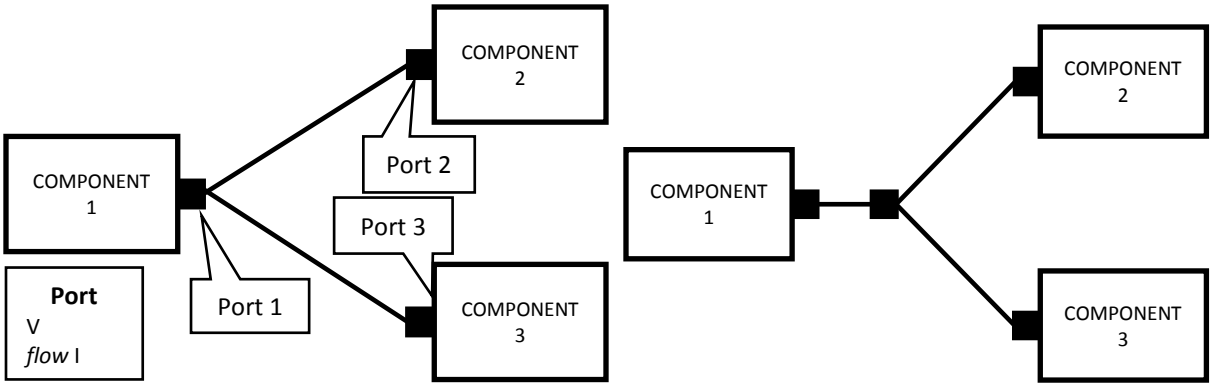


Figure 5: Equivalent graphical depictions of two links in physical-interaction modeling

According to physical-interaction link semantics, we get the following equations between the variables of the ports:

$$V_2 = V_1$$

$$V_3 = V_1$$

$$I_1 = I_2 + I_3$$

Note that in this case we obtain a set of equations and not assignments. This means the values of V_1, V_2, and V_3 can only change in ways that maintain the specified equalities. Equations do not restrict which variables affect which others. They can all affect each other within the bounds of the equations. This reflects the bidirectional nature of physical-interaction modeling.

4. Examples of modeling concepts in simulation tools

Most simulation tools provide graphical user interfaces for building models, including graphical modeling *constructs* corresponding to the modeling concepts described in Section 3. Modeling details are described using textual languages specific to the tools. Modeling constructs can be edited using the underlying textual languages of the simulation tools to specify properties and behavior of the constructs. Simulation tools often provide prebuilt modeling constructs, organized in libraries. These constructs have preset properties behavior, but users can modify parameter values before starting simulations.

In this section we present examples of modeling concepts in specific simulation tools using signal-flow and physical-interaction methodologies. As a representative of signal-flow modeling tools, we use Simulink®, based on the textual modeling and simulation language Matlab® [7]. As a representative of physical-interaction modeling, we use OpenModelica, an open source tool based on Modelica®, a non-proprietary textual modeling and simulation language [8] [9] [10]. Note that, although two simulation tools may implement the same methodology and concepts, their modeling constructs may be different.

4.1. Signal-flow modeling concepts in Simulink

Graphically, Simulink shows models as block diagrams, where blocks are components usually shown as rectangles (although other shapes are sometimes used). Links are shown as lines between blocks. Ports are shown on blocks with additional icons, or more compactly as direction markers on links.

