

An Overview of the SysML-Modelica Transformation Specification

Christiaan J.J. Paredis¹, Yves Bernard², Roger M Burkhart³, Hans-Peter de Koning⁴, Sanford Friedenthal⁵, Peter Fritzson⁶, Nicolas F Rouquette⁷, Wladimir Schamai⁸

¹Georgia Institute of Technology, ²Airbus, ³Deere & Co., ⁴ESA/ESTEC, ⁵Lockheed Martin Corp., ⁶Linköping University, ⁷Jet Propulsion Laboratory, ⁸EADS Innovation Works.

Contact info: chris.paredis@me.gatech.edu

Copyright © 2010 by Christiaan J.J. Paredis. Published and used by INCOSE with permission.

Abstract.

This paper provides an overview of the formal transformation between the two complementary languages: OMG SysML™ and Modelica. SysML is a standardized general purpose graphical modeling language for capturing complex system descriptions in terms of their structure, behavior, properties, and requirements. Modelica is a standardized general purpose systems modeling language for analyzing the continuous and discrete time dynamics of complex systems in terms of differential algebraic equations. Integrating the descriptive power of SysML models with the analytic and computational power of Modelica models provides a capability that is significantly greater than provided by SysML or Modelica individually. A standardized bi-directional transformation between the two modeling languages is being developed that will support implementations to transfer efficiently and automatically the modeling information between SysML and Modelica models without ambiguity. In addition to an overview of this bi-directional transformation approach, the paper provides a simple example to clarify the transformation principles and to illustrate the important synergies resulting from the integration between these two languages.

Introduction

The objective of the SysML-Modelica Transformation Specification (OMG SE DSIG SysML-Modelica Working Group, 2009) is to provide a bi-directional mapping between OMG SysML™ (Object Management Group, 2008) and Modelica (Modelica Association, 2009) and to leverage the benefits of both languages. By integrating SysML and Modelica, SysML's strength in descriptive modeling can be combined with Modelica's formal executable modeling capability to support analyses and trade studies.

SysML is a general-purpose systems modeling language that can be used to create and manage models of systems using well-defined, graphical constructs with underlying semantics (Object Management Group, 2008). SysML reuses a subset of UML 2 (Object Management Group, 2009) constructs and extends them by adding new modeling elements and two new diagram types. These SysML diagrams are shown in Figure 1. The set of behavioral and structural diagrams combined with the requirements diagram and parametric diagram provide an integrated view of a system. But SysML represents much more than just a set of diagrams. Underlying the diagrams, there is an

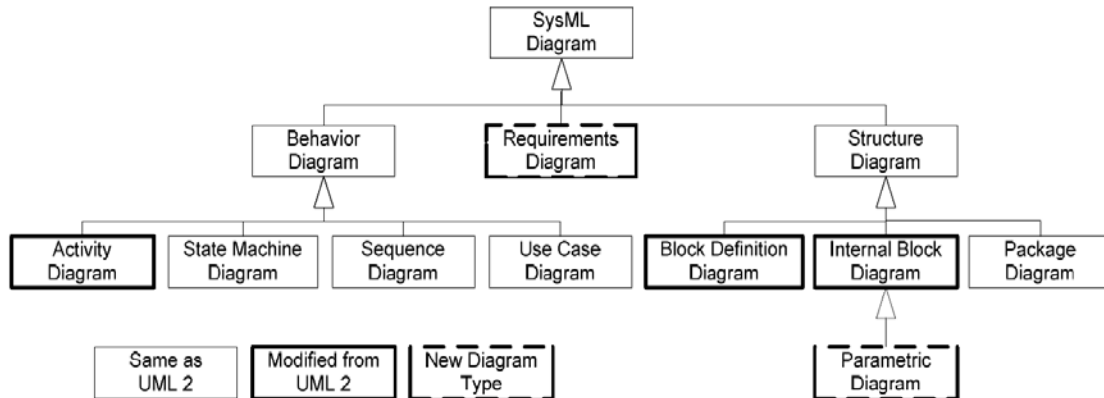


Figure 1: An overview of the SysML diagrams and their relation to UML diagrams.

abstract syntax model repository that formally represents all the modeling constructs. The graphical model provides a mechanism to organize, enter, retrieve, and view the system-descriptive data contained in the model repository. The diagrams provide multiple views of the same system model; these multiple views can be maintained consistently due to the semantic underpinning of the modeling language. In the context of SysML, the structure view primarily refers to the hierarchy and interconnections among the parts of the system, and the interconnections between the system and its external systems. The behavior view describes the sequence of events and activities that the system must execute. The requirements diagram captures text requirements in the model, and enables them to be linked to other parts of the model, to provide unambiguous traceability between the requirements and system design. Parametrics provide the bridge between the system descriptive model in SysML and other simulation and engineering analysis models. While structure and behavior are heavily based on UML, both requirements and parametrics are unique to SysML. Through these extensions, SysML is capable of representing the specification, analysis, design, verification and validation of systems.

As indicated above, the system behavior in SysML is captured through a combination of activity diagrams, state machine diagrams, and/or sequence diagrams and the associated semantics. The Foundational Subset of the UML specification (Object Management Group, 2008) provides the additional semantics to enable SysML activity diagrams to be executed in a standard way. In addition, SysML includes parametric diagrams to capture models of constraint-based behavior, such as continuous-time dynamics in terms of energy flow. However, the syntax and semantics of such behavioral descriptions in parametrics have been left open to integrate with other simulation and analysis modeling capabilities to support the execution of these models. Additional information on SysML can be found at <http://www.omg.sysml.org>.

Modelica is an object-oriented language for describing differential algebraic equation (DAE) systems combined with discrete events (Fritzson, 2004). Such models are ideally suited for representing the flow of energy, materials, signals, or other continuous interactions between system components. It is similar in structure to SysML in the sense that Modelica models consist of compositions of sub-models connected by ports that represent energy flow (undirected) or signal flow (directed). The models are acausal, equation-based, and declarative. The Modelica Language is defined and maintained by the Modelica Association (www.modelica.org), which publishes a formal specification (Modelica Association, 2009) but also provides an extensive Modelica Standard Library, which includes a broad foundation of essential models covering

domains ranging from (analog and digital) electrical systems, mechanical motion and thermal systems, to block diagrams for control (Modelica Association, 2009). Finally, it is worth noting that there are several efforts within the Modelica community to develop open-source solvers, such as in the OpenModelica Project (The Open Source Modelica Consortium, 2009).

