Integrating Models and Simulations of Continuous Dynamics into SysML

Thomas Johnson[1] Christiaan J.J. Paredis[1] Roger Burkhart[2]

[1]Systems Realization Laboratory
The G. W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
tjohnson6@gatech.edu chris.paredis@me.gatech.edu

[2]Deere & Company World Headquarters
Moline, Illinois 61265
BurkhartRogerM@johndeere.com

Abstract

In this paper, we combine modeling constructs from SysML and Modelica to improve the support for Model-Based Systems Engineering (MBSE). The Object Management Group has recently developed the Systems Modeling Language (OMG SysML™). This visual modeling language provides a comprehensive set of diagrams and constructs for modeling many common aspects of systems engineering problems, such as system requirements, structures, functions, and behaviors. Complementing these SysML constructs, the Modelica language has emerged as a standard for modeling the continuous dynamics of systems in terms of hybrid discrete-event and differential algebraic equation systems. In this paper, the synergy between SysML and Modelica is explored at three different levels: the definition of continuous dynamics models in SysML; the use of a triple graph grammar to maintain a bi-directional mapping between these SysML constructs and the corresponding Modelica models; and the integration of simulation experiments with other SysML constructs to support MBSE. Throughout the paper, an example of a car suspension is used to demonstrate these contributions.

Keywords: SysML; Modelica; model-based systems engineering; continuous dynamics; graph transformations

1 Introduction

1.1 Managing System Complexity with SysML

Contemporary systems engineering projects are becoming increasingly complex as they are handled by geographically distributed design teams, constrained by the objectives of multiple stakeholders, and inundated by large quantities of design information. Accordingly, problems encountered during the system development process generally 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 engineers cannot efficiently manage project complexity, they might overlook important design details and dependencies. Such mistakes can compromise stakeholder objectives and lead to costly design iterations or system failures.

According to the principles of model-based systems engineering (MBSE) [2], engineers can overcome these problems by replacing document-centric design methods with model-based approaches for representing and investigating their knowledge during system decomposition and definition. Models can be used to represent formally all aspects of a systems engineering problem, including the structure, function, and behavior of a system [3]. Additionally, experiments can be performed on models to eliminate poor design alternatives and to ensure that a preferred alternative meets the stakeholders' objectives. Models also facilitate collaboration by providing a common, unambiguous protocol for communicating design information.

To support MBSE, the Object Management Group has recently developed the Systems Modeling Language (OMG SysML™). SysML is a general-purpose systems modeling language that enables systems engineers to create and manage models of engineered systems using well-defined, visual constructs [4]. Instead of developing SysML as an original design, the OMG adapted the successful Unified Modeling Language (UML) to the systems engineering

field. UML is most commonly used during the development of large-scale, complex software for various domains and implementation platforms [5]. To support an application base that extends beyond software engineering, SysML reuses and extends a subset of UML 2.1 constructs:

- it extends UML classes into *blocks*;
- it enables *requirements modeling*;
- it supports *parametric modeling*;
- it extends UML dependencies into *allocations*;
- it reuses and modifies UML *activities*;
- it extends UML standard ports into *flow ports*.

Through these extensions, SysML is capable of representing many common, yet essential aspects of both system hardware and software.

1.2 Modeling System Behavior with SysML

The knowledge captured in a SysML model is intended to support the specification, analysis, design, and verification and validation of any engineered system [4]. As a result, SysML is commonly used to model system requirements, tests, structures, functions, behaviors, and their interrelationships. Although all of these models are important for ensuring project success, 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 other aspects.

SysML currently depicts system behavior using the following language constructs:

- *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 equations (called "constraints" in SysML) that establish mathematical relationships between system properties. In this paper, the focus is on *parametric diagrams* and specifically on the representation of the continuous dynamics of engineered systems within parametric diagrams. Such models are composed of differential algebraic equation (DAE) systems that represent the exchange of energy, signals, or other continuous interactions between system components. By relying on Modelica syntax and semantics, we demonstrate how such DAE systems can be modeled with only a few extensions to the basic SysML constructs (see Section 4). SysML then serves as an integration framework in which detailed Modelica models can be related to other types of systems engineering knowledge (see Section 6). The integration between SysML and Modelica creates a significant synergy: SysML benefits from the detailed Modelica semantics for representing DAE systems combined with discrete events; Modelica benefits from the broader information modeling context provided in SysML, a context that is crucial for establishing formal, unambiguous communications between systems engineers, disciplinary designers and systems analysts. To maintain consistency between the Modelica models and their corresponding abstractions in SysML, we introduce the use of triple graph grammars (TGGs) [6] to specify transformations between the two forms of models (see Section 5).

