Date: October 2009

SysML-Modelica Transformation Specification

Version Alpha 1

OMG Document Number: syseng/2009-12-01

Standard document URL: http://www.omg.org/spec/acronym/1.0/PDF

Associated File(s)*: http://www.omg.org/spec/acronym/200xxxxx

http://www.omg.org/spec/acronym/200xxxxx

Source document: syseng/2009-12-01

* Original file(s): Title (document number)

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.)

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG

specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Part I — Introduction

1 Abstract

OMG 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 based on solving 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 SysML or Modelica individually. The objectives of this document are to enable and specify a standardized bi-directional transformation between the two modeling languages that will support implementations to efficiently and automatically transfer the modeling information transfer between SysML and Modelica models without ambiguity.

The transformation approach is to specify first an extension to SysML called the SysML4Modelica profile to represent the Modelica constructs and then to specify the SysML-Modelica Transformation between the profile constructs and the Modelica language. Introducing the profile into the transformation approach is intended to simplify the transformation to Modelica and 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 he 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 creates 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 modeling tool.

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

This specification is organized as follows:

Part I — Introduction

Part II — SysML4Modelica profile

Part III — Modelica meta-model

Part IV — SysML-Modelica mapping, a bidirectional mapping between the SysML4Modelica profile and the Modelica meta-model

Annex A — Robotic Sample Problem

2 Scope

OMG SysML™ is a general-purpose systems modeling language that can be used to create and manage models of systems using well-defined constructs with underlying semantics and a graphical notation. SysML reuses a subset of UML 2 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 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 how the state of the system changes (or must change) over the time according to its own dynamics and/or to external events. The requirements

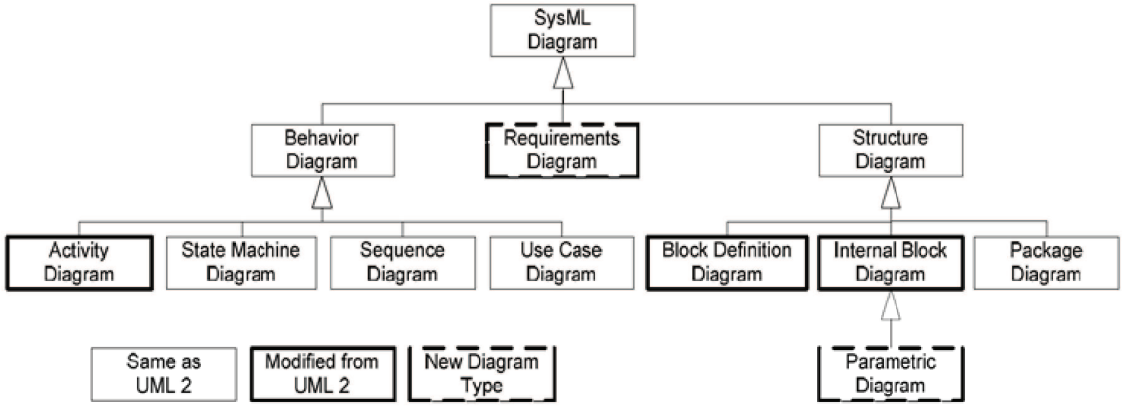


Figure 1: An overview of the SysML diagrams and their relation to UML diagrams.

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 a means to specify that interdependencies between values of some system properties and can provide a 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 graphs, state machine , and/or interactions specifications using diagrams and their associated semantics. The Foundational Subset of the UML specification provides the additional semantics to enable SysML activity graphs to be executed in a standard way. In addition, SysML includes parametric constructs 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.omgsysml.org.

Modelica is an object-oriented language for describing differential algebraic equation (DAE) systems combined with discrete events. 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, 2008] 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. 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 (www.openmodelica.org).

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.

The objective of this document is to provide a bi-directional mapping between SysML and Modelica to leverage the benefits from 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. The scope of this specification supports the objectives of the bi-directional mapping, and includes the SysML4Modelica profile, and the SysML-Modelica Transformation. Not all Modelica constructs will be represented in this profile. The focus is to include the Modelica language features that are most common and together cover the majority of the Modelica models in the standard library. When certain Modelica constructs are omitted, then this will be pointed out explicitly in this document. Changes to SysML and Modelica may be recommended as a result of this effort to enable the transformation, 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 effort.

3 Conformance

The Conformance clause identifies which clauses of the specification are mandatory (or conditionally mandatory) and which are optional in order for an implementation to claim conformance to the specification.

3.1 Compliance with SysML4Modelica profile

3.2 Compliance with the SysML-Modelica mapping

Note: For conditionally mandatory clauses, the conditions must, of course, be specified.

4 References

4.1 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- Systems Modeling Language: Specification, version 1.2 (http://www.omg.org/spec/SysML/1.2)
- Modelica Specification, v.3.1 (http://www.modelica.org/index_html/documents/Modelica Spec31.pdf)

4.2 Non-normative References

The following document contains provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- ormsc/09-02-01: MDA Foundation Model - Santa Clara AB initial comments draft (http://www.omg.org/members/cgi-bin/doc?ormsc/09-02-01.pdf)

5 Terms and Definitions

There are no formal definitions in this specification that are taken from other documents

6 Symbols

Acronyme	Meaning
CCM	CORBA Component Model
CORBA	Common Object Request Broker Architecture
CWM	Common Warehouse Metamodel
IDL	Interface Definition Language
MDA	Model Driven Architecture
MOF	Meta Object Facilities
OMG	Object Management Group
SysML	System Modeling Language
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

7 Additional Information

7.1 Changes to Adopted OMG Specifications [optional]

The following are proposed changes to the SysML Specification and designated as required or desired in order to support this specification:

(TBD)

The following are proposed changes to the Modelica Specification and designated as required or desired in order to support this specification:

(TBD)

7.2 Acknowledgments

The following companies submitted this specification:

- NoMagic Inc.
- Sparx Systems

The following companies supported this specification:

- Deere & Company
- EADS
- Georgia Institute of Technology
- Jet Propulsion Laboratory
- Linköping University
- Lockheed Martin Corporation
- NoMagic Inc.

The following people have contributed significantly to this document either directly or indirectly through discussions and feedback:

- Yves Bernard (EADS)
- Roger Burkhart (Deere & Co)
- Hans-Peter De Koning (ESA)
- Sanford Friedenthal (Lockheed Martin)
- Peter Fritzson (Linköping University)
- Nerijus Jankevicius (NoMagic Inc)
- Thomas Johnson (Georgia Tech)
- Alek Kerzhner (Georgia Tech)
- Chris Paredis (Georgia Tech)
- Russell Peak (InterCAx, Georgia Tech)
- Nicolas Rouquette (Jet Propulsion Laboratory)
- Wladimir Schamai (EADS, Linköping University)

8 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 transformations between SysML and Modelica. The profile extends the UML4SysML subset of UML and the SysML extensions to provide the concept required to capture the

Page : 5 Line : 367 Author : Chris Paredis 11/18/2009
Linkoping also submitter?
Is it restricted to platform member?

Page : 5 Line : 391 Author : Chris Paredis 11/18/2009
Include list of reviewers and other contributors

relevant Modelica semantics and enable the mapping between the two languages.