In conclusion, SysML and Modelica are two complementary languages supported by two active communities. By integrating SysML and Modelica, we combine the very expressive, formal language for differential algebraic equations and discrete events of Modelica with the very expressive SysML constructs for requirements, structural decomposition, logical behavior and corresponding cross-cutting constructs. In addition, the two communities are expected to benefit from the exchange of multi-domain model libraries and the potential for improved and expanded commercial and open-source tool support.

Related work

Pop *et al.* (Pop, et al., 2007) have worked on issues regarding the integration of UML and Modelica. They have created a UML profile called ModelicaML that enables users to depict a Modelica simulation model graphically. The ModelicaML profile reuses several UML and SysML constructs while adding several completely new language constructs. Such constructs are the Modelica class diagram, the equation diagram, and the simulation diagram. The intent of the ModelicaML profile is slightly broader than the SysML-Modelica Transformation described in this paper; ModelicaML also aims to represent all of the Modelica language in a graphical form, including equations and algorithms. The work on ModelicaML is still ongoing and has recently extended by Schamai *et al.* (Schamai, et al., 2009). In parallel to the development of ModelicaML, Johnson *et al.* (Johnson, et al., 2008) proposed a mapping between SysML and Modelica. They built directly on the SysML language and introduced stereotypes only as necessary to capture the Modelica semantics.

The SysML-Modelica Transformation Specification described in this paper grew out of the joining of these two previous efforts. In December 2008, a formal working group was established within the Systems Engineering Domain Specific Interest Group (SE DSIG) of the Object Management Group (OMG). The working group is moving forward towards the adoption of the Transformation Specification as a formal specification within OMG. More information about the most recent progress of the working group can be found at (OMG SE DSIG SysML-Modelica WG, 2009).

Integration Approach

To develop a transformation between the SysML and Modelica languages, a formal, systematic approach is used. As is illustrated in Figure 2, the transformation approach is to specify first an extension to SysML called the SysML4Modelica profile which represents the most common Modelica language constructs. This allows the Modelica concepts to be expressed in an extension of SysML that supports round-trip transformation from SysML to Modelica and back. The profile extends the UML4SysML subset of UML and the SysML extensions to that subset that are required to capture the relevant Modelica concepts and enable the mapping between the two languages.

The SysML-Modelica Transformation is then specified between the profile constructs and the Modelica language constructs as captured in the Modelica meta-model. Introducing the profile into the transformation approach is intended to simplify the transformation to Modelica, and

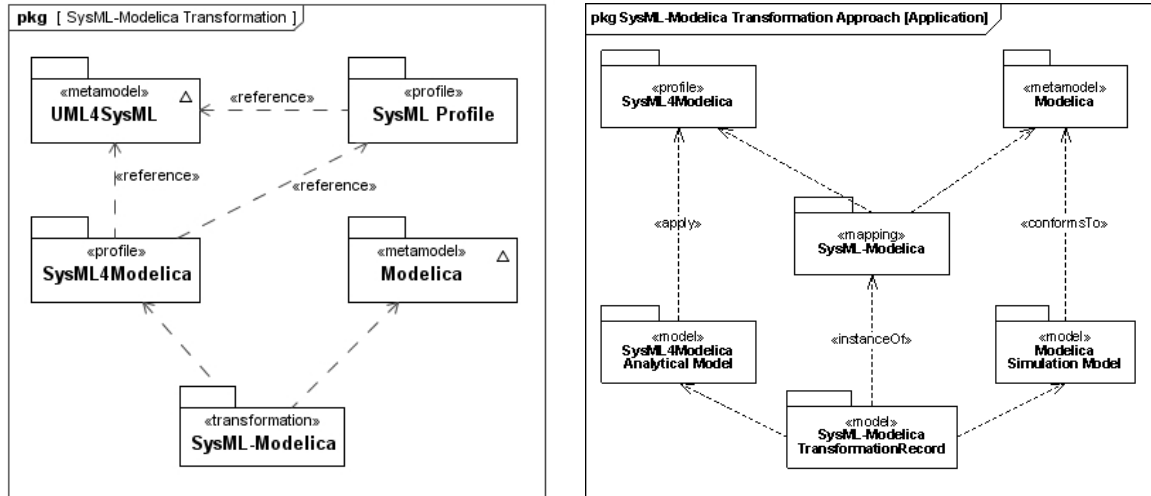


Figure 2: The SysML-Modelica Transformation in relation to SysML and Modelica.

facilitate model reuse by more directly leveraging existing model libraries within Modelica. In this way, the user first creates the system model in a SysML modeling tool as they would normally do. The user then selects the part of the model to be analyzed by Modelica (e.g., a particular subsystem) and applies the SysML4Modelica profile to create an analytic representation of that part of the model. The SysML modeling tool is expected to include this profile. The analytic representation expressed in the SysML4Modelica profile is then transformed to a Modelica model, where it can be executed by a Modelica solver.

Not all SysML modeling constructs are expressed in the SysML4Modelica profile, nor will all Modelica constructs be represented in the profile. In the definition of the Transformation, the focus has been on including all the Modelica language features that are most common and, together, cover the majority of the Modelica models in the Modelica Standard Library (Modelica Association, 2009). Some Modelica concepts are not required for the mapping, such as graphical annotations and certain concepts that are associated with pre-compilation. Changes to SysML and Modelica may be recommended as a result of the Transformation Specification effort, but these changes are subject to the adoption process for the respective specifications. Future changes could also include the introduction of additional SysML constructs into the Modelica Language or additional Modelica constructs in the SysML language; however, this is outside the scope of the current SysML-Modelica Transformation effort.