2 Related Work

The need to describe system behavior in terms of equations or constraints has been previously recognized in the work on Constrained Objects (COB's) [7, 8]. COBs provide both a graphical and lexical representation of algebraic relationships that can be used to tie design models to analysis models in a parametric fashion. These COBs recently served as the basis for the development of the SysML parametric diagrams [4]. By establishing a mapping between COBs and SysML, the integration and execution of engineering analyses (such as structural finite element analyses) within the context of SysML has been demonstrated [9]. This paper extends this past work on COBs by focusing on the modeling and simulation of the continuous dynamics of systems as defined in Modelica models.

Recently, Fritzson and Pop [10] have worked on the integration of UML/SysML and Modelica to provide support for modeling and simulating continuous dynamics. They have created a UML profile called ModelicaML that enables users to depict a Modelica simulation model graphically alongside UML/SysML information models. The ModelicaML profile reuses several UML and SysML constructs, but also introduces 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 DAEs combined with discrete state transitions modeled with 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. In this paper, the capabilities of ModelicaML and UML^H are further extended by demonstrating the integration of continuous dynamics models with other SysML constructs for requirements, structure, and design objectives, and by demonstrating the translation between SysML and Modelica through the use of TGGs.

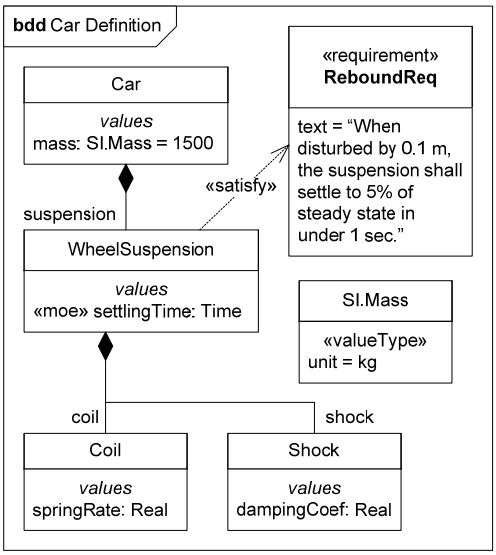

Figure 1. The SysML car suspension model.

3 An Introduction to SysML: The Car Suspension Model

Before discussing the approach for modeling continuous dynamics and simulations in SysML, this section reviews some important SysML constructs and introduces the example problem used throughout this paper.

3.1 SysML Blocks

The primary modeling unit in SysML is the *block*. As described in chapter 8 of the SysML specification [4], 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, process, function, or context. When combined together, blocks define a collection of features that describe a system or other object of interest. Hence, blocks provide a means for an engineer to decompose a system into a collection of interrelated objects.

All block declarations occur in a *Block Definition Diagram* (BDD). A BDD is used to define block features and the relationships between blocks or other SysML constructs. Figure 1 depicts the definition of a car and its suspension. A car is obviously composed of more subsystems and components, but Figure 1 is sufficient for the sake of demonstration. SysML allows a modeler to omit elements of the underlying information model that detract from the main intent of a diagram.

3.2 SysML Properties

A SysML property describes a part or characteristic of a block and consists of a named value of a specified type. In Figure 1, two important categories of properties are depicted. The first kind of property is a *part property*. Part properties represent a subsystem or component of a system and must be typed by a block. Part properties can be depicted in the *parts compartment* of a block or using a *composition association*. A composition association is depicted using a black diamond with a tail. The property name appears at the tail end of the association. For example, the block *Car* in Figure 1 owns a part property named *suspension* of type *WheelSuspension*.

The second kind of property is a *value property*. A value property appears in a block's *values compartment* and represents a quantifiable characteristic of a block (e.g. mass, length, velocity) and must be typed to a SysML *value type*. A value type is a special modeling element (similar to a block) used to assign the units of measure and dimension declared in its definition. For example, *Car* in Figure 1 has a value property *mass* which is typed to the value type *SI.Mass* to supply units of kilograms.

3.3 UML Stereotypes

A *stereotype* is a UML construct used to create customized classifications of modeling elements. Stereotypes are defined by keywords that appear inside of guillemets. These customization constructs extend the standard elements to identify more specialized cases important to specific classes of appli-

cations. Most SysML constructs have been defined as UML stereotypes, and users are allowed to create additional stereotypes to capture the specialized semantics of a particular application domain. An example of a stereotype is illustrated in Figure 1. The stereotype *«moe»* applied to the *WheelSuspension*'s value property *settlingTime* indicates that it is a measure of effectiveness.

3.4 SysML Requirements

A SysML *requirement* is used to represent a textual requirement or objective for a system, subsystem, or component. Requirements are shown with the *«requirement»* stereotype and optionally have a compartment for displaying text and identification fields. Requirements are related to other modeling elements using various dependencies such as the *satisfy* and *verify* dependencies.

