

IMECE2007-42754

MODELING CONTINUOUS SYSTEM DYNAMICS IN SYSML

Thomas A. Johnson¹
tjohnson6@gatech.edu

Jonathan M. Jobe¹
jonathan.jobe@gatech.edu

Christiaan J. J. Paredis¹
chris.paredis@me.gatech.edu

Roger Burkhart²
BurkhartRogerM@johndeere.com

¹Systems Realization Laboratory
The G. W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

² Deere & Company World Headquarters
Moline, Illinois 61265

ABSTRACT

In this paper, we present a formal approach to modeling continuous system dynamics in SysML using differential algebraic equations (DAE's). To support model-based design, the Object Management Group has recently developed the Systems Modeling Language (OMG SysML™). The language is well-suited for modeling many different aspects of large-scale, multidisciplinary engineering projects. It allows systems designers to capture information concerning system requirements, tests, structures, functions, and behaviors. However, SysML lacks explicit support for modeling continuous system dynamics using DAE's. Such models are important for representing system behavior resulting from energy or signal exchange between system components. We introduce support for modeling system dynamics in the form of a language mapping between SysML and Modelica, an equation-based, object-oriented behavioral simulation language. The bidirectional mapping provides support for creating system dynamics models in SysML that can exist alongside other SysML information models, but that can also be transformed into executable simulations by a Modelica solver. To illustrate the approach, we provide an example SysML model of a hydraulic pump.

KEYWORDS

Systems engineering, information modeling, continuous system dynamics, SysML, Modelica

1 INTRODUCTION

Contemporary systems engineering projects encompass many different domains of knowledge, exist at increasingly

large scales, and consist of multiple subsystems and components. Such systems are subject to the requirements and objectives of many different project stakeholders. Studies generally show that problems associated with the development of satisfactory systems have more to do with the organization and management of complexity than with the direct technological concerns that affect individual subsystems and specific physical science areas [1]. If system designers do not fully understand the complexity and emergent behavior of the system under development, they might overlook important design details and relationships. Such mistakes can compromise stakeholder objectives and lead to costly design iterations or system failures.

To eliminate these problems, the systems engineering *Vee model* [2, 3], seen below in Figure 1, encourages exploratory design and analysis during the early stages of system development. Such a practice ensures that system designers have an adequate grasp on the behavior and complexity of a system during the later stages of development. The left side of the Vee represents the decomposition and definition of the system based on an increasing understanding of stakeholder requirements and objectives. The base of the Vee represents the complete disciplinary specification of the system's components, while the right side of the Vee represents the integration of components and the qualification of the system.

Model-based design supports exploratory design and analysis by allowing designers to effectively represent and investigate their knowledge about the system during the decomposition and definition process. Models are used to represent formally the structure, function, and behavior of a

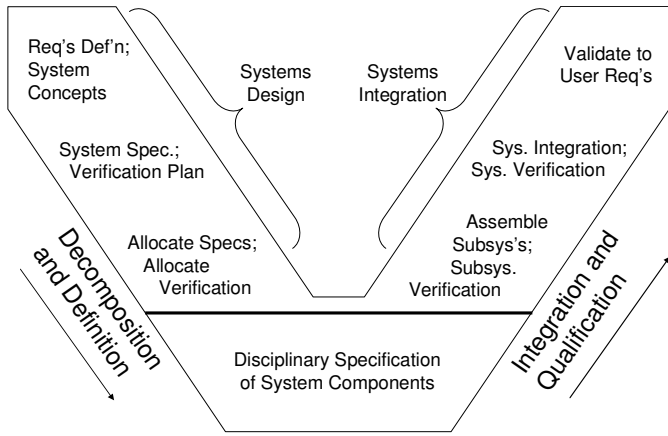


Figure 1. The systems engineering Vee Model [2, 3].

system [4]. Additionally, experiments can be performed on models to eliminate poor design alternatives and to ensure that a preferred alternative meets stakeholder objectives. Models also facilitate designer collaboration by providing a common formalism for communicating information about the system.

To support model-based design, the Object Management Group has developed the Systems Modeling Language (OMG SysML™). SysML is general-purpose systems modeling language that allows system designers to create and manage models of physical systems using well-defined, visual constructs. The knowledge captured by a SysML model is intended to support the specification, analysis, design, and verification and validation of a complex system [5]. Although information models concern system structure, behavioral models are arguably the most important. If the system does not behave in a way that satisfies stakeholder objectives, then it is useless regardless of its structure [6].

The behavior of physical systems is commonly modeled using sets of differential algebraic equations (DAE's) to represent the exchange of energy, signals, or other continuous interactions between system components. However, explicit support for creating such DAE-based representations in SysML does not currently exist. Instead, the basic constructs for creating DAE-based models must be defined in terms of the more primitive constructs available within SysML.

In this paper, we introduce a formal approach for modeling continuous system dynamics in SysML based on a bidirectional language mapping between Modelica and SysML. Modelica is an open-source, object-oriented modeling language that is particularly well-suited for modeling and simulating continuous system dynamics based on energy transfer between system components [7]. By mapping Modelica language constructs to appropriate SysML counterparts, we enable system designers to manage models of continuous system dynamics alongside other information models. Hence, designers can cope more easily with system complexity through the realization of relationships between models of a system's dynamic behavior and models of its cost, physical structure, or engineering requirements.