The SysML-Modelica Transformation leverages the fundamental concepts of the Model-Driven Architecture (MDA) (Object Management Group, 2009). Different transformation implementations can be applied to implement this specification such as the QVT and others (Object Management Group, 2008). The transformation can leverage an XMI formatted static file transfer (Object Management Group, 2007) or other mechanisms such as API's that support a dynamic interchange capability.

To develop the SysML4Modelica profile in a systematic fashion, we start from the Modelica Language Specification and identify for each Modelica language construct an equivalent construct in SysML. If an equivalent construct does not exist, stereotypes are created to extend the SysML language. The following naming convention is used to define each Modelica construct in the SysML4Modelica profile: `«modelicaConstruct»`, where *Construct* is the name of Modelica

language construct as defined in the Modelica abstract syntax definition (Modelica Association, 2009).

Even when an equivalent SysML construct exists, it is sometimes necessary to introduce a stereotype in order to distinguish the Modelica construct from the ordinary SysML construct when supporting round-trip transformation. In addition, the concrete syntax of Modelica often provides alternative representations to express the exact same semantics. In such cases, the intent is to avoid duplicating this redundancy in SysML4Modelica without loss of expressivity. For mapping purposes, one of the redundant representations is identified as the primary (most explicit) representation, and SysML4Modelica constructs are preferably mapped onto this primary representation. It should also be noted, that Modelica includes a graphical syntax using iconic representations of block diagrams that maps to its textual syntax. An example of the Modelica graphical syntax is shown in Figure 3 for a set of components connected together via Modelica connectors and connections.

Initially, the SysML-Modelica Transformation Specification provides a textual description of the mapping between Modelica and SysML4Modelica. However, it is the intent also to describe this mapping formally by defining a Triple Graph Grammar (Königs, 2005), linking the Modelica and SysML meta-models. Such a formal definition of the mapping has the additional advantage that meta-CASE tools (such as MOFLON (Weisemöller, et al., 2009)) can be used to generate executable transformations between SysML and Modelica modeling tools (assuming they support some standardized interface such as JMI (Java Community Process, 2002)). An additional implementation of the mapping is being developed as part of the OpenModelica project (The Open Source Modelica Consortium, 2009).

Semantic Comparison between SysML and Modelica

Before focusing on the detailed modeling constructs, a high-level decision needs to be made regarding the choice of SysML elements to represent Modelica modeling constructs. Although Modelica is a textual language, it also supports a graphical view through its annotation mechanism. This graphical view illustrates clearly the strong similarity that exists between SysML and Modelica. Both languages support the decomposition of systems (or behavioral models of systems) into subsystems or components and the interactions between them. For instance, the Modelica model of a motor controller (shown in Figure 3) contains subcomponents (such as motor, gearbox, and controller). The interactions between them are illustrated by edges

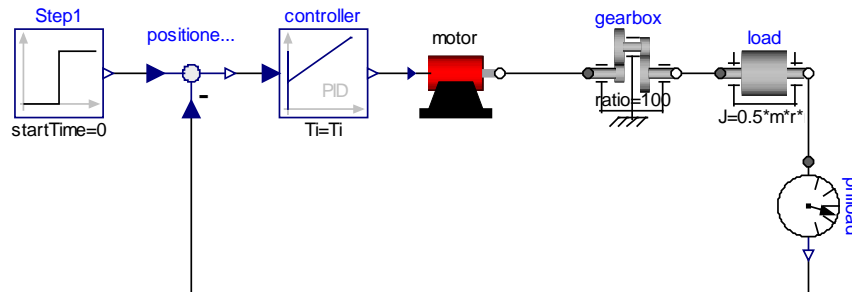


Figure 3: A Modelica model of a motor controller consisting of component models and the connections between them. The connections include both causal signal connections (e.g., in and out of the controller) and acausal energy connections (e.g., the rotational mechanical energy connections of the gearbox).

Table 1: A comparison between Modelica and SysML elements, semantics, and diagrams.

<i>Diagram</i>	Modelica	Internal Block	Parametric	Activity
Model Definition	Model	Block	Constraint Block	Activity
Model Usage	Component	Part Property	Constraint Property	Action
Port Definition	Connector	Block, Value Type, Flow Spec.	Value Type	Block, ValueType
Port Usage	Component	Flow Port	Parameter	Object Node
Edge	Connection	Connector	Binding Connector	Object Flow

connecting the interface locations (called connectors in Modelica) of the components. Such hierarchical compositions of Modelica models and the connections between them constitute the primary modeling approach in Modelica. Before considering the details of the language, it is thus important to consider carefully how these primary modeling constructs map to SysML.

As illustrated in Table 1, in SysML there are three types of diagrams that have a structure that is similar to the hierarchical, connector-based composition of Modelica models: the Internal Block Diagram (IBD), the Parametric Diagram, and the Activity Diagram. All three diagrams support some sort of “ports”, some sort of connection of “port-based” objects through “port-connections,” and hierarchical encapsulation through “port-delegation.” In the follow sub-sections, we use these three diagrams to discuss the main question: What are the SysML elements that match the Modelica semantics best?

Modelica. In Modelica, ports are called connectors and the edges between ports are called connections (Modelica Association, 2009). The ports (connectors) can include four types of quantities: inputs, outputs, flows and non-flows. Inputs and output are used when the direction of the flow is known and fixed, as for instance in signals flowing in a control system. Flow and non-flow quantities are used to describe energy or material flow (they are also sometimes referred to as through and across variables, respectively). When connecting two Modelica connectors with a connection, the semantics for inputs and outputs are causal binding: the input is assigned the value of the output to which it is connected. Input and output connectors must therefore be used in conjugate pairs, and only one output can be connected to each input. For flow and non-flow variables, the connection semantics correspond to Kirchhoff’s Laws, namely, the value of the flow variables add up to zero and the values of the non-flow variables are set equal (in an equation-based, acausal fashion). When more than one connection is made to a connector containing a flow variable, then an ideal, loss-less energy or material exchange is assumed by imposing that the values of flow variables of all connected connectors add up to zero. To impose the correct modeling of energy exchange, Modelica requires that the number of flow and non-flow quantities of a connector be equal.