4 Modeling Continuous Dynamics in SysML

In this section, the approach to modeling continuous dynamics in SysML is presented. The approach builds on the initial modeling foundation outlined in [12]. Rather than elaborating upon every detail, only the most important modeling constructs are discussed.

4.1 Objectives

A model is valuable if it increases a decision maker's ability to design a better system at an acceptable cost [13]. As explained later in this section, the continuous dynamics modeling constructs will provide value if they meet the following objectives:

- Enable the integration of continuous dynamics models into broader SysML models;
- Facilitate the execution (i.e., simulation) of these continuous dynamics models;
- Encourage model reuse;
- Facilitate efficient stakeholder communication.

The intent of these objectives is to strike an appropriate balance between the benefits expected from developing a model and the costs of encoding the required information.

Model integration is essential for managing system complexity through recognition and establishment of dependencies and associations between models of continuous dynamic system behavior and other models of system behavior, structure, or functionality. SysML is a language for describing systems engineering information and knowledge, but is by itself not executable—model execution is relegated to an editing and execution tool. To be effective, it is therefore important to establish seamless connections between SysML and simulation tools. Model reuse is another imperative for realizing significant reductions in project resource expenditures. Finally, using a unified approach for representing continuous dynamics in SysML establishes a protocol for unambiguous communication of behavioral information between designers operating in various engineering disciplines.

4.2 Modelica as a Foundation

When creating a formal approach for representing continuous dynamics in SysML, Modelica provides a strong foundation. Modelica has emerged as the language of choice for expressing continuous dynamic system behavior. It is better structured and more expressive than most alternatives such as VHDL-AMS [14] or ACSL [15]. In addition, both SysML and Modelica are similar in that they use base modeling elements that adhere to the principles of object-oriented modeling. Both languages also encourage model reuse through acausal equation-based modeling. Unfortunately, enough differences exist such that a direct one-to-one mapping is not possible. Since SysML is intended to be a general modeling language, some of the specialized semantics of Modelica do not have a direct equivalent in SysML. To overcome these differences, our approach has been to find a good balance between converting some implicit Modelica semantics into explicit constraints in SysML or, when that is not possible, extending the SysML constructs through stereotypes.

4.3 Model Declaration

When modeling continuous dynamic system behavior, a modeler must first declare the model that represents the system of interest. This involves specifying the blocks and properties needed to decompose the system to an appropriate level of abstraction. The level of abstraction is determined by the amount of detail needed to perform an acceptable system analysis. This declaration approach is analogous to creating Modelica classes that own components and variables typed to other class definitions.

To illustrate model declaration, Figure 2 displays the declaration of a continuous dynamics model of a Mass-Spring-Damper (MSD) system. This model will be used in Section 6 to perform a behavioral

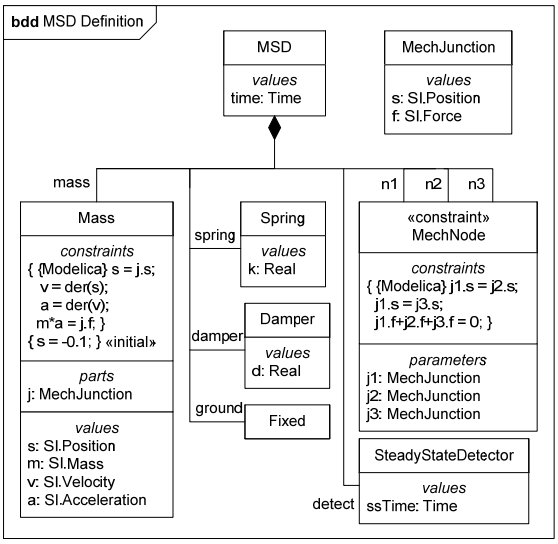

Figure 2. BDD of the *MSD* continuous dynamic system behavior model.

analysis on the car suspension model from Figure 1. The MSD system is composed of a mass, spring, damper, fixed position (i.e. ground fixture), and a detector that determines system settling time. The block *MSD* represents the declaration of the MSD system while the other blocks (*Mass*, *Spring*, *Damper*, *SteadyStateDetector*, *Fixed*, and *MechJunction*) represent the definitions of the system components.

Upon declaring the necessary models, their properties must be identified. Figure 2 depicts the declaration of both the part and value properties. *MSD* is attributed with the *mass*, *spring*, *damper*, *ground*, and *detect* part properties typed to the *Mass*, *Spring*, *Damper*, *Fixed*, and *SteadyStateDetector* block definitions, respectively. While *MSD* has no value properties, most of the block definitions to which its part properties are typed contain value properties. For example, *Mass* contains a value property *m* typed to the value type *SI.Mass*.