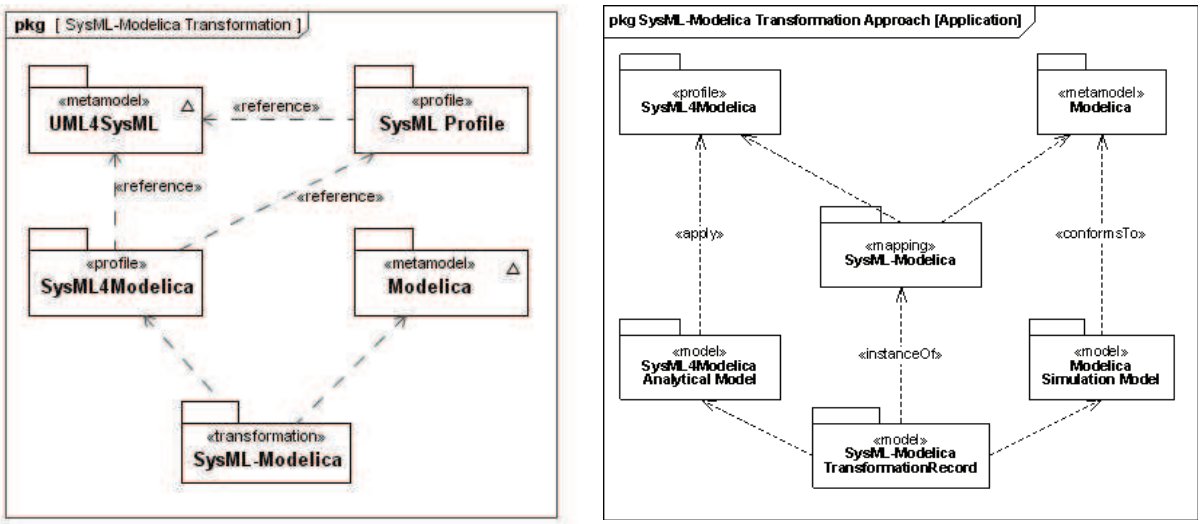


Figure 2: The SysML-Modelica Transformation in relation to SysML and Modelica.

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 from a semantic point of view. Where equivalent constructs do not exist, stereotypes are created to extend the SysML language. The following naming convention is used to define a Modelica construct in the SysML4Modelica profile: «modelica*Construct*» where *Construct* is the name of the Modelica language construct as defined in the Modelica abstract syntax definition (see Chapter qq-insert ref after combining documents).

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.

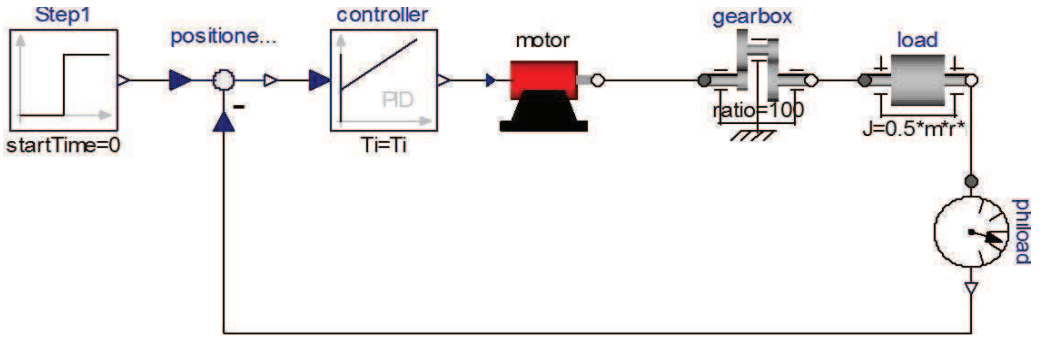


Figure 3: A Modelica model of a motor controller consisting of component models and the connections between them. The connections include both causal signal connections (e.g., in and out of the controller) and acausal energy connections (e.g., the rotational mechanical energy connections of the gearbox).

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, 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) can be used to generate executable transformations

between SysML and Modelica modeling tools (assuming they support some standardized interface such as JMI). An additional implementation of the mapping is being developed as part of the OpenModelica project.

8.1 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 models. 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 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 kind of construct built on abstractions thathave similar semantics compared to the hierarchical, connector-based composition of Modelica models: the hierarchical Blocks ,shown in Internal Block Diagrams), the Parametric Constraints (shown in Parametric Diagrams), and the Activity graphs. All three constructs support some sort of "ports", some sort of connection of "port-based" objects through "port-connections", and hierarchical encapsulation through "port-delegation". In Sections 8.1.1 through 8.1.4, we use these three constructs to discuss the main question: what are the SysML elements that match the Modelica semantics best?

Table 1 shows also how the Block Definition Diagram (BDD) may complement other kind of diagrams to visualize those constructs of .

Concepts	Modelica constructs	SysML				
		Constructs abstractions	Availability in diagrams < -------- Modelica "like" -------- >			
			BDD	IBD	Parametric	Activity
Model Definition	Model	Block	Yes	Yes	Restricted	No
Model Usage	Component	Part Property	Yes	Yes	Restricted	No
Port Definition	Connector	Block	Yes	Yes	Yes	No
		ValueType	Yes	Yes	No	No
		FlowSpecification	Yes	No	No	No
Properties of Ports	Variables	Block	Yes	Yes	Yes	Ref. Only
		ValueType	Yes	Yes	Yes	Ref. only
		FlowProperty	Yes	No	No	Ref. only
Port Usage	Component	Port Property	Yes	Yes	Yes	Ref. only
Causal link	Connection	Connector	No	Yes	No	No
		ObjectFlow	No	No	No	Yes
Acausal link	Connection	BindingConnector	No	Yes	Yes	No

Table 1: A comparison between Modelica concepts and SysML abstractions and diagrams

Page : 7 Line : 438 Author : Chris Paredis 10/25/2009
This “diagram-based” approach is a stopgap solution. I think going through each kind of Modelica Connector/Connection would have been a better choice.

8.1.1 Modelica

In Modelica, ports are called connectors and the edges between ports are called connections [Modelica Spec, Chapter 9]. 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 connecters 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 submodels (i.e., model usage in Table 1). Although Modelica does not explicitly distinguish between these three categories of "components" (i.e., connectors, variables, submodels), it may still be useful and desirable to distinguish explicitly among them when mapping to SysML.

8.1.2 SysML Hierarchical Blocks

The primary purpose of the SysML hierachical Block constructs, is to express system structural decomposition and interconnection of its parts [SysML Spec, Chapters 8 and 9]. The SysML concepts used in those constructs have quite flexible semantics and may be used to establish logical and conceptual decompositions, for instance, as in a context view [SysML Spec, Section B.4.2.1].] 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" on the blocks 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.[1] 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 the (acausal) connection between flow ports in SysML does not carry the Kirchhoff semantics as for energy/material connections in Modelica.

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

8.1.3 SysML Parametric Constraints

The purpose of Parametric Constraints is to express mathematical relationships between parameters. A Parametric Constraints is modeled through a special kind of Block named "Constraint Block". "Ports" of those blocks are Constraint Parameters and the "connections" to those parameters are made using 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.

[1] Note that an *instance* in Modelica is similar to a *usage* in UML or SysML.

Page : 8 Line : 474 Author : Chris Paredis 10/25/2009
I would say that “acausal” SysML connector (ie. BindingConnectors) does explicitly not carry the Kirchhoff semantic, since they imply equality of values on both ends.

Page : 8 Line : 474 Author : Yves BERNARD 02/24/2010
Répondre à Chris Paredis (10/25/2009, 19:34): "..."
Modified. But I believe that it's worth dedicate a full paragraph to that point.

Page : 8 Line : 485 Author : Chris Paredis 10/25/2009
Note that the usage of Binding connectors is restricted neither to constraint block nor to parametric diagrams.

Page : 8 Line : 485 Author : Yves BERNARD 03/03/2010
Reply to Chris Paredis (10/25/2009, 19:35): "..."
This information is provided in table 1

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. The causal semantics is nevertheless provided by the classical SysML connectors (cf. Table 1). 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 solution would be 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."

8.1.4 SysML Activity Graphs

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 for flows variables in Modelica.

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 seems to be the least appropriate for a mapping from Modelica Class, although they will be explored when mapping the Modelica Function and Algorithm to SysML4Modelica.