This paper presents our approach to establishing support for modeling continuous system dynamics in SysML. To provide

an appropriate background for the approach, we first highlight SysML and other related work. We then elaborate on the objectives for the information modeling approach. Based on these objectives, we develop the most important modeling constructs of the approach and illustrate them using an example SysML model of a constant-displacement, hydraulic pump.

2 RELATED WORK

To establish an appropriate background for this paper, this section highlights SysML and two methods commonly used for describing system dynamics using DAE's, namely, causal signal blocks and acausal, equation-based modeling. We also review two extensions of the Unified Modeling Language (UML), ModelicaML and UML^H, used to model DAE-based system dynamics.

2.1 SysML

SysML has been developed by the OMG to provide simple but powerful constructs for modeling a wide range of systems engineering problems. Models created in SysML capture design knowledge and are not typically executable; however, the knowledge can be transformed into analysis models that are used in simulation languages.

The specification of SysML reuses a subset of UML 2.1 and extends it where necessary [5]. Adopted in November 1997, UML is a visual language for specifying, constructing, and documenting the artifacts of software, business models, and other applicable systems. It is a general-purpose modeling language that can be used with all major object and component methods. The language is commonly used during the development of large-scale, complex software for various domains and implementation platforms [8].

The SysML profile was developed to extend UML for increased support of systems engineering projects. It extends UML in the following manners:

- SysML blocks extend UML classes
- SysML enables requirements modeling
- SysML supports parametric modeling
- SysML allocations extend UML dependencies
- SysML reuses and modifies UML activities
- SysML flow ports extend UML standard ports

Figure 2 depicts the SysML diagram taxonomy as a graphical representation of SysML's extension of UML. A block with a regular or bold border represents a UML diagram type that has been reused or modified in SysML, respectively. Blocks with a dashed border represent new diagram types, namely, the *requirements* and *parametric diagrams*.

2.2 Relevant Behavioral Modeling Formalisms

Systems engineers currently use several different formalisms for modeling continuous system dynamics. One such formalism models continuous dynamics using signal blocks in an assignment-based, causal approach. The Matlab/Simulink language is an example of such a formalism. The Matlab language relies on algorithmic and block diagram representation of behavior. In such a representation, system structure and

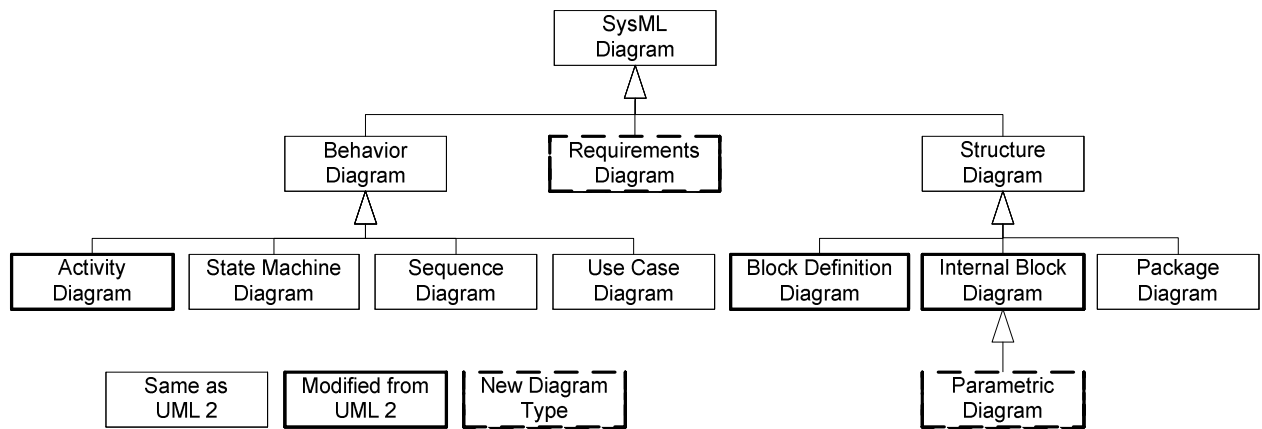


Figure 2. The SysML diagram taxonomy [5].

behavior are loosely connected in sets of expression-based assignments [9].

Alternatively, other formalisms for representing continuous system dynamics rely on acausal, equation-based modeling and object-oriented system decomposition. Modelica is an example of such a formalism. The language is commonly used to model and simulate system behavior in many different engineering domains (e.g., multi-body mechanics, electronics, hydraulics, or controls). Behavior is modeled using differential and algebraic equations.

Being an equation-based language, Modelica does not require a user to assign causality. A Modelica solver has the capability to solve equations and automatically assign causality. This is advantageous when dealing with energy-based systems. For example, consider an arbitrary system with a rotational-mechanical input shaft. Suppose that the system is initially planned to be driven by a constant torque source. When solving the DAE describing the system dynamics, the torque must be considered as the independent variable. If system designers must later consider an input based on shaft speed, then the independent variable must be changed to angular velocity. A causal model then requires significant change while an acausal model does not change at all. Hence, acausal models of continuous system dynamics are valuable due to their promotion of model reuse.

2.3 Relevant Work on the Integration of UML and Modelica

Recently, Fritzson and Pop [10] have worked on issues regarding the integration of UML and Modelica. They have created a UML profile called ModelicaML that enables users to graphically depict a Modelica simulation model. Its intent is similar to ours. The ModelicaML profile reuses several UML and SysML constructs while creating completely new language constructs. Such constructs are the Modelica class diagram, the equation diagram, and the simulation diagram.