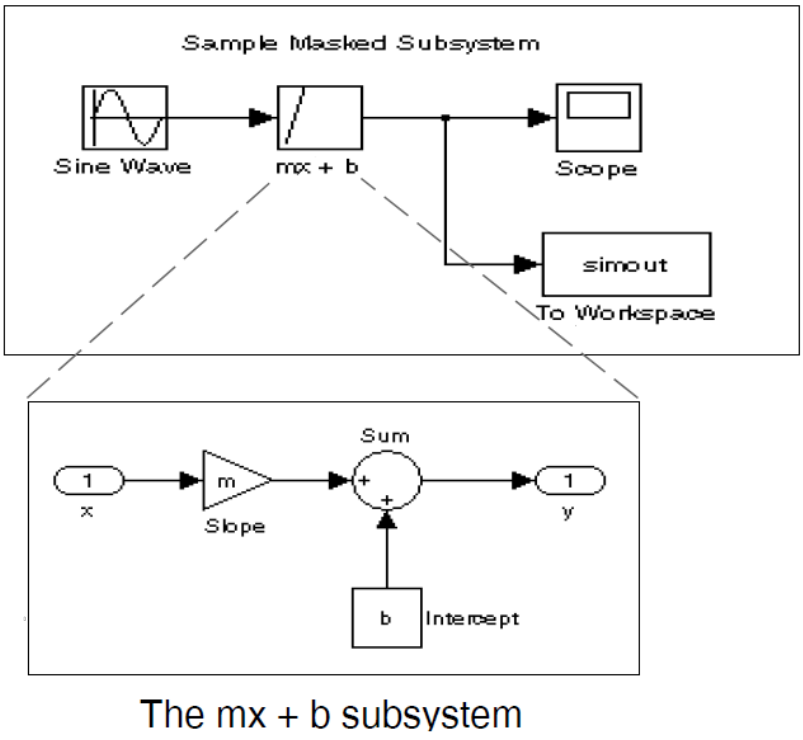


Figure 6: Block diagrams

Figure 6 shows an example of a block diagram with blocks that generate signals (Sine Wave in this example), process signals (mx+b), and show time evolution of signals (scope), as well as links to move signals downstream. Additionally, Figure 6 presents the internal structure of the subsystem "mx+b", showing input/output ports explicitly as round-cornered rectangles.

Models are saved in structured text files containing keywords on groups of parameter-value pairs. The model file describes model components in hierarchical order and as follows [7]:

```
Model {
        <Model Parameter Name> <Model Parameter Value>
        ...
        BlockDefaults {
                <Block Parameter Name> <Block Parameter Value>
                ...
        }
        AnnotationDefaults {
                <Annotation Parameter Name> <Annotation Parameter Value>
                ...
        }
```

```
        System {
                <System Parameter Name> <System Parameter Value>
                ...
                Block {
                        <Block Parameter Name> <Block Parameter Value>
                        ...
                }
                Line {
                        <Line Parameter Name> <Line Parameter Value>
                        ...
                        Branch {
                                <Branch Parameter Name> <Branch Parameter Value>
                                ...
                        }
                }
                Annotation {
                        <Annotation Parameter Name> <Annotation Parameter Value>
                        ...
                }
        }
}
```

As shown above, the file has four sections. In the Model section, aspects of the model are defined, such as the model name, and simulation parameters. The BlockDefaults section defines values for block parameters to be used when the modeler does not specify any, while AnnotationDefaults gives purely graphical information in the model. The top-level system and each subsystem in the model are described in separate System sections. Each System section defines system-level parameters and includes Block, Line, and Annotation sections for each block, line, and annotation in the system. Each Line that contains a branch point includes a Branch section that defines the branch line.