In addition to connectors, Modelica models can contain variables and sub-models (i.e., model usage in Table 1). Although Modelica does not explicitly distinguish between these three categories of “components” (i.e., connectors, variables, sub-models), it may still be useful and desirable to distinguish explicitly among them when mapping to SysML.

SysML Internal Block Diagrams. The primary purpose of Internal Block Diagrams (IBD) in conjunction with Block Definition Diagrams (BDD), is to express system structural

decomposition and interconnection of its parts. The SysML concepts used in the IBD/BDD viewpoints have quite flexible semantics and may be used to establish logical and conceptual decompositions, for instance, as in a context diagram. The Blocks in SysML are similar to Classes in Modelica (specifically the specialized class types of Model, Block, Connector, etc.). Blocks can be decomposed in the same way Modelica Classes can be decomposed.

The “ports” in IBDs are called Ports and the connections between ports are called Connectors. There are two kinds of ports: Flow Ports and Standard Ports. The Standard Ports are particularly geared towards service-based interactions by representing the interfaces (e.g., software methods) that are provided or required by a particular block. Such service-based interactions are not appropriate for modeling the connections found in Modelica. Flow Ports on the other hand do provide semantics that reflect Modelica connectors more closely. A Flow Port describes an interaction point through which input and/or output of items such as data, material, or energy may flow in and out of a block. For Modelica-type interactions, the “items” could be either signals (for input and output quantities) or energy/material (for flow and non-flow quantities). In Modelica these interactions are modeled as instances of Modelica Connector types. Such instances do not have a direction of flow associated with them directly, but should be interpreted as containing either inputs, outputs, or energy/material flows based on the definition of the Connector type of which they are an instance. This is similar to SysML nonatomic FlowPorts typed by FlowSpecifications, although one may argue that the combination of a flow and non-flow variable in a Modelica energy/material connector constitute one concept (i.e., one energy or material flow) and should therefore be modeled as an atomic rather than non-atomic flow port. In addition, the (acausal) connection between flow ports in SysML does not explicitly carry the Kirchhoff semantics as for energy/material connections in Modelica.

An additional subtle difference in semantics lies in the fact that, in SysML, the type of a flow property defined in a flow port specifies what *can* flow through that port; what actually flows must be defined by associating an Item Flow to a SysML Connector (the connection between the flow ports). In Modelica, no such differentiation between what *can* flow and what *actually* flows is made. This makes sense because Modelica describes the behavior of what actually happens (what flows) rather than a specification of an interface (what can flow).

In conclusion, although IBDs seem to have very similar constructs to Modelica, there are some subtle differences in semantics so that new stereotypes will have to be introduced to adequately capture the Modelica semantics of Connectors and Connections.

SysML Parametric Diagrams. The purpose of Parametric Diagrams is to express mathematical relationships between parameters. In Parametric Diagrams, the “ports” are Constraint Parameters and the “connections” are Binding Connectors. Inside a Constraint Block, mathematical relationships are defined constraining its Constraint Parameters. A Constraint Property is a usage of a Constraint Block. Its Constraint Parameters are then bound to other Constraint Parameters or to Properties of Blocks. The semantics of a Binding Connector indicate a mathematical equality between the (Block) Properties or Constraint Parameters being connected. This mathematical equality is an acausal relationship.

Although the Binding Connectors share the acausal nature of energy-connections in Modelica, they are currently missing the notions of a Modelica Flow variable and of causal inputs and outputs (Note: an issue has been submitted requesting the addition of causality specifications in parametrics to future versions of SysML). The equivalent of a Binding Connector does not

actually exist in Modelica, but can be captured in a non-graphical fashion by introducing an equality equation between the two variables that are bound. Therefore, in order to capture the semantics of a Modelica connection, one would have to introduce a new SysML connector element that is equivalent to a Modelica Connector, and that reflects the semantics of Kirchhoff's laws. Another possibility would be to make the equations for Kirchhoff's laws, which are implicit in Modelica connections, explicit as another SysML Constraint Property. This option is appealing because it makes the semantics very explicit, but has the disadvantage that it makes the models more cumbersome to create and more difficult to read.

Finally, unlike Blocks, Constraint Blocks do not have Value Properties that are not Constraint Parameters. As a result, (local) variables in Modelica would have to be represented as Constraint Parameters, making it difficult to distinguish them from "ports."

In conclusion, the intent of Parametric Diagrams is similar to the intent of Modelica Models, and they therefore deserve consideration. However, the types of connections that exist in Modelica do not exist in Parametric Diagrams and vice versa. As a result, the use of Parametric Diagrams will require the introduction of additional constructs (stereotypes).

SysML Activity Diagrams. The purpose of an Activity graph in SysML is to specify the transformation of inputs to outputs through a controlled sequence of actions. An Activity decomposes into Actions. In activity graphs, the Object Nodes (i.e., Pins and Parameter Nodes) correspond to buffers to place input and output tokens. The connections between Object Nodes correspond to Object Flows. These flows typically represent the transfer of one or more objects at a discrete moment in time, although it is possible to specify a streaming flow that could be continuous, i.e., the time between arrival of tokens (or "objects") is zero. It is this latter case that needs to be described in terms of differential equations and is also closest to the semantics of Modelica's flows. However, the strict notion of flows from output to inputs in Activity graphs is not imposed in Modelica (Note: this flow direction would correspond to a constraint on the sign of a flow variable, but has nothing to do with mathematical causality).

In conclusion, only the special case of continuously streaming object flows seems to match the Modelica semantics of energy flow, and even for that case, the semantics are quite different. Activity graphs therefore are the least appropriate for a mapping from Modelica Class, although they will be explored when mapping the Modelica Functions and Algorithms to SysML4Modelica.

Selected Diagram: SysML Internal Block Diagram with Embedded Constraints. It is clear from the discussion in the previous sections that there is not a single viewpoint that embeds the Modelica semantics perfectly. As a result, the use of more than one SysML viewpoint with multiple stereotypes needs to be defined to extend the SysML semantics.