Nytsch-Geusen [11] developed a specialized version of UML called UML^H. This version is used in the graphical description and model-based development of hybrid systems in Modelica. The author presents hybrid system models as Modelica models that are based on DAE's combined with

discrete state transitions modeled using the Modelica statechart extension. Using a UML^H editor and a Modelica tool that supports code generation, Modelica stubs can be automatically generated from UML^H diagrams so that the user must only insert the equation-based behavior of the system in question.

While ModelicaML, UML^H, and our SysML modeling approach attempt to integrate behavioral models into broader information models, our approach utilizes existing SysML language constructs, which are already intended to integrate models created using various formalisms. In contrast with the developers of ModelicaML and UML^H, we integrate Modelica models by extending the semantics of SysML instead of by creating a separate UML profile with new language constructs.

3 OBJECTIVES FOR THE BEHAVIORAL MODELING APPROACH

Why do we need another behavioral modeling approach in SysML? SysML describes the functions and behaviors of a system by including the following language constructs:

- SysML *Activity diagrams* describe the inputs, outputs, sequences, and conditions for coordinating various system behaviors
- *Sequence diagrams* describe the flow of control between actors and a system or its components
- *State machine diagrams* are used for modeling discrete behavior through finite state transition systems
- *Parametric diagrams* allow users to represent mathematical constraints amongst system properties

The first three of these modeling constructs promote causal behavioral modeling in terms of discrete events. The last one enables a user to model algebraic equations that establish mathematical relationships between system properties. While many types of system behavior can be modeled using these constructs, direct support for modeling continuous dynamics is not provided. By extending the semantics of various SysML constructs and adding reusable modeling elements, we can establish a well-defined approach for modeling continuous dynamics and add functionality to the language.

Why should Modelica be the base language for our SysML modeling approach? Due to the high degree of similarity between SysML and Modelica constructs for building system models, a formal approach for representing dynamic system behavior in SysML can be based on a bidirectional mapping between the two languages. In accordance with the systems engineering Vee model, both SysML and Modelica promote the decomposition of complex systems into discrete components and subsystems that are later integrated into a system model. Additionally, the base objects in both languages share similar semantic structures. Both object structures are composed of properties that define its parts and characteristics. They can also both utilize interface objects for modeling interactions with other objects. Aside from SysML's ability to model general information, the main differences between the languages reside in their constructs for modeling system dynamics. A semantic mapping between the two languages supports the extension of SysML's capability to model system dynamics based on DAE's.

How expressive should the modeling constructs be? The information obtained from a model is valuable if it increases a decision maker's ability to design a better system at an acceptable cost [12]. We can ensure that the formal modeling constructs encourage the development of valuable models by providing enough means for a user to create models at the necessary level of abstraction. As we explain later in this section, the modeling constructs will provide value if they meet the following objectives:

- The constructs must enable the integration and bidirectional transformation of SysML and Modelica models
- The constructs must encourage model reuse
- The constructs must facilitate efficient stakeholder communication

By establishing these objectives for our constructs, we intend to strike a balance between the benefits expected from formalizing a model and the costs of encoding the required information and knowledge.

Model integration is essential for managing system complexity. By integrating a Modelica-based behavioral model, systems engineers can establish important relationships between a system's behavior and its other aspects. Hence, a systems engineer can formally recognize relationships between a particular model of continuous dynamics and other models of system behavior, structure, or function (e.g., system architecture, cost, mass, reliability, requirements, or test models).

The formal approach must also facilitate bidirectional model transformation between SysML and Modelica. Bidirectionality enables the transformation of SysML design models into Modelica analysis models that can be simulated by a Modelica compiler and runtime equation solver. Alternatively, bidirectionality enables the transformation of Modelica models into SysML semantics. If a modeler has created a Modelica behavioral model of a system or component, he or she can integrate the model into the appropriate SysML project.

A formal approach for representing continuous dynamics in SysML must also establish a standard that encourages model reuse. If a designer can avoid creating every model from scratch by reusing or modifying preexisting models, he or she can realize significant reductions in the use of project resources. This reuse is contingent upon the assumption that the reusable models have been created using a well-defined formalism.

The final objective is to create an approach that provides a common ground upon which project stakeholders can communicate efficiently. Unambiguous communication is very important during the development of a complex system. By using a formal approach for representing system behavior in SysML information models, behavioral knowledge can be unambiguously shared amongst designers operating in various engineering disciplines. Additionally, a formal approach enables other individuals, such as project managers, target consumers, or expert consultants, to contribute effectively to the design effort.

4 SYSML BEHAVIORAL MODELING OVERVIEW

In this section, we establish the formal approach for modeling continuous system dynamics in SysML. We satisfy our objectives for the modeling approach by establishing equivalent SysML modeling formalisms for every necessary element of the Modelica language specification. Such elements include the declaration of a model and its interface, properties, reuse abilities, internal composition, and internal behavior.

4.1 Model Declaration

The fundamental similarity between SysML and Modelica is the use of objects. The primary modeling unit in Modelica is the *class*. Classes provide the structure for objects and serve as templates for creating objects from class definitions [13]. Classes are defined by their properties and equation-based behavior.