8.1.5 Selected foundation concept: SysML Hierachical Block with Embedded Constraints

It is clear from the discussion in the previous sections that there is not a single concept that embeds the Modelica semantics perfectly. As a result, the use of more than one SysML concept with multiple stereotypes will need 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. This is illustrated in Section 8.2

8.2 Illustrative Example

The following example is intended to illustrate the concepts of how the transformation approach can be used to provide a context for the normative specification in Part II of this specification. 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 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.

Page : 9 Line : 488 Author : Chris Paredis 10/25/2009
I think it is already available through classical SysML connectors. Using together classical properties, ConstraintProperties, classical connectors and binding connectors should cover the need.

Page : 9 Line : 488 Author : Yves BERNARD 03/03/2010
Reply to Chris Paredis (10/25/2009, 19:36): "..."
Modified

Page : 9 Line : 508 Author : Chris Paredis 10/25/2009
I'm not sure of that. Do we have a well accepted mathematical definition of causality? Don't we speak about physical causality there? Then, don't input/output variables in Modelica and input/output pins in Activity graphs have this physical causality semantic?

Page : 9 Line : 521 Author : Yves BERNARD 03/03/2010
The philosophy of this example have to be discussed : should it really refere to the not yet defined SysML4Modelica profile or just give the initial data for an example that will be used an a n illustration all along the profile definition that is provided by the following parts? (my preference)

Page : 9 Line : 521 Author : Chris Paredis 10/25/2009
This paragraph should be restricted to the description of the Modelica version of the model, with its textual and graphical part. Showing the SysML counter-part based on stereotype that has not been already defined provide more confusion than help.

Page : 9 Line : 521 Author : Chris Paredis 10/25/2009
Issues in having analytical model in SysML
Redundancy between descriptive/design and analytical model
Potential inconsistency between descriptive/design and analytical model current solution: analysis context
Difference in paradigm for dynamic models (Modelica) and other modelling paradigms (

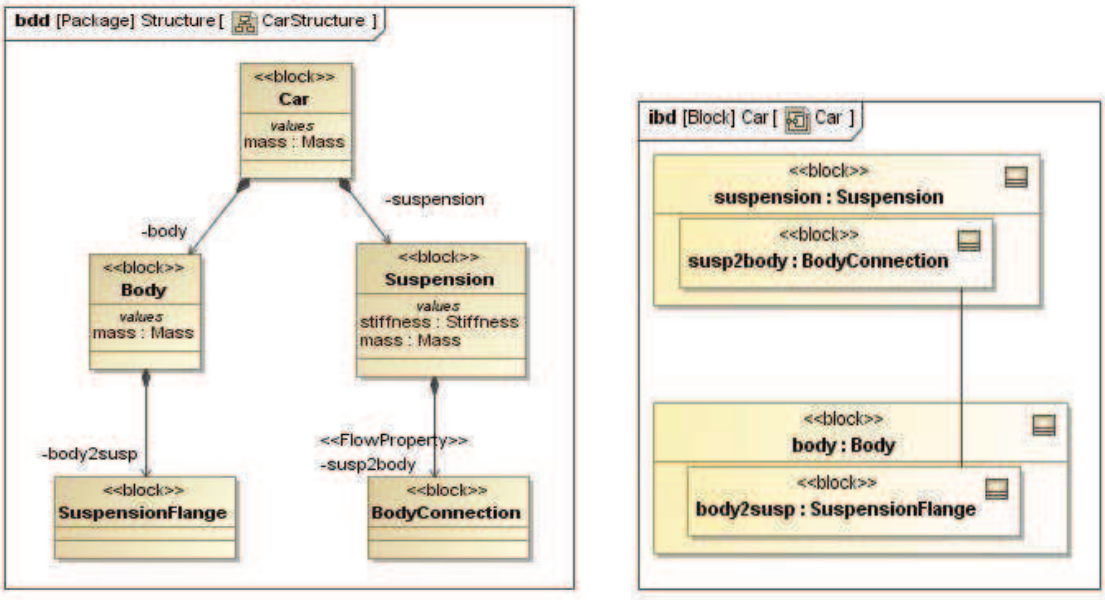


Figure 4: SysML descriptive model of a car suspension visualized as a BDD and IBD.

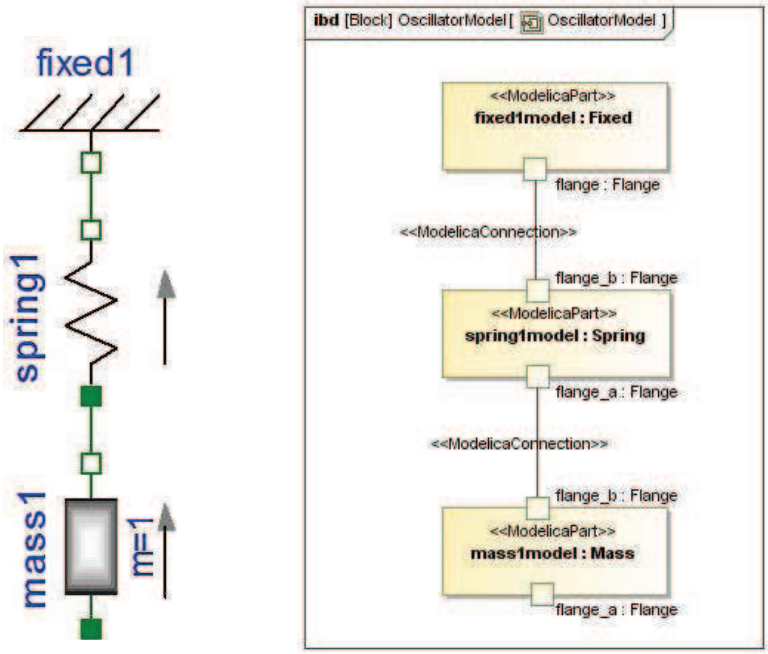


Figure 5: Mass-Spring model for a car suspension, in Modelica (left) and SysML4Modelica (right).

Page : 10 Line : 1 Author : Chris Paredis 10/25/2009
Could we give the Modelica textual representation along with or in place of the graphical one? As far as I understand, Modelica diagrams are just annotations. They don't provide all the information (cf §4.1.3). Then they should not be used as a reference for SysML mapping.

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 4 could be almost identical to the Modelica representation on the left.