In the following we present excerpts from a model file for diagrams shown in Figure 6.

```
Model {
        Name "slope intercepts"
        ...
        BlockDefaults {...}
        AnnotationDefaults {...}
        % Structure of the model
        System { Name “slopeintercept”
                ...
                Block {
                        BlockType Scope
                        Name “Scope”
                        NumInputPorts “1”
                        ...
                }
                Block {
                        BlockType Sin
                        Name “SineWave”
                        ...
                }
                Block {
                        BlockType ToWorkspace
                        Name “To Workspace”
```

```
        ...
}
Block {
        BlockType SubSystem
        Name “mx+b”
        ...
}
% Structure of the subsystem
System {
        Name “mx+b”
        ...
        Block {
                BlockType Inport
                Name “x”
                Port “1”
                SampleTime -1
                ...

        }
        Block {
                BlockType Constant
                Name “Intercept”
                Value “b”
                ...
        }
        Block {
                BlockType Gain
                Name “Slope”
                Gain “m”
                ...
        }
        Block {
                BlockType Sum
                Name “Sum”
                Inputs “|++”
                ...
        }
        Block {
                BlockType Outport
                Name “y”
                Port “1”
                Inputs “|++”
                ...
        }
        % Specification of links
        Line {
                SrcBlock “Intercept”
                SrcPort 1
                DstBlock “Sum”
                DstPort 2
        }

        Line {
                SrcBlock “Sum”
                SrcPort 1
```

```
                    DstBlock “y”
                    DstPort 1
                }
                Line {
                    SrcBlock “x”
                    SrcPort 1
                    DstBlock “Slope”
                    DstPort 1
                }
            }
            Line {
                SrcBlock “Sine Wave”
                SrcPort 1
                DstBlock “mx+b”
                DstPort 1
            }
            Line {
                SrcBlock “mx+b”
                SrcPort 1
                ...
                Branch{
                DstBlock “Scope”
                DstPort 1
                }
                Branch{
                DstBlock “To Workspace”
                DstPort 1
                }
            }
        }
        ...
}
```

The above example shows a hierarchically structured model file. Blocks are identified by name and type. The section starting with the keyword System are blocks depicted in the block diagram window of the graphical interface. If a block is a subsystem then an additional System section is needed, placed in the interior of the System section to which the subsystem belongs, forming a system hierarchy. Links between ports are specified with the name and number of the source port (output) and the name and number of the destination port (input). Ports are automatically numbered, starting with 1 in each block. If a new port is added, the next available number is assigned to it. When a port is deleted, the other ports are automatically renumbered to ensure that the ports are in sequence and that no numbers are omitted.

Next we address specification of block behavior, based on the model structure above. Block behavior is described using S-functions, which can be written in Matlab, C, or C++ languages. S-functions are very general, in that they are able to accommodate continuous, discrete and hybrid systems. An S-function has the following form

```
function [sys,x0,str,ts]=f(t,x,u,flag,p1,p2,...)
```

where f is the S-function's name, t is the current time, x is the state vector of the corresponding S-function block, u is the block's inputs, flag indicates a task to be performed, and p1, p2, ... are

the block's parameters. During simulation, f is repeatedly invoked, using a flag to indicate the task to be performed for a particular invocation. The general template of an S-function implemented as follows [4].

```
function [sys,x0,str,ts] = f(t,x,u,flag,p1,p2,...)
switch flag,
        case 0,[sys,x0,str,ts] = mdlInitializeSizes(lb,ub,xi);
        case 1,[sys,x0,str,ts] = mdlDerivatives(t,x,u,lb,ub);
        case 2,[sys,x0,str,ts] = mdlUpdate(t,x,u);
        case 3,[sys,x0,str,ts] = mdlOutputs(t,x,u);
        case 9,[sys,x0,str,ts] = mdlTerminate(t,x,u);
end
```

The tasks corresponding to cases 0 and 9 execute only once, at the beginning and at the end of the simulation, respectively. In the initialization step (task mdlInitializeSizes), several subtasks are performed: state initialization and state constraints determination (upper and lower bounds); determination of the number of continuous and discrete states (the continuous, discrete or hybrid nature of the block), interface description through determination of the number of inputs and outputs (they can be dynamically sized), and sample time initialization (in the case of discrete blocks). The termination step (task mdlTerminate) performs any actions required at termination of the simulation, such as freeing memory.