4.4 Model Interface

To interact with other models, a given model must have a well-defined interface. Models used in the description of a system's continuous dynamic behavior generally interact using exposed *across* and *through* variables [16]. Since across and through variables are the only means of interaction, they should be encapsulated inside of reusable blocks that are typed to the part properties of another block. These part properties are then exposed to other system components and subsystems. This type of interface is similar to the usage of Modelica connectors.

To illustrate the declaration of a model interface, Figure 2 depicts a block named *MechJunction*. This is a reusable block that encapsulates position and force value properties corresponding to translational across and through variables. To define the interfaces for each component of *MSD*, the appropriate number of part properties are declared for each component and then typed to *MechJunction*. For example, *Mass* has one part property *j* typed to *MechJunction*.

4.5 DAE-Based Internal Behavior

To define a model's DAE-based internal behavior, Modelica relies on equations declared in the equation clause of a given class. Similarly, this is accomplished by placing SysML *constraints* on a given block. A constraint is simply the representation of an equation that constrains a block's value properties. Constraints appear between braces and are displayed in a block's constraints compartment. To model initial conditions, a constraint can be assigned the *«initial»* stereotype. This stereotype is an extension to SysML; it can only be assigned to constraints and implies that the constraint only holds true at the beginning of a simulation.

Usages of constraints and the *«initial»* stereotype are shown in Figure 2. The internal behavior of the block *Mass* is defined using four regular constraints and one initial constraint. Note that the constraints explicitly refer to the Modelica language, but other syntax could be used according to the modeler's preferred executable language.

4.6 Energy and Signal Flow between System Components

To model the flow of energy through a system and its components, a means of interaction must be provided to the interface part properties described in Section 4.3. Generally, the flow of energy in a system is described using the equivalent of Kirchhoff's circuit laws: at a connection, all across variables are equal, while all the through variables add up to zero. While this is modeled implicitly in Modelica using *connect clauses*, our SysML modeling approach explicitly models the interaction with reusable *constraint blocks*. As defined in the SysML specification [4], a constraint block is a specialized form of the SysML block and is intended to package commonly used equations in a reusable, parameterized fashion. Constraint blocks can be identified by the *«constraint»* stereotype that appears in their namespace compartment. To use the definition of a constraint block, another block or constraint block can

declare a *constraint property* and assign the type to a constraint block. Using a SysML *parametric diagram*, the parameters used in the definition of the constraint can be bound to the properties of another block or constraint block using *binding connectors*. A binding connector implies a *pure equality* constraint between two objects. If the objects are part properties, then all of the sub-properties belonging to each part are equal. It is this difference between the semantics of SysML binding connectors and Modelica connections that necessitates the inclusion of an explicit node constraint block in SysML.

Figure 2 shows the definition of a constraint block named *MechNode*. This constraint block has three parameters *j1, j2,* and *j3* of type *MechJunction*. The across and through variables of these parameters are subject to the three packaged constraints that describe Kirchhoff's circuit laws for a translational mechanical system. MSD owns three constraint properties typed to *MechNode* to enable the interaction of its part properties. Figure 3 displays a parametric diagram that depicts the part interactions as a result of binding usages of *MechJunction*.

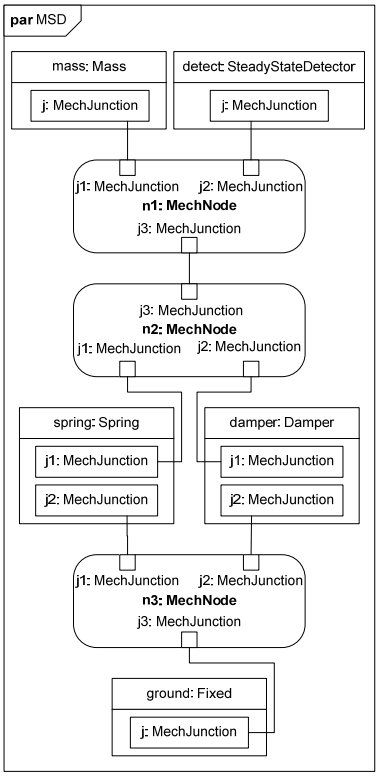

Figure 3. Parametric diagram of the *MSD* model.

5 SysML and Modelica Integration

Currently, system engineering problems are solved using a wide range of domain-specific modeling languages. Moreover, it is unlikely that a single unified modeling language will be able to model in sufficient detail the large number of system aspects addressed by current domain-specific languages. One should not "reinvent the wheel" by creating an all-encompassing systems engineering language capable of modeling and simulating every aspect of a system. On the other hand, managing a large number of models in different languages also poses problems, including communication ambiguity and the preservation of information consistency. To alleviate these problems, a model integration framework is needed for managing the various modeling languages used to solve systems engineering problems.