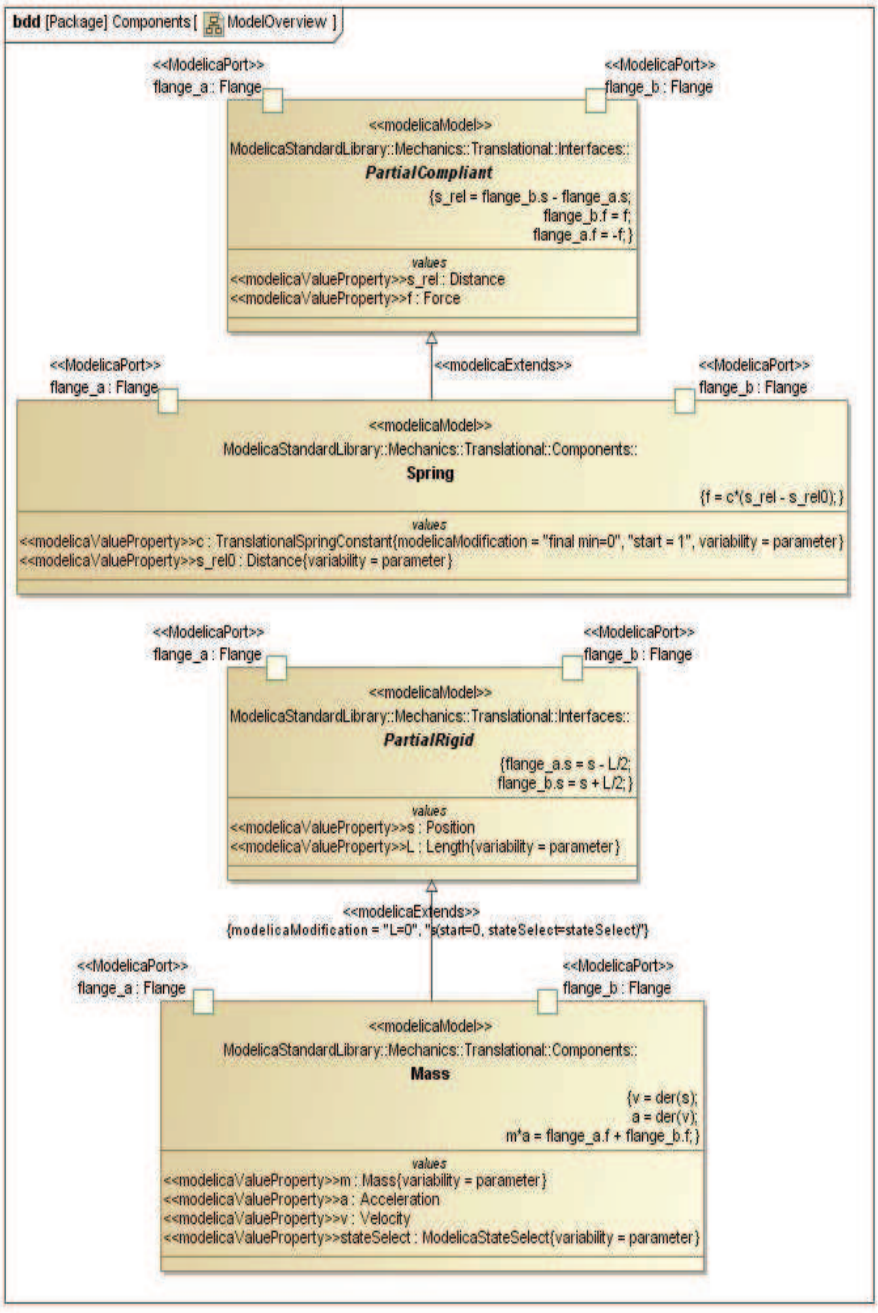


Figure 6: Continuous dynamics models for Mass and Spring defined in the Modelica Standard Library.

In Figure 5, the usages of these models, stereotyped as «modelicaPart» are connected to each other at their

Page : 11 Line : 537 Author : Chris Paredis 10/25/2009
Not so simple as discussed with Nerijus in San Antonio…

sf. Once can still apply icons to individual model elements, so I think this is accurate. For example, one could specify a stereotype called icon, that includes the image as a stereotype property and apply this stereotype to selected model elements. Perhaps we should include this as a recommendation or Nerijus proposal.

«modelicaPort» by «modelicaConnection». These connections carry the semantics of Kirchhoff's Laws (in this example —or, more generally, the same semantics as an equivalent Modelica connection). These semantics can be made more explicit by using a Parametric Constraint (Figure 7).

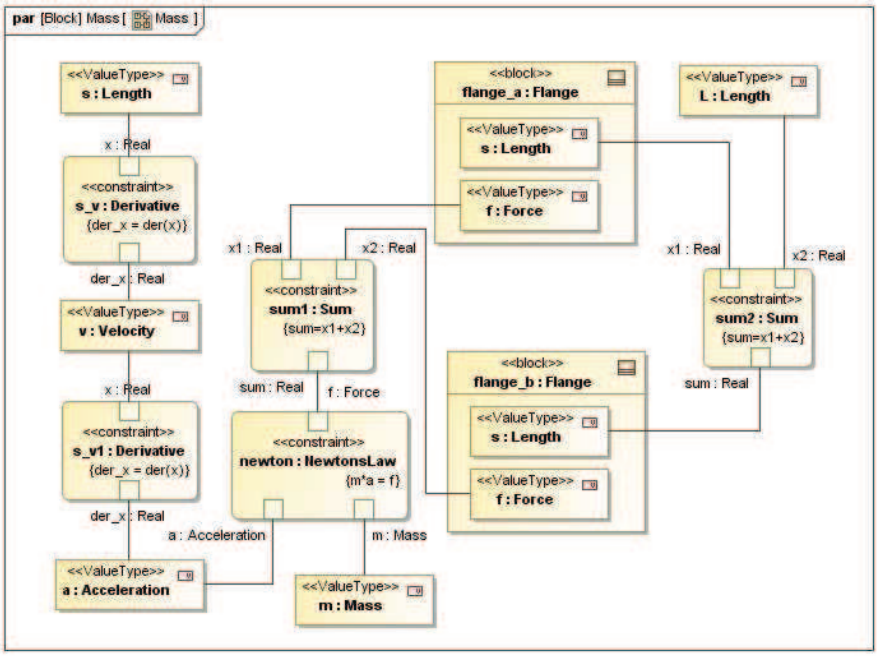


Figure 7: Mass model as it could be represented in a Parametric Diagram.

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 further used in the SysML4Modelica profile, but the parametrics still provides the underlying semantics for capturing the detailed equations. However, this complexity can often be abstracted and made not visible to the modeler.

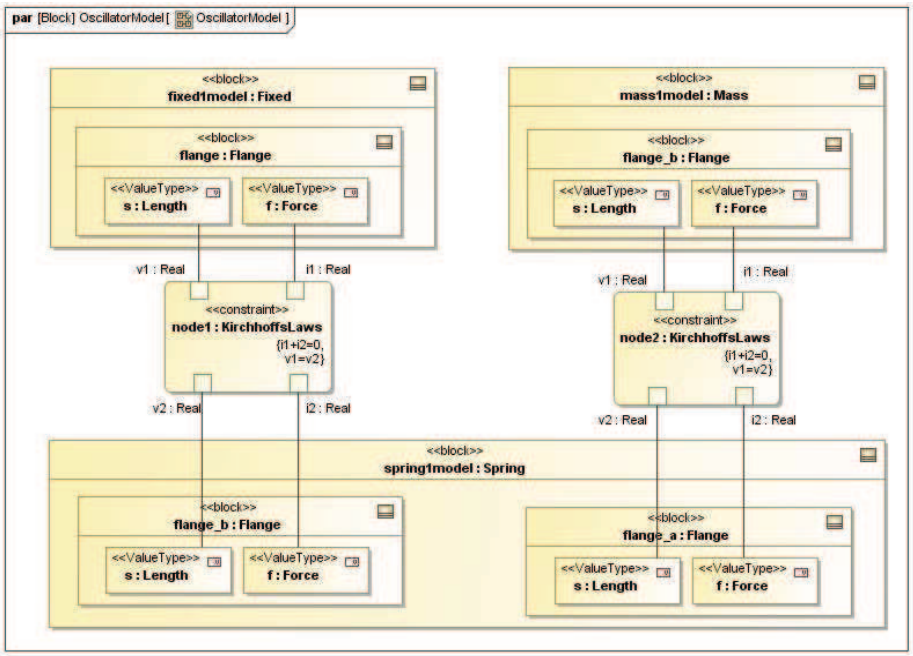


Figure 8: Mass-Spring model as represented in a Parametric Diagram.

Page : 12 Line : 1 Author : Chris Paredis 10/25/2009
Could we give the Modelica textual representation along with or in place of the graphical one? As far as I understand, Modelica diagrams are just annotations. They don’t provide all the information (cf §4.1.3). Then they should not be used as a reference for SysML mapping.

Page : 12 Line : 546 Author : Sanford Friedenthal 11/01/2009
Let's work on this wording so we are in agreement on the role of parametrics in this profile.

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.