Blocks, ConstraintBlocks, FlowPorts, classical Connectors and BindingConnectors can be used to map Modelica Models, Components, Connectors, and Connections to SysML, assuming an extension of connector is defined to support the Kirchhoff semantics. This could be expected since Constraint Blocks are restricted versions of regular Blocks. Actually, Constraint Blocks and Parametric Diagrams are too restricted. For instance, Constraint Blocks cannot have value properties (only constraint parameters), and the only connectors allowed in a Parametric Diagram are binding connectors, which have semantics of equality constraints and can thus not be further restricted to represent Kirchhoff's laws as is needed for Modelica. Parametric Diagrams could be useful to capture the semantics of Modelica if one wants to make explicit the equations that are implicit in Modelica connections. This is illustrated in the next section.

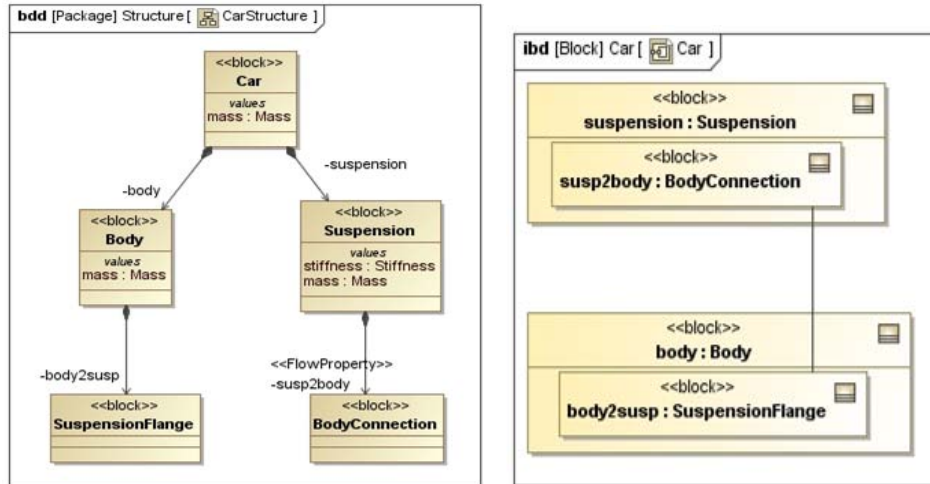


Figure 4: Descriptive model of a car suspension visualized as a BDD and IBD.

Illustrative Example

Consider the design of a car suspension. As illustrated in Figure 4, the suspension can be described in the context of a car using a descriptive SysML model, expressed in a BDD and corresponding IBD.

Assume now that one needs to evaluate the dynamic response of the suspension by simulating the car body's position as a function of time. A possible continuous dynamics model for such a simulation models the suspension as a linear spring and the car body as a point mass. This model is illustrated in Figure 5 in both Modelica and in the SysML4Modelica profile which represents the corresponding Modelica constructs. By stereotyping SysML ports and connectors, the semantics of Kirchhoff's laws have been introduced into SysML.

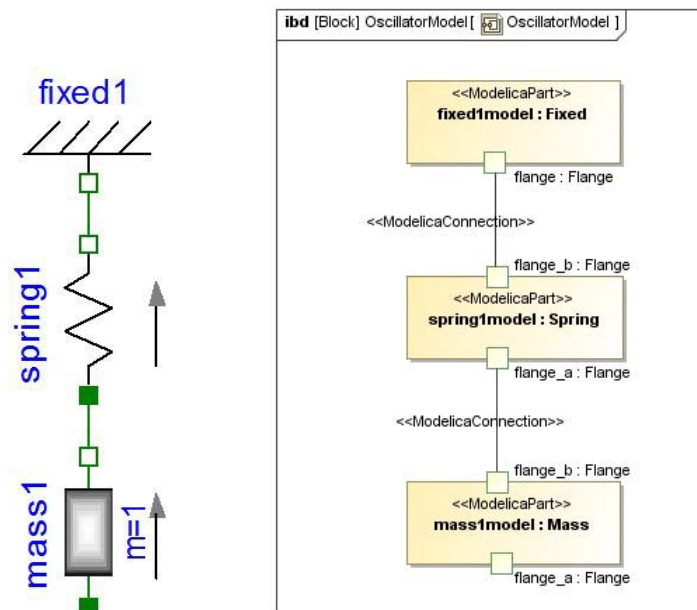


Figure 5: A Mass-Spring model for a car suspension, in Modelica (left) and SysML4Modelica (right).

The SysML parts are stereotyped as «modelicaPart» (i.e., mass1model, spring1model, fixed1model), that correspond to usages of models from the Modelica Standard Library. For instance, as illustrated in Figure 6, the library Modelica.Mechanics.Translational.Components includes definitions of continuous dynamics models for a Spring and a Mass. Note that one could apply stereotypes in SysML that include icons equivalent to the elements from the Modelica library so that the SysML4Modelica representation in Figure 5 on the right could be almost identical to the Modelica representation on the left.