SysML can provide an answer to this need for model integration. Using SysML, a modeler can abstract a domain-specific language to a level that permits its interaction with other system models. For example, a Modelica model is an excellent way to capture hybrid discrete/DAE-based system behavior, but is not capable of modeling system structure or requirements. Using the modeling approach outlined in Section 4, a modeler can abstract a Modelica model into SysML syntax to represent dependencies and associations with other system models[1].

While SysML is a valuable integration tool, much of that value could be detracted if engineers must manually transform domain-specific models into SysML and vice-versa. In the case of continuous dynamics models, we need an approach for accomplishing automated, bidirectional transformations between the SysML and Modelica languages.

Many methods exist for completing model transformations between two or more modeling languages (metamodels). Two common transformation tools are OMG's Queries/Views/Transformations (QVT) [17] and TGGs [6].

The QVT specification provides a set of languages for querying a source model that complies with a source metamodel and transforming it into a target model that complies with a target metamodel. Two QVT languages, *Relations* and *Core*, are used

[1] Dependencies and associations are UML constructs for expressing types of relationships between information objects.

to declaratively model the relationships between source and target metamodels at different levels of fidelity. The *Operational Mappings* language is then used to perform imperative transformations based on the relationships depicted in the *Core* or *Relations* languages. Overall, QVT is a powerful and widely accepted model transformation tool; however, the imperative nature of the *Operational Mappings* language hampers bidirectional transformations.

TGGs are similar to QVT in intent but are declarative by nature. Accordingly, TGGs are particularly useful for completing complex, bidirectional model transformations. In a TGG, the metamodels for the source and target languages are defined as graphs. The mapping between the two languages is then represented as a set of graph transformation rules applied to a third graph: a *correspondence graph*. For example, a SysML block would be related to a Modelica class using a correspondence entity named *block2class* with one relation pointing to the *block* entity (in the SysML metamodel graph) and one to the *class* entity (in the Modelica metamodel graph). By querying a model space containing SysML or Modelica models, transformations are performed until the model space complies with the specified TGG.

Due to the declarative, bidirectional nature of TGGs, one set of graph transformation rules can be used to transform SysML models into Modelica and vice-versa. Although a TGG is used for this transformation, others have shown that QVT is equally expressive and capable [18]. The TGG and graph transformation rules have been encoded in the Visual Automated Model Transformations (VIATRA) [19] framework. VIATRA enables modelers to create models in a declarative fashion and use pattern recognition to complete graph transformations in a sequential fashion using machines. To demonstrate this TGG, a Java plug-in for Eclipse has been implemented to transform SysML models developed in the Embedded Plus (E+) modeling environment into Modelica models using the OpenModelica [20] compiler (OMC) and Modelica Development Tooling (MDT) plug-in for Eclipse. The functionality of this plug-in is depicted in Figure 4.

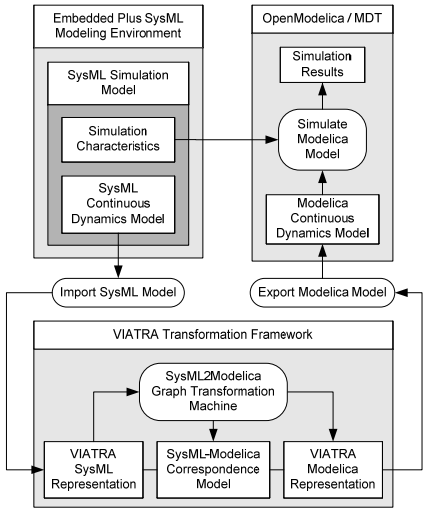

Figure 4. Functionality of the SysML-to-Modelica transformation Eclipse plug-in.

6 Modeling Simulations in SysML

In the context of model-based systems engineering, models and simulations allow systems engineers to investigate and predict the behavior of system alternatives without the need for physical prototyping. For example, a continuous dynamics model of a MSD can be used to simulate and predict the behavior of a car suspension alternative. This section describes how a continuous dynamics model can be related to other relevant design information in SysML: binding of model parameters in a *model context*; defining an experiment performed on a model in a *simulation*; defining a measure of effectiveness as the result of a simulation; and using an *abstracted simulation* in the context of design optimization.

6.1 Defining the Model Context

In systems engineering, a continuous dynamics model is always used in a particular model context. Within this model context the elements of the system structure are bound to the corresponding elements of the analysis model. In current practice, engineers do not always distinguish between the physical structure or system topology and the corresponding system behavior. For instance, it is common practice to use an electric circuit diagram as the representation for defining both the circuit topology as well as the behavior of the circuit in a SPICE simulation. As systems become more complex there often is a need to represent a system by multiple simulation models, corresponding to different levels of abstraction or different disciplinary perspectives. The use of an explicit model context as suggested here facilitates the preservation of consistency amongst all the separate models.