In our approach, a Modelica class maps directly to a SysML *block*, which is the primary modeling unit in SysML. As described in Chapter 8 of [5], a block is a modular unit of a system description. A block can represent *anything*, whether tangible or intangible, that describes a system. For instance, a block could model a system, subsystem, part, characteristic, or process. When combined together, blocks define a collection of features that describe a system or other object of interest. Hence, blocks provide a means for a system designer to decompose a system into a collection of interacting objects.

To make Modelica easier to read and maintain, special class restrictions were developed for expressing the intended function of a class [7]. Example restrictions are models, connectors, types, and functions. While the restrictions are useful, they are not necessary in most cases. One can usually maintain model validity by replacing a restricted class with a regular class. Exceptions addressed in this paper are the *connector* and *type* restrictions (discussed later in this section).

Figure 3 and Figure 4 illustrate the equivalence of SysML blocks and Modelica classes. Figure 3 shows a textual Modelica model of a constant-displacement pump. The pump is modeled as a system consisting of an ideal pump, two fluid volumes, a

```

model ConDispPump
  IdealConDispPump idealConDispPump;
  VolTherm vol_a;
  VolTherm vol_b;
  FluidPort_a port_a;
  FluidPort_b port_b;
  LaminarResistance leakage;
  Rotational.Interfaces.Flange flange_b;
  Rotational.Interfaces.Flange flange_a;
equation
  connect(volA.port_a, port_a1);
  connect(volB.port_b, port_b1);
  connect(idealConDispPump.port_b, volB.port_a);
  connect(idealConDispPump.port_a, volA.port_b);
  connect(vol_a.port_b, leakage.port_a);
  connect(vol_b.port_a, leakage.port_b);
  connect(idealConDispPump.flange_b, flange_b);
  connect(idealConDispPump.flange_a, flange_a);
end ConDispPump;

```

Figure 3. Modelica textual description of a constant-displacement hydraulic pump.

leakage, two rotational flanges, and two fluid ports. Although the pump is modeled using the class restriction “model”, bidirectionality can be maintained by a SysML block that does not recognize the restriction. As seen in Figure 4, the SysML equivalent is a block named *ConDispPump* that exists in a *block definition diagram* (BDD). A BDD depicts the features of a block and the various relationships between blocks or other SysML constructs. In this case, the features of the pump are the various blocks used as *part properties* (discussed in Section 4.2). The relationships in the model include the *composition* relationship (discussed in Section 4.2) and the *specialization* relationship (discussed later in this section).

Model Properties

In Modelica, internal properties of a model are called *components*. A component can represent a part (discussed in Section 4.2) or an internal characteristic (e.g. mass, length, or position). One can tell whether a component represents a part or a characteristic by identifying the class to which the component is typed. Characteristic components are usages of classes with the *type* restriction. These classes map directly to SysML *value types*. Both Modelica types and SysML value types can be used to assign the units of measure or dimension declared in its definition. For example, in Figure 5, *PartialPumpMotor* has a variable *tau* that is a usage of the *SI.Torque* class. The definition of *SI.Torque* states that it has units of “N-m”. Figure 4 shows the declaration of the *SI.Torque* value type in SysML, along with its usage in the declaration of the *PartialPumpMotor* block.

Since Modelica characteristic components and SysML *value properties* are characterizations of their owning objects, a one-to-one mapping exists between the two. This mapping is illustrated by Figure 4 and Figure 5. Except for the two flange connector components, everything in the declaration of the *PartialPumpMotor* Modelica model is a component typed to a class with a type restriction. These components are mapped to SysML by means of the value properties shown in the *values* compartment of the *PartialPumpMotor* block.