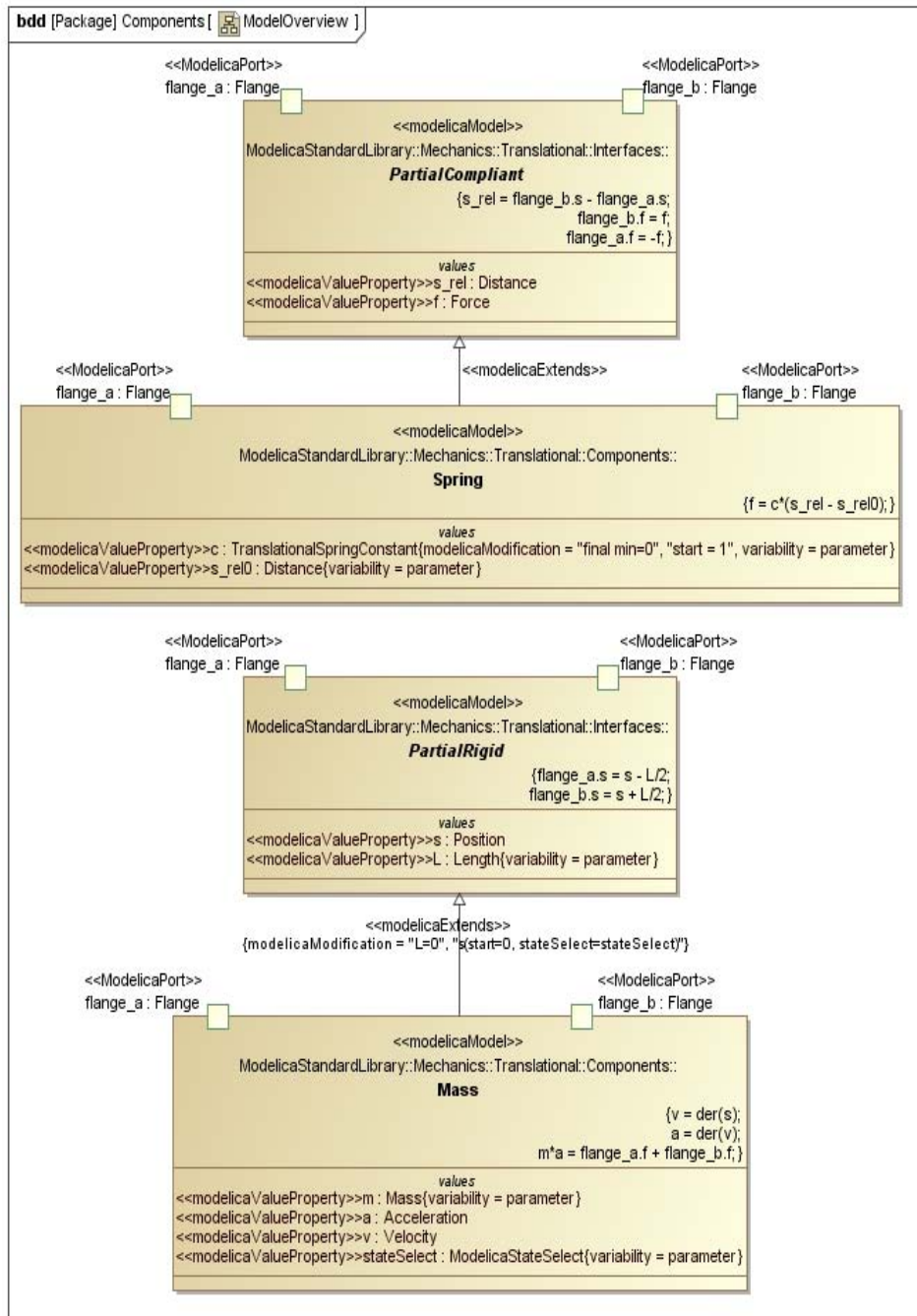


Figure 6: Continuous dynamics models for Mass and Spring defined in the Modelica Standard Library.

In Figure 6, the usages of these models, stereotyped as «modelicaPart» are connected to each other at their «modelicaPort» by a «modelicaConnection». These connections carry the semantics of Kirchhoff's Laws (in this example—or, more precisely, the same semantics as an equivalent Modelica connection). These semantics can be made more explicit by using a Parametric Diagram as is shown in Figure 7. But, as one can see by comparing Figure 7 and Figure 5, this comes at a cost of a much larger and less readable diagram. Similarly, one could have represented the internal equations of the Mass model in a Parametric Diagram, as is illustrated in Figure 8, but again, the more explicit semantics come at a cost of increased complexity. For this reason, only Blocks and Internal Block Diagrams are used in the SysML4Modelica profile. The parametrics still provide the underlying semantics for capturing the detailed equations, but this complexity can often be abstracted and made invisible to the modeler.

Finally, it is worth illustrating how the SysML4Modelica continuous dynamics model in Figure 5 relates to the SysML descriptive model in Figure 4. Since both the descriptive and the continuous dynamics models are views of the same system, they cannot be independent of each other. Changes to the descriptive model are likely to require corresponding changes to the continuous dynamics model and vice versa. Such dependencies can be modeled in an analysis context — the context in which the analysis model (i.e., the continuous dynamic analysis in this case) is defined.

The analysis context is illustrated in Figure 9. It establishes the dependencies between the descriptive model components and their corresponding analysis models. In addition, the detailed bindings between the descriptive and analysis properties are defined in the Parametric Diagram illustrated in Figure 10.