To relate the structure to the behavior, a *model context* block is defined with two part properties: one usage of the system model and one usage of the analysis model. If mathematical relationships be-

yond simple equivalence exist between the known elements of the system model and the corresponding elements of the analysis model, additional constraint blocks can also be defined. Finally, a parametric diagram of the model context block is created to bind the known system elements to the corresponding analysis elements.

In the lower portion of Figure 5, the block *ModelContext* is defined as owning usages of *MSD*, *Car*, and a constraint block named *MassRelation.* In Figure 6, a corresponding parametric diagram is shown establishing a relationship between the MSD and car masses. Inside of this parametric diagram, *msd.mass.m* is defined as one quarter of the mass of *mcCar.mass* by connecting them to the appropriate parameters on the constraint property *massRel*.

6.2 Modeling the Simulation

A simulation is an experiment performed on a computational model [21]. Before a simulation can be performed, the experiment needs to be completely defined: the initial values and boundary values, the outputs to be observed, and potentially the process steps one should go through in the experiment (e.g., time traces of external inputs). From a modeling perspective, all of these aspects can be captured in the model itself or in extensions of the model defined using the same Modelica/SysML constructs described in Section 4. One can therefore assume that the "model" as defined in the model context is fully specified — all the parameters are bound to values and the set of system equations is non-singular. Under those assumptions, the only additional information that needs to be provided is the start and end

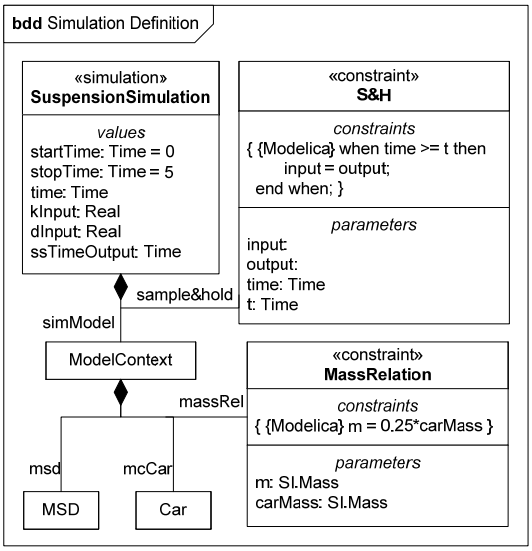

Figure 5. BDD of the *SuspensionSimulation* block.

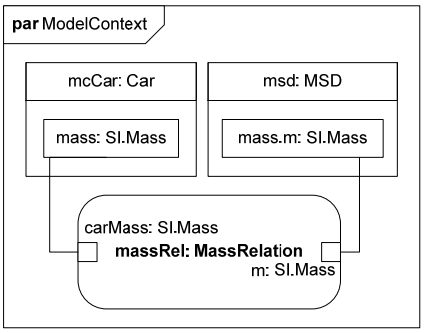

Figure 6. Parametric diagram of the *ModelContext*.

time of the simulation.

To make the semantics of a simulation explicit in SysML, we have defined a *«simulation»* stereotype. As is illustrated in Figure 5, this stereotype requires the inclusion of a *time* property, which represents the simulation time; *startTime* and *stopTime* properties; and a *simModel* block. The meaning of the stereotype is then that all the properties in the *simModel* are evaluated as a function of *time* from *startTime* to *stopTime*. Note that this stereotype completely defines a simulation experiment in a fashion that is independent of any particular simulation solver. In addition, note that Modelica semantics differ from SysML semantics which require the explicit definition of a local simulation time property to which all time-varying system properties can be bound.

6.3 Abstracting the Simulation

A simulation as defined in the previous section allows a systems engineer to define an experiment in which the system behavior can be observed. However in systems engineering, simulations are often used to make decisions. In that case, the same experiment is often performed on multiple variations of the same system — the design or decision alternatives. It then becomes important to abstract this simulation formally by clearly defining the inputs (the properties that can take on different values from one simulation run to the next), and the outputs (the properties that are of interest to the design, for instance, a measure of effectiveness that drives a design optimization). The relationship between inputs and outputs of the simulation can then itself be considered as a model. Unlike the model of the system, this input-output model is an algebraic relationship, albeit a very complex one that requires running the entire simulation to compute the outputs from the inputs. When abstracting (or "wrapping") a simulation in this fashion in support of decision making, it is justifiable to assume that the outputs of the simulation are scalar quantities (decisions can only be made