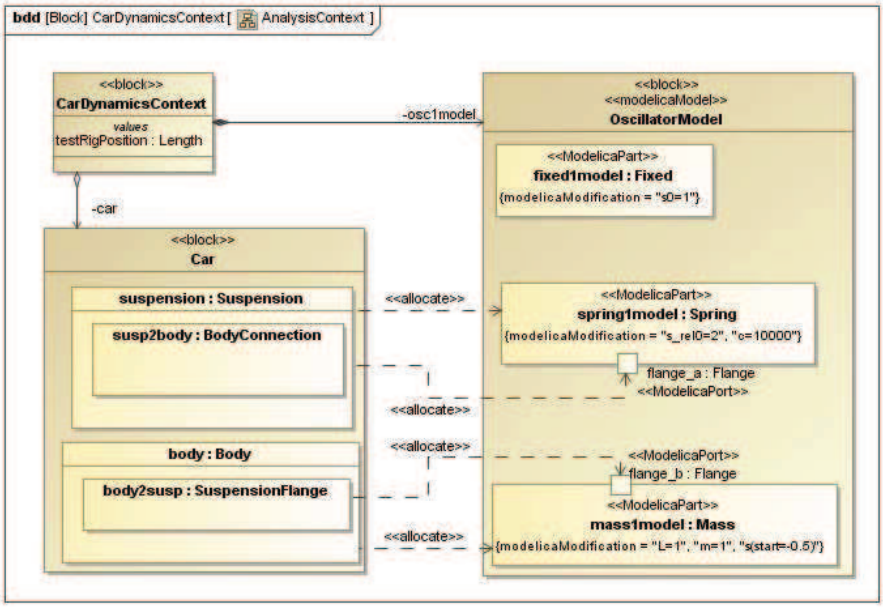


Figure 9: The Block Definition Diagram for the Analysis Context of the continuous dynamics analysis.

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.

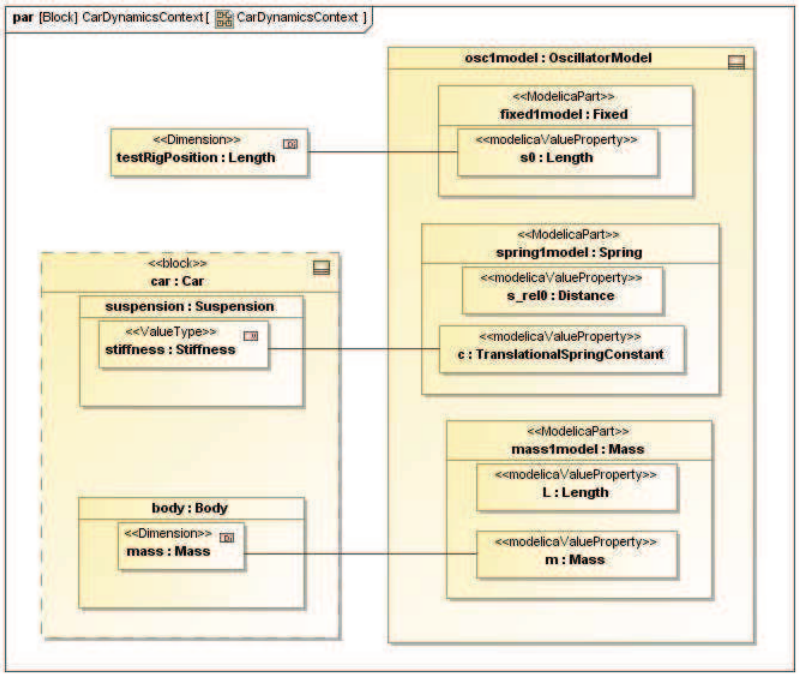


Figure 10: The Parametric Diagram for the Analysis Context of the continuous dynamics analysis; the properties of the descriptive model are bound to the corresponding properties in the analysis model.

Page : 13 Line : 554 Author : Chris Paredis 10/25/2009
The terminology in the text needs to be related explicitly to the notation used in the diagrams.
What is an analysis context in the model?
(maybe we should have a stereotype)
What are descriptive elements in the model?
What are analysis models in the model?
-How are dependencies captured in the model?
(it may be not obvious for some readers to equate “dependency” in the text with a <<Describe>> relationship in the model or an allocation relationship as Sandy is suggesting)

Page : 13 Line : 555 Author : Chris Paredis 10/25/2009
Perhaps we could use allocation for this dependency to represent to the mapping between the two user models.

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.

Part II – SysML4Modelica Profile

This part of the SysML-Modelica Transformation Specification describes the stereotypes that represent the Modelica modeling constructs in SysML. As illustrated in Figure 1, the stereotypes, together with the library of predefined types, are organized in sub-packages and profiles in the the SysML4Modelica profile. In Chapter 1, all the stereotypes related to the Modelica restricted classes are introduced. In Chapter 2, the predefined Modelica types and the enumerations used in the SysML4Modelica profile are defined. In Chapter 3, the Modelica equivalent of properties are defined — called Component Declarations in Modelica. Finally, in Chapter 4, the Equation and Algorithm sections of Modelica models are covered.

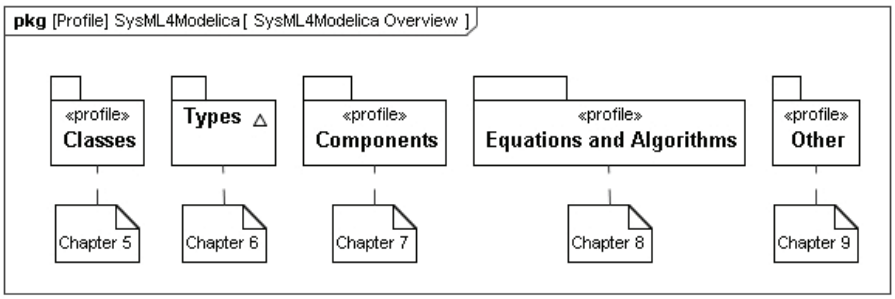


Figure 1: Package diagram with an overview of the SysML4Modelica profile.

1 Class Definition

1.1 Overview

The class concept is the basic structural unit in Modelica. Classes provide the structure for objects and contain equations and algorithms, which ultimately are the basis for the executable simulation code. The most general class is "model". Specialized classes such as "record", "type", "block", "package", "function" and "connector" have most of the properties of a "model" but with restrictions, which need to be preserved in SysML to support round-trip mapping.

The following production rules define the different specialized classes. The reference in parentheses on the right indicates the section of this document in which the particular language element is discussed in detail:

```
stored_definition:
   [ within [ name ] ";" ]                                    (1.2)
   { [ final ] class_definition ";" }                         (1.2)

class_definition :
   [ encapsulated ]                                           (1.2)
   [ partial ]                                                (1.2)
   (  class                                                   (1.2)
    | model                                                   (1.3)
    | record                                                  (1.4)
    | block                                                   (1.5)
    | [ expandable ] connector                                (1.6)
    | type                                                    (1.7)
    | package                                                 (1.8)
    | function )                                              (1.9)
   class_specifier
```

```
class_specifier :
     IDENT string_comment composition                                          (3)
   | IDENT "=" base_prefix name [ array_subscripts ]                           (1.7)
   [ class_modification ] comment                                              (3)
   | IDENT "=" enumeration "(" ( [enum_list] | ":" ) ")" comment               (1.7)
   | IDENT "=" der "(" name "," IDENT { "," IDENT } ")" comment                (1.9)
   | extends IDENT [ class_modification ] string_comment composition           (1.10)
   end IDENT
```

The following table lists the SysML stereotypes for representing the specialized Modelica classes. Using this approach the modeler only needs to apply the respective stereotype to indicate all the semantics and restrictions of the associated Modelica class. This information is represented graphically in . In the following subsections, the details of each stereotype are described.

Table 1: Mapping for the Modelica specialized classes.