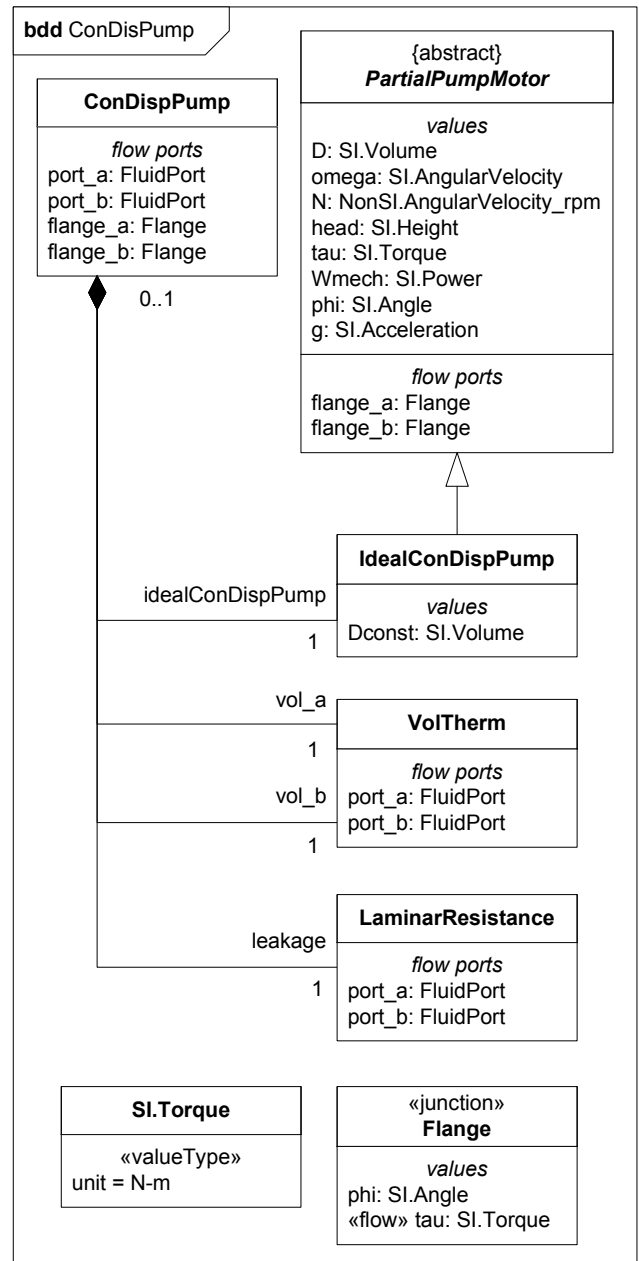


Figure 4. Block definition diagram of the SysML behavioral model of a constant-displacement, hydraulic pump.

Model Interface

For a model to interact with other models, it must have a well-defined interface. In Modelica, a model’s interface consists of components typed to connectors. Connectors are restricted classes that hold across and through variables, but have no equations defining their behavior. Across and through variables characterize the flow of energy through an object. For example, when describing hydraulic energy, pressure is an across variable while volumetric flow rate is a through variable. To describe continuous interaction between two classes, connector components are connected using *connect statements* (discussed in Section 4.2). When two connectors are

```

model IdealConDispPump
  "Ideal constant displacement pump with
  mechanical connector for the shaft"
  extends PartialPumpMotor;
  parameter SI.Volume Dconst = 0.001 "Constant
  pump displacement";
equation
  D = Dconst;
end IdealConDispPump;

partial model PartialPumpMotor
  "Partial model for displacement pumps or motors"
  extends FluidPower.Interfaces.PartialTwoPort;
  // the variables
  SI.Volume D "Pump displacement";
  SI.AngularVelocity omega "Shaft angular
  velocity";
  NonSI.AngularVelocity_rpm N=Cv.to_rpm(omega)
  "Shaft rotational speed";
  SI.Height head "Pump head";
  SI.Torque tau "Torque needed for pumping fluid";
  SI.Power Wmech "Mechanical power";
  constant SI.Acceleration=Modelica.Constants.g_n;
  SI.Angle phi "Shaft angle";
  // the connectors
  Rotational.Interfaces.Flange flange_a;
  Rotational.Interfaces.Flange flange_b;
equation
  // Flow equation
  q_flow_a = D*N/60;
  head = dp/(medium_b.d*g);
  ...
end PartialPumpMotor;

```

Figure 5. Modelica textual model of an ideal, constant-displacement, hydraulic pump and a partial model of a hydraulic pump/motor.

connected, their across variables are equal while their through variables add to zero. Only components typed to connector classes can take part in this relationship.

In SysML, blocks can encapsulate their internal properties, composition, and behavior by interacting with other blocks using *flow ports*. As described in Chapter 9 of [5], a flow port is an interface property through which a block can exchange data, material, or energy with other blocks. Flow ports can either be *atomic* or *non-atomic*. An atomic flow port characterizes a single item that flows through a block, while a non-atomic flow port is used to characterize the flow of multiple items.

Modelica connectors typically represent only one type of continuous interaction between two classes. If a Modelica class must partake in more than one interaction, more connectors are added to its definition. When a connector remains unconnected, nothing can flow through it and the corresponding through variable is therefore set equal to zero.

This practice is equivalent to a SysML block using multiple atomic flow ports for describing its interactions with other blocks. Usages of atomic flow ports are typed to single, external block definitions that contain across and through variables in the form of value properties. In accordance with the Modelica *flow prefix* assigned to components representing through variables, a «flow» stereotype is applied to value properties representing through variables. A stereotype is a UML construct for adding customized classifications of

modeling elements. Such customizations extend the standard elements to identify more specialized cases important to specific classes of applications.

To ensure the preservation of bidirectionality, the block definitions to which the flow ports are typed must include a «junction» *stereotype*. SysML already uses the term “connector” (discussed in Section 4.2) to mean something other than a Modelica connector, so we use «junction» as a different term in SysML to maintain the mapping to Modelica connectors.

Figure 5 shows the declaration of connectors for the pump example in Modelica. *PartialPumpMotor* owns two connectors called *flange_a* and *flange_b*, which are defined by the *Flange* connector type defined outside of *PartialPumpMotor*. Figure 4 shows the corresponding elements declared in a SysML block definition diagram. The *PartialPumpMotor* block owns two flow ports called *flange_a* and *flange_b* that are typed by a block called *Flange*. The external definition of the *Flange* block would include the stereotype «junction» to indicate their specialized usage to define energy flows. Note that these semantics are slightly different from those specified in Chapter 9 of [5]. To maintain the parallelism with Modelica, we type the flow port to a «junction» rather than to the item that flows through the junction.

Abstract Models, Inheritance, and Redefinition

Both languages support model reuse through the object-oriented concepts of abstract classes, inheritance, and redefinition. If a Modelica class is tagged with the *partial* keyword, the class is not fully defined and cannot be instantiated. Instead, the class serves as a template that can be extended through object inheritance. Similarly, SysML supports the concept of an *abstract* block acting as a partially defined model. Commonly, these partial or abstract definitions would be extended by means of a specialization relationship such that a new high-fidelity object could be defined in a system model or component library.

Blocks can be extended through the use of specialization relationships such that a child block inherits and refines the structure of a parent block. The same inheritance relationship exists in Modelica by means of the *extends clause*.