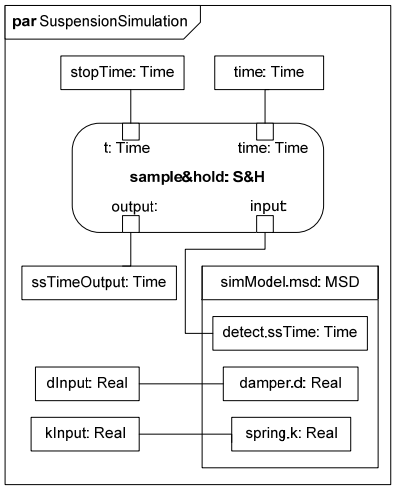

Figure 7. Parametric diagram of *SuspensionSimulation.*

based on scalars because vectors cannot be rank-ordered [22]). Sometimes this requires that one include additional modeling elements in the continuous dynamics model to define these scalar measures of effectiveness. For instance, in the BDD in Figure 5 and the corresponding parametric diagram in Figure 7, the suspension simulation has been abstracted into an input-output model with inputs as the decision variables, *dInput* and *kInput* (bound to the damping and stiffness of the suspension), and an output as the measure of effectiveness, *ssTimeOutput* (the steady-state time of the mass-spring-damper system). The output has been bound to a model property through a sample and hold constraint property, *sample&hold*, making explicit that the output takes on the value of the time-varying property *detect.ssTime* when the simulation time equals *stopTime*. In general, more complex models may be necessary to relate scalar outputs to time-varying simulation properties.

6.4 Embedding a Simulation into an Analysis

Once a simulation has been abstracted into an input-output model, it can be used in support of analyzing system alternatives with respect to stakeholder requirements and measures of effectiveness, as is illustrated in Figures 8 and 9. Analyses generally verify that a system alternative meets a certain system requirement, which can be modeled explicitly using the *«verify»* dependency. A parametric diagram of that block can be used to connect the system alternative to the simulation, as is illustrated in Figure 9. Instead of binding the simulation inputs and outputs directly to the corresponding value properties of the system alternative, one could also define an optimization problem in which the stiffness and damping are optimized with respect to one or more

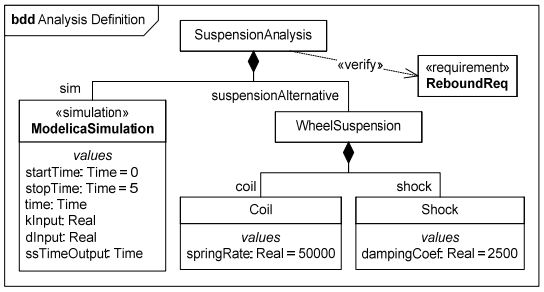

Figure 8. BDD of the *SuspensionAnalysis* block.

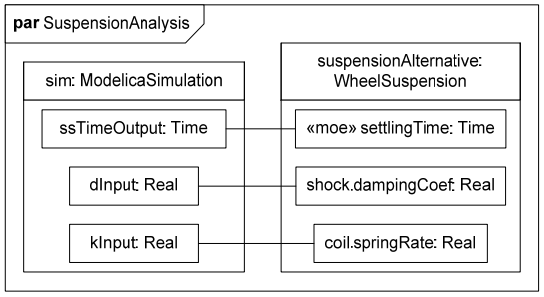

Figure 9. Parametric diagram of *SuspensionAnalysis*.

measures of effectiveness. Whenever there is a need for repeated evaluation of the simulation with different inputs, it is desirable to embed the simulation explicitly in an analysis context as is shown in Figure 8.

7 Discussion and Closure

In this paper, we have introduced an approach for combining SysML and Modelica in a synergistic fashion. No single language or formalism can possibly capture all of the knowledge and information needed to solve systems engineering problems. While Modelica is well-suited for describing the dynamic behavior of complex systems, it offers no support for relating that behavior to stakeholder requirements. Similarly, SysML allows one to define the high-level relationships between requirements and functional, physical and operational architectures of a system, but lacks the detailed semantics to capture for instance geometry. It is therefore crucial that capabilities are developed for relating in a formal framework the different knowledge representations commonly employed in systems engineering problems. SysML provides the foundation for making a first step in that direction. The general-purpose and adaptable nature of the language enables system engineers to interrelate their preferred knowledge representations. In addition, formal metalevel mappings as described by TGGs provide a promising founda-

tion for bidirectional mappings between the different knowledge representations.

Using the modeling approaches described in this paper, engineers will be more capable of managing system complexity through the modeling of dependencies between continuous dynamic system behavior and other system aspects. Additionally, the mapping of SysML to Modelica and the resulting transformation abilities enable engineers to describe their systems at a higher level of abstraction while still maintaining the benefits of executable knowledge representations.