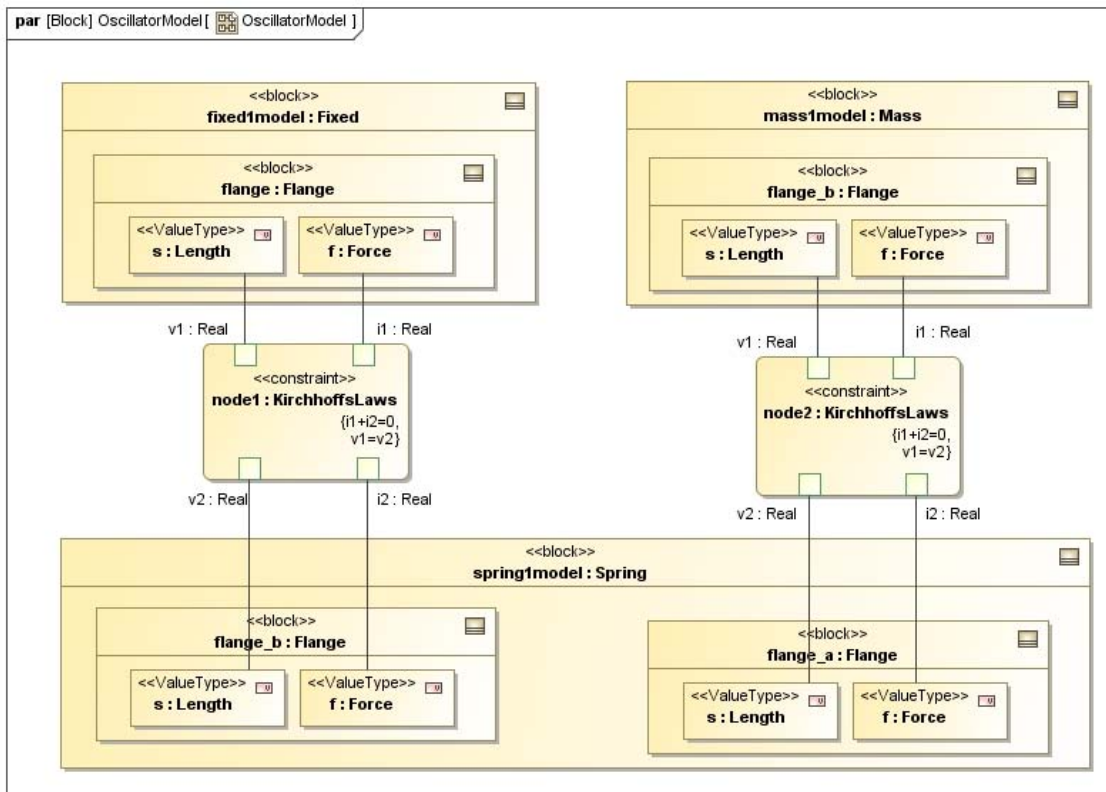


Figure 7: Mass-Spring model as represented in a Parametric Diagram.

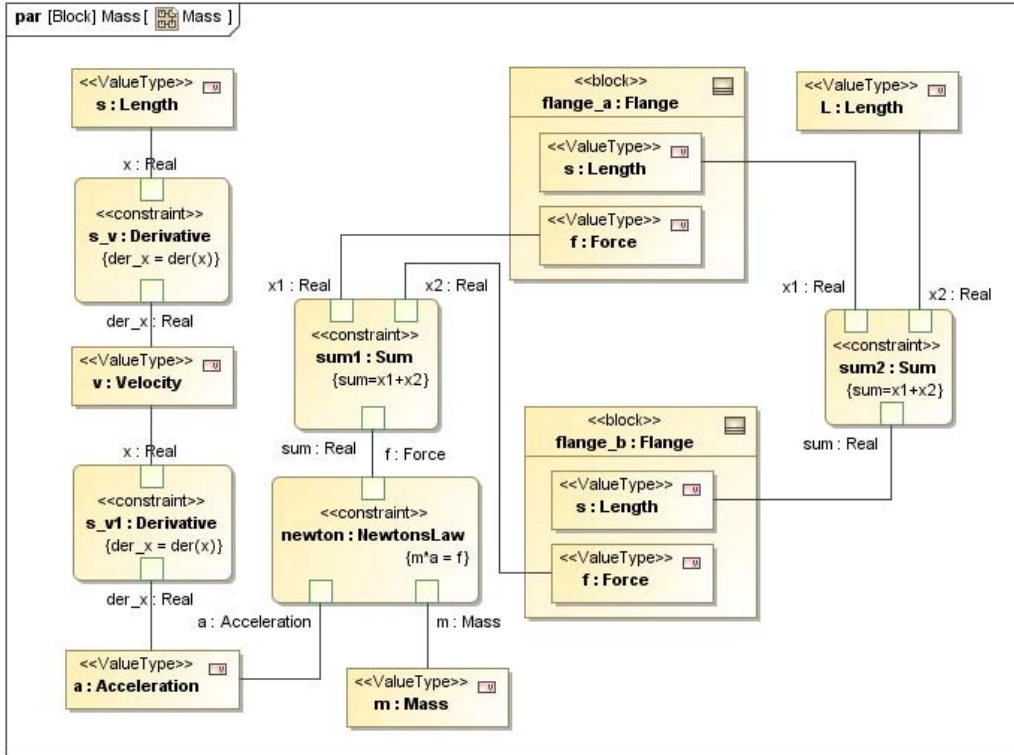


Figure 8: Mass model as it could be represented in a Parametric Diagram.

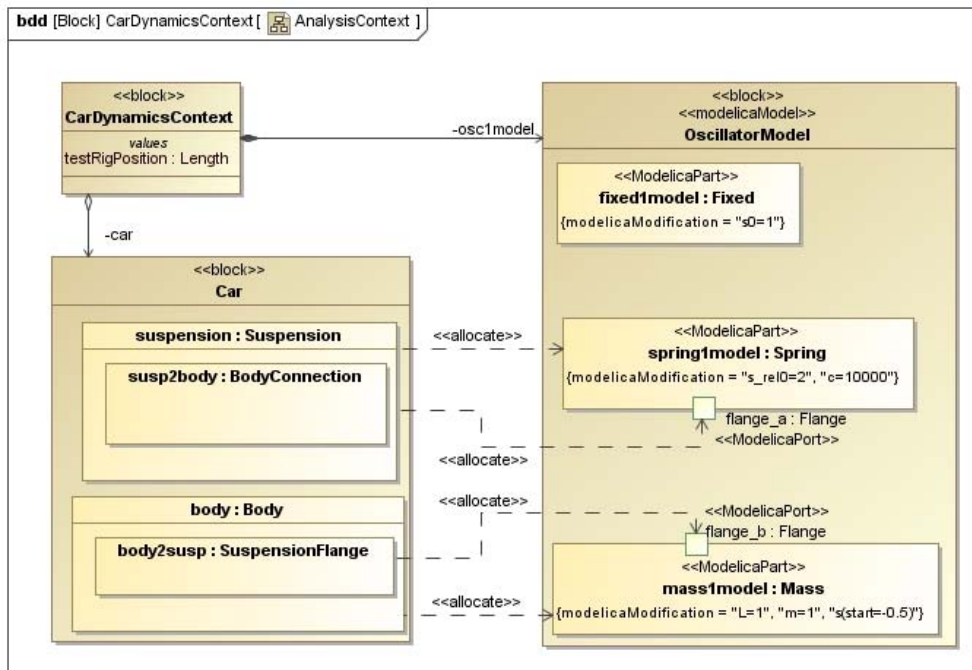


Figure 9: The Block Definition Diagram for the Analysis Context of the continuous dynamics analysis.