Figure 5 illustrates the concepts of a partial class and class inheritance in Modelica. As seen in the figure, the model *IdealConDispPump* extends the partial model *PartialPumpMotor*. The equivalent SysML semantics can be seen in the BDD in Figure 4. The SysML block *PartialPumpMotor* is partially defined due to *{abstract}* appearing in the block’s namespace. That block is extended by a specialization relationship (the white arrow) between it and the *IdealConDispPump* block.

Modelica also supports model reuse through the use of replaceable properties and their redeclaration. A Modelica class can have components that are tagged by the *replaceable* keyword. This allows the component to be redefined using the *redeclare* construct when its owning class is typed by a component in another class. In SysML, every property of a

block is considered to be replaceable using standard UML mechanisms of redefinition.

4.2 Composition of the Internal Structure

The internal structure of a Modelica class is represented by connections between the connectors of its part components. Part components are components typed to classes without type restrictions (e.g. classes, models, or connectors). Connections between components typed to connectors are made using *connect statements* that reside in the *equation clause* of the owning class. Modelica classes share across and through variables using these connect statements. If two elements are connected, their across variables are equal and their through variables add to zero. Such a connection implicitly defines the existence of the above two equations.

To map this concept to SysML, we must first present a SysML representation of the Modelica part component. Since this type of component is a part of a model, it is mapped to a SysML *part property*. A part property is a property that must exist to completely define its owning block. Parts are typed by blocks that describe system components or subsystems.

Figure 3 and Figure 4 depict Modelica part components and SysML part properties, respectively. In Figure 3, the *ConDispPump* Modelica model includes eight components. These components are the ideal pump, two fluid volumes, laminar resistance, two fluid ports, and two mechanical flanges. We must note that the fluid ports and mechanical flanges are typed to connector classes. Accordingly, the connectors map to SysML flow ports while the other four components map to part properties.

SysML part properties can be depicted in two fashions that are semantically equivalent. Part properties can show up in a block compartment labeled *parts*. However, in Figure 4, we depict the other fashion. This fashion involves the placement of a composition relationship (black diamond with a tail) between the owning block and its part. At the end of the tail, the user must specify the usage name and the multiplicity (the number of equal usages). In Figure 4, composition relationships are present between the *ConDispPump* block and the *IdealConDispPump*, *VolTherm*, and *LaminarResistance* blocks. The multiplicity for each usage is one. The usage names are consistent with the Modelica component names. Two separate composition relationships are attached to the *VolTherm* block to indicate that the two usages are not equal.

Once the necessary components are represented in a SysML behavior model, connect statements must be modeled between appropriate pairs of connectors belonging to part components. Due to the mapping between Modelica connectors and SysML typed flow ports, we rely on the semantic construct used to describe interacting flow ports. This construct is the SysML *connector* (depicted by a simple line, not to be confused with the Modelica connector) and it appears in an *internal block diagram* (IBD). An IBD describes the internal structure of a block in terms of connections between its properties or between the ports of its parts. In our modeling approach, a SysML connector placed between two flow ports implies that the flow

ports are exchanging energy or signals by means of across and through variables.

Figure 6 shows a graphical representation of the textual Modelica model seen in Figure 3. The IBD of the SysML pump model, shown in Figure 7, parallels the structure seen in Figure 6. For each connect statement in the Modelica model, an equivalent SysML connector is depicted in the IBD.

4.3 Equation-based Internal Behavior

The internal behavior of a Modelica class is defined by placing necessary equations in the equation clause of a class. An equation is used to create relationships between characteristic components. Equations are normally differential or algebraic in nature, but can also include the use of conditional logic. As an aside, Modelica does handle assignment-based, causal behavioral modeling; however, it is relied on less frequently than equation-based modeling and is not addressed in this paper.

Before we address equation modeling in SysML, we must first describe the *constraint block*. As seen in Chapter 10 of [5], blocks can own constraint blocks as properties in order to describe parametrically the relationships among a block's properties. A constraint block used by a block or another constraint block is known as a *constraint property*. A constraint block is composed of its *constraints* and *parameters*. A constraint is a mathematical construct, such as an expression or equation. A parameter is a property of the constraint block used in the constraint.

To depict parametric models created using constraint properties, the SysML IBD has been extended by the parametric diagram. In a parametric diagram, constraint properties constrain the value properties of the owning block using *binding connectors*. A binding connector is simply a connector that implies equality. Hence, when a value property is connected to the parameter of a constraint property, the two are constrained to be equal.

In SysML, the basic form of an equation can be represented using combinations of primary operators and block properties. A primary operator could be addition, subtraction, multiplication, or division. Additionally, other simple yet essential operators are incorporated (e.g., time derivative, maximum value, or if-then-else). Such operators are modeled using constraint blocks with constraints that represent the primary operator. For example, a multiplication constraint

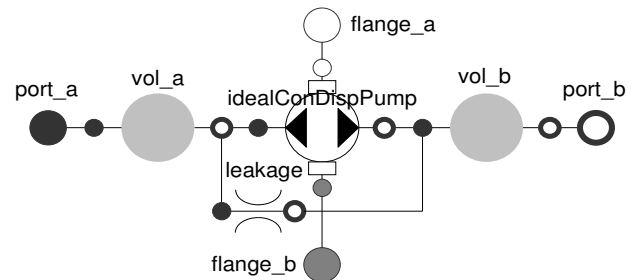


Figure 6. Modelica graphical representation of the textual model of a constant-displacement, hydraulic pump seen in Figure 3.