The behavior of a block, that is the description of an ODE corresponding to the block, is implemented through the tasks mdlDerivatives(t,x,u,lb,ub), mdlUpdate(t,x,u) and mdlOutputs(t,x,u), which are called during the simulation process. The task mdlDerivatives computes the derivatives of the continuous components of the state vector, that is, it implements

$$\dot{x}(t) := f(x(t), u(t), t, \theta)$$

The task mdlUpdate computes the values of the state vector at the current time of the simulation. In the case of discrete blocks, the state are updated according to

$$x(k+1) = f(x(k), u(k), k, \theta)$$

while in the case of continuous states, they are computed using the values of the state derivatives obtained as a result of the mdlDerivatives task.

The task mdlOutputs computes the outputs of the block at the current time of the simulation, that is, it implements the assignments

$$y(t) := h(x(t), u(t), t, \theta)$$

in the case of continuous states, or

$$y(k) := h(x(k), u(k), k, \theta)$$

in the case of discrete case.

As indicated by the numbering of the tasks, other tasks can be invoked during the simulation process. These tasks are related to numerical algorithms used to solve the differential equations of the blocks. A complete list of the tasks that can be implemented in a S-function can be found in [7].

4.2. Physical-interaction modeling concepts in Modelica

Modelica is a non-proprietary textual language for specification of complex systems, potentially containing subcomponents from multiple engineering domains, such as mechanical, electrical, electronic, hydraulic, thermal, control, or process-oriented. The language is designed to support computer simulation of dynamic systems, where behavior evolves as a function of time [9] [11] [12]. The mathematics of Modelica is based on equations rather than assignments (as in the case of Simulink), which permits physical-interaction modeling. It is an object-oriented language with a general class concept that facilitates reuse of components.

We begin our discussion by showing how the planar pendulum in Example 1 is described.

```
class Pendulum        "Planar Pendulum"
    Real x, y, vx, vy;
    parameter Real g = 9.81, m = 1, l=0.5, lambda =1;
    equation
            m*der(vx)+2*lambda*x = 0;
            m*der(vy)-m*g+2*lambda*y = 0;
            der(x) = vx;
            der(y) = vy;
            x^2+y^2-l^2 = 0;
end Pendulum;
```

Class is the most general concept in Modelica. It has a number of specializations, but the example above defines a general class, Pendulum, that has four variables of type real: x, y, vx, and vy. The parameters of the class are specified using the keyword parameter. The behavior of the class is described in a special section, called equation. The keywords used to define parameters and behavior are the same as or similar to the mathematical terms: Real refers to the real numbers, der(x) refers to the time derivatives of x.

To simplify reading and maintenance, special keywords are introduced for specific uses of the general class concept. The relevant specialized classes for modeling and simulating dynamical systems are model, block and connector. A model is the same as a class, that is, the keywords are interchangeable, and corresponds to a component. They support physical-interaction modeling. A block is a class where all ports are declared as inputs or outputs, as in Simulink, supporting signal-flow modeling.

A connector is a class used to define kinds of ports. Modelica components (classes, models, blocks) define ports of particular kinds by defining variables typed by connectors. These variables/ports can be linked together using the keyword connect. As pointed out earlier, variables in physical-interaction modeling can be specified as conserved and non-conserved. Conserved variables are preceded by the keyword flow, while non-conserved variables have no keyword. We will show in the following an example of how components and interconnections between components are represented.

4.3. Comparison of signal-flow and physical-interaction modeling on an electrical circuit

In this section we apply the signal-flow and physical-interaction methodologies on an electrical circuit and discuss their advantages and disadvantages with respect to the modeling complexity and accuracy.

4.3.1. **Physical-interaction model of an electrical circuit**

Consider the electrical circuit in Figure 7 (shown with OpenModelica graphical notation), which has four types of components: resistors, capacitors, inductors, and AC voltage source [10].