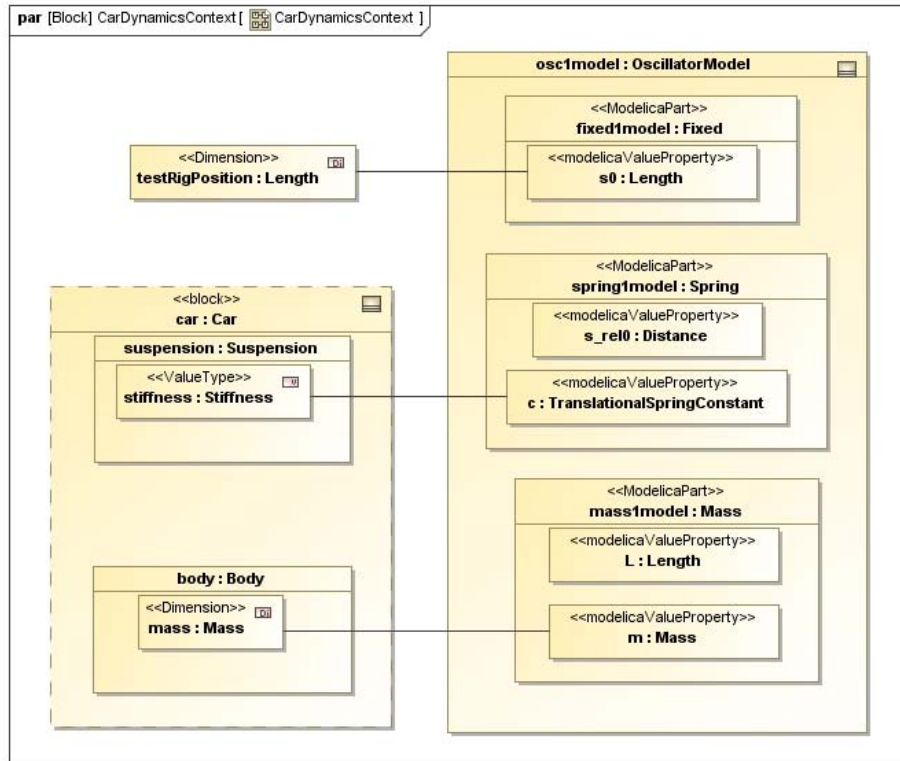


Figure 10: The Parametric Diagram for the Analysis Context of the continuous dynamics analysis; the properties of the descriptive model are bound to the corresponding properties in the analysis model.

For very simple problems, one could consider combining the descriptive and analysis views into one model; e.g., suspension and spring1model would be combined into one component that includes both the descriptive properties and the analysis constraints/equations. However, for larger problems in which more than one analysis perspective needs to be considered (e.g., mechanical, electrical, controls, manufacturing, different levels of abstraction, etc.), combining all such analyses into one model would be difficult to manage. One would likely encounter problems with naming conflicts or duplication of properties. In addition, combining all the models severely limits the opportunity for model reuse because models from libraries (such as the Modelica Standard Library) would have to be combined with descriptive models rather than just included in an analysis context.

Summary

In this paper, we have introduced a formal transformation between the SysML4Modelica profile and the Modelica language. This transformation is currently under development within the Systems Engineering DSIG of OMG. The SysML-Modelica transformation is defined as a formal, bi-directional transformation as envisioned in the Model-Driven Architecture approach. For each Modelica language construct as defined in the abstract syntax meta-model, a corresponding language construct in the SysML4Modelica profile is identified and a mapping relationship is established. [Note to reviewers: Implementations of this transformation are currently under development and the authors plan to include a pointer to the reference implementations when submitting the final version of this paper].

Acknowledgments

The authors would like to acknowledge the support for the MagicDraw tool provided by NoMagic Inc. In addition, the ideas presented in this paper benefitted from discussions with other members of the Systems Engineering DSIG, including Michael Chonoles, Sébastien Gérard, Nerijus Jankevicius, Steve Jenkins, Alek Kerzhner, Eldad Palachi, Russell Peak, and Ed Seidewitz. Finally, we would like to acknowledge the students who did pioneering thesis work in this direction at Linköping University and Georgia Tech: David Akhvlediani, Thomas Johnson, and Adrian Pop.

References

Fritzson, P. 2004. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. New York, NY: Wiley-IEEE Press.

Java Community Process. 2002. *Java™ Metadata Interface (JMI) Specification*. Sun Microsystems, Inc.

Johnson, T. A., Paredis, C. J. J., and Burkhart, R. M. 2008. Integrating models and simulations of continuous dynamics into SysML. *6th International Modelica Conference*, Modelica Association, Bielefeld, Germany, pp. 135-145.

Königs, A. 2005. Model transformation with triple graph grammars. *Model Transformations in Practice, Satellite Workshop of MODELS 2005* Montego Bay, Jamaica.

Modelica Association, 2009, Modelica Standard Library 3.1, <http://www.modelica.org/libraries/Modelica>.

Object Management Group. 2007. *XMI Mapping Specification, v2.1.1*. OMG, Needham, MA.

———. 2008. *OMG Systems Modeling Language (OMG SysML), v1.2*. OMG, Needham, MA.

———. 2008. *Semantics of a foundational subset for executable UML models*. OMG, Needham, MA.

———. 2008. *Query/View/Transformation, v1.0*. OMG, Needham, MA.

———. 2009. *UML 2.2 superstructure specification*. OMG, Needham, MA.

———. 2009. *Model-Driven Architecture*. OMG, <http://www.omg.org/mda/>.

OMG SE DSIG SysML-Modelica Working Group. 2009. *SysML-Modelica transformation specification*. http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-modelica:sysml_and_modelica_integration.

Pop, A., Akhvlediani, D., and Fritzson, P. 2007. Towards unified systems modeling with the ModelicaML UML profile. *International Workshop on Equation-Based Object-Oriented Languages and Tools*. Linköping University Electronic Press, Berlin, Germany.

Schamai, W., Fritzson, P., Paredis, C. J. J., and Pop, A. 2009. Towards unified system modeling and simulation with ModelicaML: Modeling of executable behavior using graphical notations. *7th Modelica Conference 2009*, ed. F. Casella. Linköping University Electronic Press, Como, Italy.

The Open Source Modelica Consortium. 2009. The Openmodelica Project. <http://www.openmodelica.org>.

Weisemöller, I., Klar, F., and Schürr, A. 2009. Development of tool extensions with MOFLON. *Model-Based Engineering of Embedded Real-Time Systems*, Heidelberg: Springer Verlag.