Modelica Construct	SysML Base Class	SysML4Modelica	
		New Stereotype	Comments
abstract generalization for all Modelica classses	UML4SysML::Classifier	«modelicaClassDefinition»	See Section 1.2
Class and Model	SysML::Blocks::Block	«modelicaModel»	See Section 1.3
Record	SysML::Blocks::Block	«modelicaRecord»	See Section 1.4
Block	SysML::Blocks::Block	«modelicaBlock»	See Section 1.5
Connector	SysML::Blocks::Block	«modelicaConnector»	See Section 1.6
Type	SysML::Blocks::Block SysML::Blocks::ValueType UML4SysML::Enumeration	«modelicaType»	See Sections 1.7
Package	SysML::Blocks::Block	«modelicaPackage»	See Section 1.8
Function	UML4SysML::FunctionBehavior	«modelicaFunction»	See Section 1.9

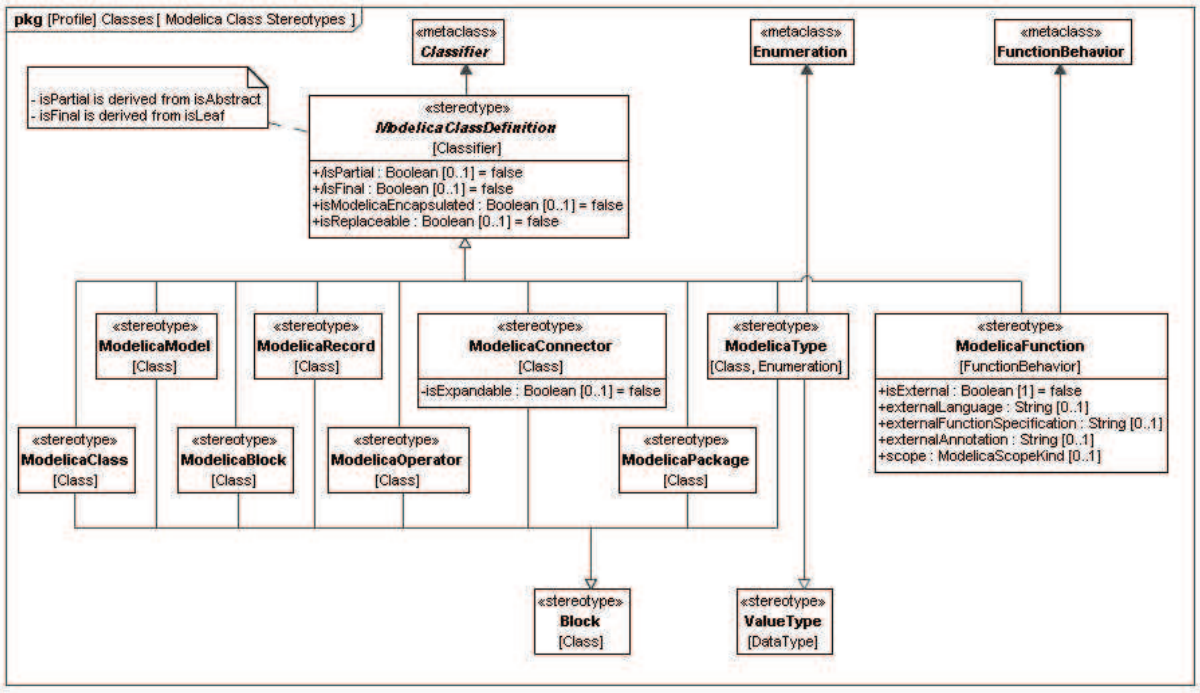


Figure 2: Package diagram with an overview of the stereotypes for Modelica Classes

1.2 «modelicaClassDefinition»

Stereotypes

- Classifier (from UML4SysML)

Abstract Syntax

- See Figure 2.

Description

A Modelica `class` is the basic structural unit in Modelica. However, because it lacks precise semantics, the `class` construct should never be used in Modelica. Without precise semantics, a Modelica tool cannot easily check whether any restrictions are violated. Therefore, the constructs that are specialized from Modelica `class` should be used instead.

In the context of the SysML4Modelica profile, the Modelica `class` construct is mapped to the stereotype «modelicaClassDefinition» which is abstract and thus cannot be instantiated directly. This choice has been made because it is desirable to have the additional semantics specified by the specialized classes. In addition, as clearly shown in Figure 2, the stereotypes associated with the specialized classes derive from different SysML constructs and thus cannot be mapped to a single common construct for a Modelica `class`. The abstract stereotype «modelicaClassDefinition» serves the purpose of grouping the attributes that apply to all the Modelica specialized classes. It stereotypes UML::Classifier, which is a common generalization for the stereotypes of all the specialized classes.

Just like UML Classifiers, a «modelicaClassDefinition» can contain nested class definitions. Such nested definitions can be of any restricted class type derived from «modelicaClassDefinition». For instance, a «modelicaConnector» can contain a «modelicaPackage».

Modelica classes are often defined using a short class definition syntax. For example, the type `Force` could be defined as:

```
type Force = Real[3](unit={"N.m","N.m","N.m"});
```

Rather than supporting such short class definitions explicitly, the SysML4Modelica profile supports only the longer

(but equivalent) form (Note: in the Modelica abstract syntax the two forms are often represented identically):

```
type Force
  extends Real[3](unit={"N.m","N.m","N.m"});
end Force;
```

In the remainder of this section, all the common attributes and associations for all the constructs specialized from Modelica `class` are described. In subsequent sections for the individual specialized constructs, only the constraints on these attributes and associations will be described in detail.

Attributes

- /isFinal : Boolean [0..1]
  In Modelica, the definition of a class can be qualified to be `final` (Modelica Specification 7.2.6). This means that the declared class cannot be further modified through (local) type modifications. Note that this is identical to the UML attribute isLeaf for redefinable elements (UML Specification 7.3.46) which, if true, indicates that no further redefinitions are possible.
  The isFinal attribute is true when the `final` prefix is present in Modelica; false otherwise. Its default value is *false*. This is derived from *isLeaf*.

- /isPartial : Boolean [0..1]
  The Modelica `partial` construct has the same semantics as the isAbstract attribute in SysML. The isPartial attribute is true when the `partial` prefix is present in Modelica; false otherwise. Its default value is false. This is derived from isAbstract.

- isModelicaEncapsulated : Boolean [0..1]
  As explained in Modelica Specification 5.3.2, the Modelica `encapsulated` construct limits the scope of name lookup. An `encapsulated` package can be moved within the package hierarchy without affecting the local name resolutions. These semantics are different from the isEncapsulated attribute of Blocks in SysML (SysML Specification 8.3.2.2). An encapsulated block is treated as a black box; no connections can be made to its internal parts directly. A second difference in semantics is that in Modelica the `encapsulated` prefix can be applied to *all* classes, although it is most commonly applied to packages. It is therefore necessary to introduce isModelicaEncapsulated as a new attribute so that it becomes available also for specialized class stereotypes that do not derive from a SysML Block.
- The isModelicaEncapsulated attribute is true when the `encapsulated` prefix is present in Modelica; false otherwise. Its default value is *false*.

- isReplaceable : Boolean [0..1]
  As explained in Modelica Specification 7.3, the Modelica prefix `replaceable` is most commonly applied to components (see Section 3) , but can also be applied to a Modelica `class` to indicate that a local model definition can be redeclared when the containing model is used. The isReplaceable attribute is true when the `replaceable` prefix is present in Modelica; false otherwise. Its default value is *false*.

- fromLibrary : String [0..1]
  A model in SysML4Modelica often corresponds to a model that has already been defined in a Modelica library. Rather than duplicating the entire definition of such a model, the attribute fromLibrary is used to specify the fully qualified path to the model in the Modelica library. When converting such a SysML4Modelica model to Modelica, the definition of the models is omitted and instead the fully qualified library path is used as type. In addition, for such models, only the ports are defined in SysML4Modelica so that they can still be connected to other models. All other details (value properties and parts) are omitted because they are already defined in the corresponding Modelica library.
  The fromLibrary attribute should only be defined when a corresponding Modelica model exists. Its default value is *null*.

Associations

No additional associations.

Page : 4 Line : 83 Author : Chris Paredis 03/15/2010
Need feedback here. Is this a good idea?