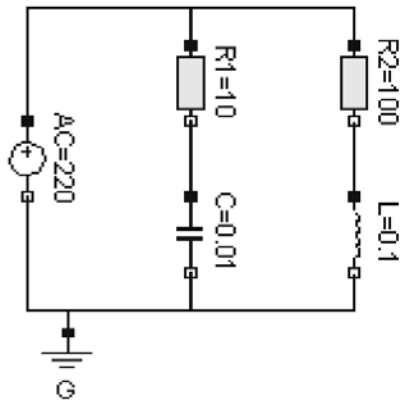


Figure 7 - Example of electrical circuit

In order to model the electrical circuit, each of these components must be described first. Note that each component has two interaction points, that is, two ports. The electricity exchanged between components is described by two variables: voltage and current, where current is a conserved variable because it is the rate of movement of electrons, which must be conserved across links, while voltage is a non-conserved variable, because it has the same value across links. The kind of port is described using a connector class:

```
connector Pin
  Voltage  v;
  flow Current i;
end Pin;
```

where Voltage and Current are predefined data types.

We are now ready to define the components of the circuit. Variables typed by Pin are ports of the component, because Pin is a connector. Each component has two variables: voltage and current of the components, the first giving the voltage drop across the component, the second the current through the component. These two variables can be expressed in terms of the variables of the ports. The resistor, capacitor, inductor and AC voltage source are given by the following models.

```
model Resistor "Ideal electrical resistor"
        Pin p,n;
        Voltage v;
        Current i;
        parameter Real R(unit="Ohm") "Resistance";
        equation
                v = p.v - n.v;
                0 = p.i + n.i;
                i = p.i;
                R*i = v;
end Resistor

model Capacitor "Ideal electrical resistor"
        pin p,n;
        Voltage v;
        Current i;
        parameter Real C(unit="F") "Capacitance";
        equation
                v = p.v - n.v;
                0 = p.i + n.i;
                i = p.i;
                C*der(v)=i;
end Capacitor

model Inductor "Ideal electrical inductor"
        pin p,n;
        Voltage v;
        Current i;
        parameter Real L(unit="H") "Inductance";
        equation
                v = p.v - n.v;
                0 = p.i + n.i;
                i = p.i;
                L*der(i)=v;
end Inductor
```

```
model VSourceAC "Sin Wave Voltage source"
        pin p,n;
        Voltage v;
        Current i;
        parameter Voltage V=220 "Amplitute";
        parameter Real f(unit=”Hz”) = 50 “Frequency”
        equation
                v = p.v - n.v;
                0 = p.i + n.i;
                i = p.i;
                v = VA*sin(2*f*3.14*time);
end VSourceAC
```

In a similar way, we can model the electrical ground.

```
model Ground "Ground"
        pin p;
        equation
                p.v = 0;
end Ground;
```

Having defined the components, the electrical circuit in Figure 7 is modeled as

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

The connect keyword adds a set of equations following physical-interaction link semantics. For example

```
connect (AC.p, R1.p);

connect (R1.p, R2.p);
```

generates the following equations:

```
AC.p.i+R1.p.i+R2.p.i = 0;

AC.p.v = R1.p.v;

R1.p.v = R2.p.v;
```

4.3.2. **Signal-flow model of an electrical circuit**

Modelica also supports signal-flow modeling with block type classes. For these type of classes, all ports must have assigned direction. In the following, we model the electrical circuit presented in Figure 7, using a signal-flow approach. To this end, we must choose for each component a set of input and outputs. The differential equation governing the capacitor's behavior is given by

$$\frac{dv(t)}{dt} = \frac{1}{C} i(t)$$

In signal-flow modeling, outputs are produced from current and past inputs, and cannot depend on future inputs. Since in the above equation the derivative is taken with respect of the voltage, we are going to choose as input the current $i(t)$ and as output the voltage $v(t)$. In the case of the inductor, the behavior is given by

$$\frac{di(t)}{dt} = \frac{1}{L} v(t)$$

In the case of the resistors, the input and output will be chosen as needed. That is, the voltage can be chosen as output and the current can be chosen as input, or the other way around. Figure 8 shows a signal-flow model of the electrical circuit presented in Figure 7 [13].