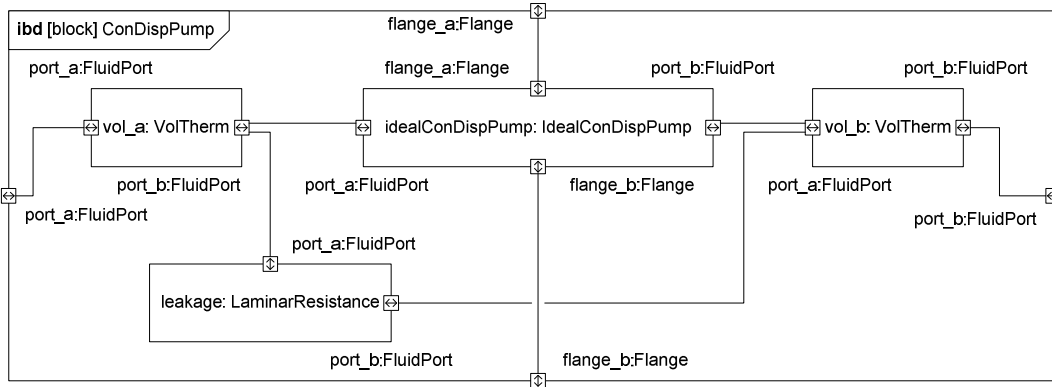


Figure 7. Internal block representing the internal structure of the *ConDispPump* SysML model.

block would have the constraint $\{op1 * op2 = result\}$. Once a large collection of these constraint block operators are created, they can be stored in a library such that a user could create equations by combining the corresponding constraint properties in a block's PAR.

While modeling equations as combinations of primary operators is most basic and promotes model reuse, the modeling process can become overwhelming when attempting to represent large equations or sets of equations. Hence, we also introduce simplified constructs for equation modeling. Constraints can exist in a textual form that relies on Modelica's mathematical syntax. This is accomplished by setting a full Modelica equation or set of equations as the constraint of a single constraint block. Consequently, the parameters in that constraint must be depicted as parameters of the corresponding constraint block. SysML constraint blocks already support a language field that indicates the textual language in which a constraint is stated. To contain Modelica-based constraints, the language field of a constraint block must be set to "Modelica," recognizing the dependence on Modelica syntax.

Even though the simplified constructs for equation modeling are easier to use when representing large equations, they do not encourage model reuse. If a block has three different equations representing its internal behavior, a user will likely be forced to create constraint properties from three original constraint blocks. The decision to utilize either the basic or simplified constructs for equation modeling is left to the user.

To solve DAE's, Modelica relies on parameter initialization and initial equations that exist in the *initial equation clause*. This clause is similar to the equation clause, but provides start values for a Modelica solver. Initial equations are modeled in SysML by placing an «initial» stereotype on a constraint property. This stereotype implies that the constraints are just like initial equations in Modelica.

To demonstrate these constructs for equation modeling, the PAR in Figure 8 depicts the two equations in the *PartialPumpMotor* Modelica model from Figure 5. At the top of the diagram, we demonstrate the simplified approach to equation modeling. Three value properties are connected to a constraint property named *flowEqn*. The three parameters of *flowEqn* are related to each other as specified in the constraint shown in the attached note expressed in Modelica syntax.

At the bottom of the PAR, we demonstrate equation modeling using primary operators. The head equation is created by making appropriate connections between value properties and parameters and then connecting parameter *c* of the *Multiplication* constraint property to parameter *b* of the *Division* constraint property. From top to bottom, the equation reads as follows:

1. Parameter *c* of the *Multiplication* constraint property is equal to *medium_b.d* multiplied by *g*.
2. Parameter *b* of the *Division* constraint property is equal to parameter *c* of the *Multiplication* constraint property.
3. *head* is equal to *dp* divided by parameter *b* of the *Division* constraint property.

Since this is an acausal equation, this is only one interpretation. The equation could also be read from bottom to top.

4.4 Composition of a System Behavioral Model

To compose a continuous dynamics behavioral model of a system, a user must represent the components of a system, the connections between their interfaces, and the equations that constrain their properties. A SysML system model will typically start with a high-level model composed solely of part properties and connectors between component flow ports. SysML allows the total definition of a system to be split across many diagrams, to emphasize different levels or aspects of an overall system being described.

The SysML model of the constant-displacement, hydraulic pump is itself a behavioral model of a system. However, since it has ports on its periphery, it can also be used as a subsystem. A true system model could be represented by connecting the pump model to models of a tank, control valve, hydraulic lines, or various actuators such that the result is a larger behavioral model with no ports *or* ports that permit connections between the environment or system actors.