Constraints

[1] isFinal has the same value as isLeaf
isFinal = self.isLeaf

[2] isPartial has the same value as isAbstract
isPartial = self.isAbstract

[3] «modelicaClassDefinition» is an abstract stereotype and cannot be instantiated.

[4] Any generalization relationship to/from «modelicaClassDefinition» must be stereotyped to a «modelicaExtends» relationship.

[5] A «modelicaClassDefinition» can only contain nestedClassifiers stereotyped to a restricted type specializing «modelicaClassDefinition».

Additional Notes

The Modelica `within` clause is explained in Modelica Spec. 3.1, Section 13.2.2.3. It defines where in the package hierarchy the subsequent class definitions are located. This is important in Modelica to allow large package structures to be divided over multiple model files. As long as fully qualified type identifiers are used, the `within` clause is not relevant in SysML4Modelica and is therefore not supported in the SysML4Modelica profile.

1.3 «modelicaClass» and «modelicaModel»

Generalizations

- «modelicaClassDefinition» (from SysML4Modelica::Classes)
- «block» (from SysML)

Abstract Syntax

- See Figure 2.

Description

The Modelica specialized class `model` is the most general specialized class; it is equivalent to the general Modelica `class` construct. All the Modelica class elements are allowed in models: variables, connectors, sub-models, equations and algorithm sections. A model can also include state variables. Modelica does not differentiate between a `model` and a `class`. Although redundant, we therefore include both the equivalent stereotypes «modelicaClass» and «modelicaModel».

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

(All constraints apply to both «modelicaClass» and «modelicaModel»)

[1] A «modelicaModel» must have a Name.

[2] A «modelicaModel» can only have Properties that are stereotyped to «modelicaPart», «modelicaPort», or «modelicaValueProperty».

[3] A «modelicaModel» can only contain Behaviors that are stereotyped to «modelicaFunction», or «modelicaAlgorithm».

[4] A «modelicaModel» can only be contained in a «modelicaClassDefinition».

[5] A «modelicaModel» can only specialize other classifiers derived from «modelicaBlock», or «modelicaRecord». The stereotype «modelicaExtends» must be applied to the generalization relationship.

[6] All other attributes or associations inherited from «block» or Classifier are not relevant and should be set to their default values. This includes the attributes: isActive, isEncapsulated.

1.4 «modelicaRecord»

Generalizations

- «modelicaClassDefinition» (from SysML4Modelica::Classes)
- «block» (from SysML)

Abstract Syntax

- See Figure 2.

Description

The Modelica specialized class `record` is restricted to contain only public declarations of components that in turn also contain only public declarations. A complete description of `record` is available in Modelica Specification, Section 4.6:

```
Only public sections are allowed in the definition or in any of its com-
ponents (i.e., equation, algorithm, initial equation, initial algorithm
and protected sections are not allowed). May not be used in connections.
The elements of a record may not have prefixes input, output, inner,
outer, or flow. Enhanced with implicitly available record constructor
function. Additionally, record components can be used as component refer-
ences in expressions and in the left hand side of assignments, subject to
normal type compatibility rules.
```

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

[1] A «modelicaRecord» must have a Name.

[2] A «modelicaRecord» can only have Properties that are stereotyped to «modelicaValueProperty».

[3] Any «modelicaValueProperty» owned by an instance of «modelicaRecord» must have *visibility=public, flowFlag=nonflow, causality=null*, *scope=null*.

[4] A «modelicaRecord» can only be contained in a «modelicaClassDefinition».

[5] A «modelicaRecord» can only specialize other classifiers derived from «modelicaRecord». The stereotype «modelicaExtends» must be applied to the generalization relationship.

[6] All other attributes or associations inherited from «block» or Classifier may not be used. This includes the attributes: isActive, isEncapsulated; and the ownedElements: Behavior, Constraint.

1.5 «modelicaBlock»

Generalizations

- «modelicaClassDefinition» (from SysML4Modelica::Classes)
- «block» (from SysML)

Abstract Syntax

- See Figure 2.

Description

The Modelica specialized class `block` is very similar to a `model` except that all its connectors must be either an input or output making it similar to a Simulink block. A complete description of `block` is available in Section 4.6 of the Modelica Specification:

```
Same as model with the restriction that each connector component of a
block must have prefixes input and/or output for all connector variables.
[The purpose is to model input/output blocks of block diagrams. Due to the
restrictions on input and output prefixes, connections between blocks are
only possible according to block diagram semantic]
```

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

[1] A «modelicaBlock» must have a Name.

[2] A «modelicaBlock» can only have Properties that are stereotyped to «modelicaPart», «modelicaPort», or «modelicaValueProperty».

[3] Any «modelicaValueProperty» owned by an instance of «modelicaBlock» must have *causality=input* or *output*.

[4] A «modelicaBlock» can only contain Behaviors that are stereotyped to «modelicaFunction», «modelicaAlgorithm», or «modelicaInitialAlgorithm».

[5] A «modelicaBlock» can only contain Constraints that are stereotyped to «modelicaEquation» or «modelicaInitialEquation».

[6] A «modelicaBlock» can only be contained in a «modelicaClassDefinition».

[7] A «modelicaBlock» can only specialize other classifiers derived from «modelicaBlock» or «modelicaRecord». The stereotype «modelicaExtends» must be applied to the generalization relationship.

[8] All other attributes or associations inherited from «block» or Classifier may not be used. This includes the attributes: isActive, isEncapsulated.

1.6 «modelicaConnector»

Generalizations

- «modelicaClassDefinition» (from SysML4Modelica::Classes)
- «block» (from SysML)

Abstract Syntax

- See Figure 2.

Description

The Modelica specialized class `connector` is a `model` that cannot contain equations or algorithms in any of its components. A complete description of `block` is available in Section 4.6 and Chapter 9 of the Modelica Specification:

```
No equations or algorithms are allowed in the definition or in any of its
components. Enhanced to allow connect(..) to components of connector
classes.
```

Attributes

- isExpandable : Boolean [0..1]
  As explained in Modelica Specification 9.1.3, the Modelica `expandable` prefix can be applied to a `connector`. The primary purpose of expandable connectors is to allow for the convenient modeling of bus interfaces. The isExpandable attribute is true when the `expandable` prefix is present in Modelica; false otherwise. The default value is false.

Associations

No additional associations.

Constraints

[1] A «modelicaConnector» must have a Name.

[2] A «modelicaConnector» can only have Properties that are stereotyped to «modelicaPart», «modelicaPort», or «modelicaValueProperty».

[3] None of the Properties owned by an instance of «modelicaConnector» can contain Behaviors or Constraints (at any level of containment).

[4] A «modelicaConnector» can only be contained in a «modelicaClassDefinition».

[5] A «modelicaConnector» can only specialize other classifiers derived from «modelicaConnector», «modelicaType», or «modelicaRecord». The stereotype «modelicaExtends» must be applied to the generalization relationship.

[6] All other attributes or associations inherited from «block» or Classifier may not be used. This includes the attributes: isActive, isEncapsulated; and the ownedElements: Behavior, Constraint.

1.7 «modelicaType»

Extensions

- Enumeration (from UML4SysML)

Generalizations

- «modelicaClassDefinition» (from SysML4Modelica::Classes)
- «valueType» (from SysML)
- «block» (from SysML)

Abstract Syntax

- See Figure 2.

Description

The Modelica specialized class `type` is restricted to predefined types, enumerations, arrays of type or classes extending from type. It is enhanced to allow extension of predefined types. In the SysML4Modelica profile, the extension from predefined types is handled by making the predefined types instances of «modelicaType» (See Chapter 2).

The only additional distinct type for which a corresponding representation needs to be defined in SysML4Modelica is an enumeration. Enumerations are handled separately through the stereotype «modelicaEnumeration»

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

[1] A «modelicaType» must have a Name.

[2] A «modelicaType» can only be contained in a «modelicaClassDefinition».