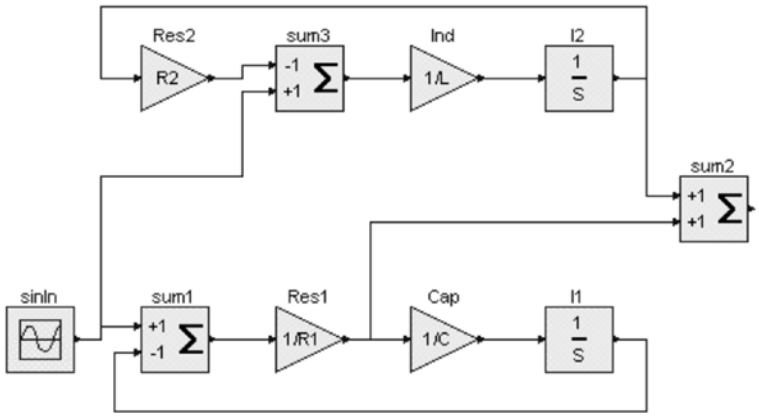


Figure 8 – A signal-flow modeling methodology approach for the electrical circuit in Figure 7.

Although mathematically equivalent with the physical-interaction representation, applying signal-flow to physical applications increases the complexity of the model, as follows:

- Blocks do not show the physical things they are modeling. In our example, the physical things are resistors, capacitors, inductors and sources. In particular, representing the resistor is different depending on how it is used, rather than reusing the same resistor component, as in physical-interaction modeling.
- Inputs and outputs must be chosen for each component separately, even if they represent the same physical thing, such as a resistor (again impairing reusability).
- The ODEs used in a block must be manually derived from conservation equations and DAEs describing component behavior. In particular, determining consistent initial conditions may be more challenging.

In what follows, we present the signal-flow implementation using Modelica. We start by modeling the signals block exchange among them.

```
connector Signal
  Real      val;
end Signal;
```

The signal-flow representation of the AC source is given by

```
block VSourceAC "Sin Wave Voltage source"
        output Signal out_sig;
        parameter Voltage V=220 "Amplitute";
        parameter Real f(unit=”Hz”) = 50 “Frequency”
        equation
                out_sig = VA*sin(2*f*3.14*time);
end VSourceAC
```

Note that we use the specialized class block reflecting the signal-flow approach and that we specifically mention that out_sig is the output by using the keyword output. As we can see in Figure 8, we need three more components: Summation, Gain and Integrator, presented in what follows.

```
block Summation
        input Signal in_sig1, in_sig2;
        output Signal out_sig;
        parameter Real scale1 = 1.0;
        parameter Real scale2 = 1.0;
        equation
                out_sig = scale1*in_sig1.val+ scale1*in_sig1.val;
end Summation
```

```
block Gain
        input Signal in_sig;
        output Signal out_sig;
        parameter Real K = 1.0 “Gain factor”;
        equation
                out_sig.val = K*in_sig.val;
end Gain
block Integrator
        parameter Real init_value = 0.0;
        input Signal in_sig;
        output Signal out_sig(val(start=init_val));
        equation
                der(out_sig.val) = in_sig.val;
end Integrator
```

Having defined the necessary component, the signal-flow representation of the electrical circuit in Figure 7, is given by

```
model circuit
    VsourceAC sinIn;
        Summation sum1(scale2=-1);
        Summation sum2;
        Summation sum3(scale1=-1);
        Gain  Res2(K=10);
        Gain  Res1(K=0.1);
        Gain Cap (K=100);
        Gain Ind (K=10);
        Integrator I1;
        Integrator I2;
equation
        connect (sinIn.out_sig, sum1.in_sig1);
        connect (sinIn.out_sig, sum3.in_sig2);
        connect (Res2.out_sig, sum3.in_sig1);
        connect (sum3.out_sig, Ind.in_sig);
        connect (Ind.out_sig, I2.in_sig);
        connect (I2.out_sig, Res2.in_sig);
        connect (I2.out_sig, sum2.in_sig1);
        connect (sum1.out_sig, Res1.in_sig);
        connect (Res1.out_sig, Cap.in_sig);
        connect (Res1.out_sig, sum2.in_sig2);
        connect (Cap.out_sig, I1.in_sig);
        connect (I1.in_sig, sum1.in_sig2);
end circuit;
```