In this paper, the intent has been to take advantage of SysML's adaptability and to make a step towards the unification of various modeling formalisms. While the continuous dynamics modeling approach described in this paper builds on the Modelica language, it still maintains a certain language independence thanks to the general, declarative nature of Modelica. TGGs could be developed to map SysML to the syntax of other languages, with the restriction that when mapping to a causal, procedural modeling language, a compiler must be used to assign causalities and sort the equations.

The ongoing efforts towards the unification of engineering knowledge representations in SysML are exciting steps for the systems engineering community. Utilizing and increasing the abilities of SysML promises to improve the current state of systems engineering and bring to fruition the benefits of MBSE.

Acknowledgements

This work has been funded by Deere & Company. Additional support was provided by the ERC for Compact and Efficient Fluid Power, supported by the National Science Foundation under Grant No. EEC-0540834. The authors would also like to thank Sandford Friedenthal, Leon McGinnis and Russell Peak for the discussions that helped crystallize the ideas presented in this paper.

References

[1] Sage, A. P., and Armstrong Jr., J. E., 2000, *Introduction to Systems Engineering*, John Wiley & Sons, Inc., New York, NY.

[2] Fisher, J., 1998, "Model-Based Systems Engineering: A New Paradigm," *INCOSE Insight*, 1(3)

[3] Gero, J. S., 1990, "Design Prototypes: A Knowledge Representation Schema for Design," *AI Magazine*, 11(4), pp. 26-36.

[4] Object Management Group, 2007, "OMG Systems Modeling Language Specification," http://www.omg.org/cgi-bin/doc?ptc/07-09-01.

[5] Booch, G., Jacobson, I., and Rumbaugh, J., 2005, *The Unified Modeling Language User Guide*, Addison-Wesley Professional.

[6] Schürr, A., 1994, "Specification of Graph Translators with Triple Graph Grammars," in *WG'94 Workshop on Graph-Theoretic Concepts in Computer Science*.

[7] Peak, R. S., and Wilson, M. W., 2001, "Enhancing Engineering Design and Analysis Interoperability Part 2: A High Diversity Example," *First MIT Conference Computational Fluid and Structural Mechanics (CFSM)*, Cambridge, Massachusetts, USA.

[8] Peak, R. S., Burkhart, R. M., Friedenthal, S. A., Wilson, M. W., Bajaj, M., and Kim, I., 2007, "Simulation-Based Design Using SysML-Part1: A Parametrics Primer," in *INCOSE Intl. Symposium*, San Diego, CA.

[9] Peak, R., Friedenthal, S., Moore, A., Burkhart, R., Waterbury, S., Bajaj, M., and Kim, I., 2005, "Experiences Using SysML Parametrics to Represent Constrained Object-Based Analysis Templates," *7th NASA-ESA Workshop on Product Data Exchange (PDE)*, Atlanta, GA, USA.

[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 ModelicaModels," in *International Workshop on Equation-Based Object-Oriented Languages and Tools*, Linköping University Electronic Press, Berlin, Germany.

[12] Johnson, T. A., Paredis, C. J. J., Burkhart, R. and Jobe, J. M., 2007, "Modeling Continuous System Dynamics in SysML," in *2007 ASME International Mechanical Engineering Congress and Exposition*, ASME, Seattle, WA.

[13] Keeney, R. L., 1994, "Creativity in Decision Making with Value-Focused Thinking," *Sloan Management Review*, 35(4), pp. 33-41.

[14] Christen, E., and Bakalar, K., 1999, "VHDL-AMS - A Hardware Description Language for Analog and Mixed-Signal Applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, **40**(10), pp. 1263-1272.

[15] Mitchell, E. E. L., and Gauthier, J. S., 1976, "Advanced Continuous Simulation Language (ACSL)," *SIMULATION*, **26**(3), pp. 72-78.

[16] Paynter, H., 1961, *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, MA.

[17] Object Management Group, 2007, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification," http://www.omg.org/docs/ptc/07-07-07.pdf.

[18] Greenyer, J., Kindler, E., 2007, "Reconciling TGGs with QVT," in *Model Driven Engineering Languages and Systems, MoDELS 2007*, Springer, Berlin / Heidelberg.

[19] Varró, D., 2003, *VIATRA: Visual Automated Model Transformation*, Thesis, Department of Measurement and Information Systems, University of Technology and Economics, Budapest.

[20] Fritzson, P., et al., , 2007, "OpenModelica System Documentation," http://www.ida.liu.se/labs/pelab/modelica/OpenModelica/releases/1.4.3/doc/OpenModelicaSystem.pdf.

[21] Fritzson, P., 2004, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, Piscataway, NJ.

[22] Keeney, R. L., and Raiffa, H., 1976, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, Jon Wiley and Sons, New York.