5. SUMMARY

In this paper, we have introduced a formal approach to modeling dynamic system behavior in SysML by means of a language mapping between SysML and Modelica. This approach provides support for modeling system behavior based on the exchange of energy and signals. Several behavioral modeling constructs already exist as part of the SysML

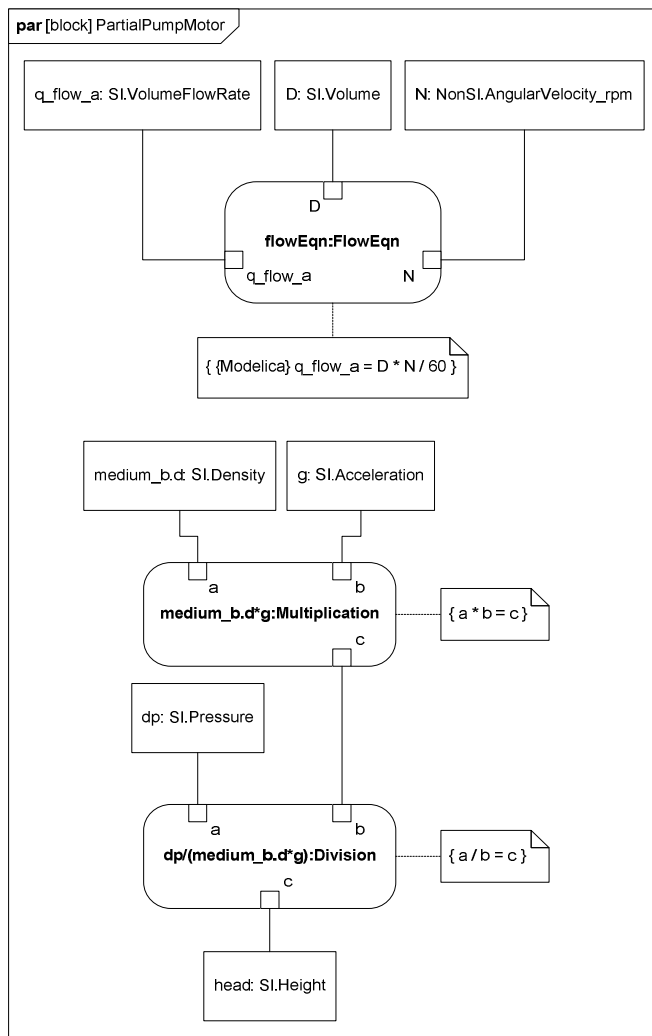


Figure 8. Parametric diagram of the SysML behavioral model of the *PartialPumpMotor*.

specification, but none of them are directly capable of modeling DAE's. We have shown that existing elements of SysML can be adapted for this purpose. To better understand the level at which Modelica semantics should be represented in SysML, we first established our objectives for a formal modeling approach. We concluded that the primary objective is to enable the integration of Modelica models in SysML. Upon further establishing our objectives, we described our modeling approach in a fashion that satisfied those objectives. We must note that the modeling overview in this paper is only a partial description of the mapping-based approach. Instead of describing every fine detail, we attempted to convey the overall approach by addressing its most important features.

Modeling and understanding system behavior and its effects on other system aspects is of great importance in ensuring that a system meets stakeholder objectives and requirements. By establishing support for the integration of continuous behavior models in SysML, we hope to improve SysML's ability to aid designers during the development of contemporary, complex systems.

ACKNOWLEDGMENTS

This material is based in part upon work performed within the ERC for Compact and Efficient Fluid Power, supported by the National Science Foundation under Grant No. EEC-0540834. Additional supported was provided by Deere & Company.

REFERENCES

- [1] Sage, A. P., and Armstrong Jr., J. E., 2000, *Introduction to Systems Engineering*, John Wiley & Sons, Inc., New York, NY.
- [2] Forsberg, K., and Mooz, H., 1992, "The Relationship of Systems Engineering to the Project Cycle," *Engineering Management Journal*, 4(3), pp. 36-43.
- [3] Forsberg, K., and Mooz, H., and Cotterman, H., 2000, *Visualizing Project Management: A Model for Business and Technical Success*, John Wiley & Sons, Inc., New York, NY.
- [4] Gero, J. S., 1990, "Design Prototypes: A Knowledge Representation Schema for Design," *AI Magazine*, 11(4), pp. 26-36.
- [5] Object Management Group, 2007, "OMG Systems Modeling Language Specification," <http://www.omg.org/cgi-bin/doc?ptc/07-02-03>.
- [6] Aughenbaugh, J. M., and Paredis, C. J. J., 2004, "The Role and Limitations of Modeling and Simulation in Systems Design," *ASME International Mechanical Engineering Congress and RD&D Expo*, ASME, Anaheim, CA USA.
- [7] Modelica Association, 2005, "Modelica Language Specification," <http://www.modelica.org/documents/ModelicaSpec22.pdf>.
- [8] ISO/IEC, 2005, "Unified Modeling Language Specification," <http://www.omg.org/cgi-bin/apps/doc?formal/05-04-01.pdf>.
- [9] Blaha, M. R., and Rumbaugh, J. R., 2004, *Object-Oriented Modeling and Design with UML*, Prentice Hall, Upper Saddle River, NJ.
- [10] Pop, A., and Akhvlediani, D., and Fritzson, P., 2007, "Towards Unified Systems Modeling with the ModelicaML UML Profile," in *International Workshop on Equation-Based Object-Oriented Languages and Tools*, Linköping University Electronic Press, Berlin, Germany.
- [11] Nytsch-Geusen, C., 2007, "The Use of UML within the Modelling Process of Modelica-Models," in *International Workshop on Equation-Based Object-Oriented Languages and Tools*, Linköping University Electronic Press, Berlin, Germany.
- [12] Keeney, R. L., 1994, "Creativity in Decision Making with Value-Focused Thinking," *Sloan Management Review*, 35(4), pp. 33-41.
- [13] Fritzson, P., 2004, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, Piscataway, NJ.