Details about the graphical representation of a component are prefixed by the keyword annotation. Graphical annotation information includes position, size, and icon of the component as it appears on the diagram, common properties such as the local coordinate system, or information associated with link lines, such as route and color. The example below shows the class Resistor with graphical annotations.

```
model Resistor
  Pin p annotation (extent=[-110, -10; -90, 10]);
  Pin n annotation (extent=[ 110, -10;  90, 10]);
  parameter R "Resistance in [Ohm]";
equation
  R*p.i = p.v - n.v;
  n.i = p.i;
public
  annotation (Icon(
    Rectangle(extent=[-70, -30; 70, 30], style(fillPattern=1)),
    Text(extent=[-100, 55; 100, 110], string="%name=%R"),
    Line(points=[-90, 0; -70, 0]),
    Line(points=[70, 0; 90, 0])
    ));
end Resistor;
```

5. Conclusion

In this report, we present the results of analyzing modeling methodologies for dynamical systems, identifying two primary ones: signal-flow and physical-interaction. We show that although the two methodologies use the same concepts, they have significant semantic differences that make them appropriate for different modeling applications. To support the analysis, we give representative examples of the two methodologies in two simulation tools, and show how modeling concepts are implemented. These results are presented as part of creating an abstract representation of simulation tools that can be used within a framework for system modeling, such as SysML/UML.

References

[1] J. Banks, J. S. Carson, L. B. Nelson and D. M. Nicol, Discrete-Event System Simulation (third edition), New York, NY: Prentice Hall, 2001.

[2] P. P. J. van den Bosch and A. C. van den Klauw, Modeling Identification and Simulation of Dynamical System (first edition), CRC-Press, 1994.

[3] P. Fritzon, Introduction to Modeling and Simulation of Technical and Physical Systems, New York, NY : IEEE Press Wiley, 2011.

[4] Object Management Group, "OMG Systems Modeling Language," www.omg.org/specs/SysML/1.2/, 2011.

[5] P. Kunkel and V. Mehrmann, Differential-algebraic equations: analysis and numerical solution, Zurich, Switzerland: European Mathematical Society, 2006.

[6] S. Schulz, "Four Lectures on Differential-Algebraic Equations," Humboldt University of Berlin, Berlin, Germany, 2003.

[7] The MathWorks Inc, SIMULINK-Dynamic Systems Simulation for MATLAB, The MathWorks Inc, 2001.

[8] Open Source Modelica Consortium, "Open Modelica Users Guide," https://www.openmodelica.org/images/docs/userdocs/OpenModelicaUsersGuide.pdf, 2012.

[9] Modelica Association, "Modelica: A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification," https://modelica.org/documents/ModelicaSpec32.pdf, 2010.

[10] Modelica Association, "Modelica: A Unified Object-Oriented Language for Physical Systems Modeling - Tutorial," https://modelica.org/documents/ModelicaTutorial14.pdf, 2000.

[11] M. M. Tiller, Modeling with Modelica, Norwell, MA: Kluwer Academic Publisher, 2001.

[12] J. Bugard and D. Broman, "Modelica Tutorial – Modeling and Simulation with OpenModelica and MathModelica," Linkoping University, Linkoping, Sweeden, 2009.

[13] Modelica Association, "Modelica: A Unified Object-Oriented Language for Physical Systems Modeling - Tutorial," 2000.