[3] A «modelicaType» can only specialize other classifiers derived from «modelicaType». The stereotype «modelicaExtends» must be applied to the generalization relationship.

[4] All other attributes or associations inherited from «block» or Classifier may not be used. This includes the attributes: isActive, isEncapsulated; and the ownedElements: Property (including part, port and value properties), Behavior, Constraint.

1.8 «modelicaPackage»

Generalizations

- «modelicaClassDefinition» (from SysML4Modelica::Classes)
- «block» (from SysML)

Abstract Syntax

- See Figure 2.

Description

A Modelica `package` has broader semantics than just a container for other model elements as in SysML. Although it may only contain declarations of classes and constants, these declarations can be replaceable and can be inherited from parent packages, so that the package itself should be thought of as a model. The corresponding SysML4Modelica construct, «modelicaPackage», therefore generalizes «block» rather than Package. As compared to Modelica `class`, a Modelica `package` is enhanced to allow for the import of elements of packages. (See also Modelica Spec. 3.1, Chapter 13.)

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

[1] A «modelicaPackage» must have a Name.

[2] A «modelicaPackage» can only have Properties that are stereotyped to «modelicaValueProperty».

[3] Any «modelicaValueProperty» owned by an instance of «modelicaPackage» must have *variability=constant*. (ref. Modelica Specification 4.6, package)

[4] A «modelicaPackage» can be contained in a «modelicaClassDefinition» or in a UML4SysML::Package.

[5] A «modelicaPackage» can only specialize other classifiers derived from «modelicaPackage». The stereotype «modelicaExtends» must be applied to the generalization relationship.

[6] All other attributes or associations inherited from «block» or Classifier may not be used. This includes the attributes: isActive, isEncapsulated; and the ownedElements: Behavior, Constraint.

1.9 «modelicaFunction»

Extensions

- FunctionBehavior (from UML4SysML)

Generalizations

- «modelicaClassDefinition» (from SysML4Modelica::Classes)

Abstract Syntax

- See Figure 2.

Description

The Modelica specialized class `function` represents a callable section of procedural algorithmic code. It is similar to a SysML FunctionBehavior. Compared to a general Modelica `class`, quite a few restrictions and enhancements apply; refer to the Modelica Spec. 3.1, Section 12.2 for details.

As described in the Modelica Spec. 3.1, Section 12.9, a Modelica `function` may refer to an external function specifier (e.g., an external C or Fortran function):

```
function IDENT string_comment
{ component_clause ";" }
[ protected { component_clause ";" } ]
```

```
external [ language_specification ] [ external_function_call ]
[annotation ] ";"
[ annotation ";" ]
end IDENT;
```

Several additional attributes are included in «modelicaFunction» to capture such semantics.

At this point, SysML4Modelica only allows for function definitions; functions cannot be "called" explicitly – they can only be referred to in opaque Modelica syntax portions of the model.

Attributes

- isExternal: Boolean [1]
  Indicates whether the opaque body of the FunctionBehavior should be considered or whether the external function definition should be linked. The isExternal attribute is true when the `external` keyword is present in Modelica; false otherwise. Default value is *false*.

- externalLanguage: String [0..1]
  The language in which the external function specifier is defined. It should only be defined when *isExternal = true*. The externalLanguage attribute has the value of the string (without quotes) following the `external` keyword in Modelica. Default value is "C".

- externalFunctionSpecification: String [0..1]
  The complete specification of the externally defined function (see Modelica Spec. 3.1, Section 12.9 for details). If is not defined, then the name of the «modelicaFunction» is used. It should only be defined when *isExternal = true*. Default value is *null*.

- externalAnnotation: String [0..1]
  String containing a Modelica annotation associated with the external function call. It should only be defined when *isExternal = true*. Default value is *null*.

Associations

No additional associations.

Constraints

[1] A «modelicaFunction» must have a Name.
[2] A «modelicaFunction» can only have Parameters that are stereotyped to «modelicaFunctionParameter».
[3] Any «modelicaFunctionParameter» (owned by an instance of «modelicaPackage») for which *causality=input* may not be assigned values in the body of the function (i.e., it is read-only).
[4] A «modelicaFunction» can only have zero or one *body* attribute.
[5] If *isExternal=false* then *externalLanguage, externalFunctionSpecification,* and *externalAnnotation* must not be defined.
[6] If *isExternal=true* then *body* and *language* must not be defined.
[7] A «modelicaFunction» must have *language="Modelica"* if the *body* attribute is defined.
[8] The *body* of the function must be represented in the Modelica syntax and must constitute a valid Modelica algorithm section.
[9] A «modelicaFunction» can only be contained in a «modelicaClassDefinition».
[10] A «modelicaFunction» can only specialize other classifiers derived from «modelicaFunction». The stereotype «modelicaExtends» must be applied to the generalization relationship.
[11] All other attributes or associations inherited from FunctionBehavior or Classifier may not be used.

1.10 «modelicaExtends»

Extensions

- Generalization (from UML4SysML)

Abstract Syntax

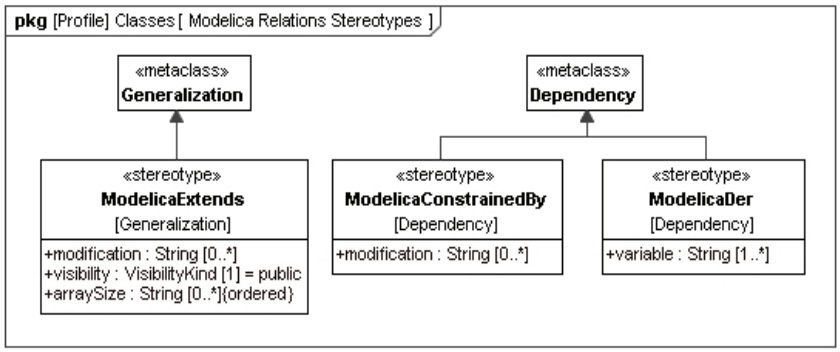


Figure 3: Modelica Relations stereotype definitions

Description

The `extends` clause of Modelica is equivalent to a SysML Generalization. The only difference is that in Modelica the type being extended can be locally modified (Modelica Spec. 3.1, Section 7.1):

```
extends_clause :
   extends name [ class_modification ] [annotation]

constraining_clause :
   extends name [ class_modification ]
```

Similar local type modifications can be used when defining usages (i.e., Modelica components – see Chapter 3). In both cases the SysML4Modelica mapping currently captures the local modifications only as a text string in Modelica syntax. A separate modification can be defined for every component of a «modelicaClassDefinition»; in Modelica these modifications are grouped, separated by commas, and surrounded by parentheses. Each such modification is represented in SysML4Modelica as a separate string. It corresponds thus to an argument as defined in the following extract of the Modelica EBNF (Modelica Spec. 3.1, section 7.2):

```
class_modification :
    "(" [ argument_list ] ")"

argument_list :
    argument { "," argument }

argument :
    element_modification_or_replaceable
  | element_redeclaration

element_modification_or_replaceable:
    [ each ] [ final ] ( element_modification | element_replaceable)

element_modification :
    component_reference [ modification ] string_comment

element_redeclaration :
    redeclare [ each ] [ final ]
   ( ( class_definition | component_clause1) | element_replaceable )

element_replaceable:
    replaceable ( class_definition | component_clause1)
        [constraining_clause]

component_clause1 :
    type_prefix type_specifier component_declaration1
```

```
component_declaration1 :
    declaration comment
```

Multiple inheritance is supported in Modelica. Therefore, more than one «modelicaExtends» relationship is allowed for a single «modelicaClassDefinition». The `extends` clause can be applied to any of the restricted classes (including packages).

If the extends clause appears in a protected section of the Modelica model, then all the elements of the base class become protected elements of the specialized class. It is therefore important to specify whether the «modelicaExtends» relation is *public* or *protected*.

Not every restricted class can inherit from every other restricted class. Refer to Modelica Spec. 3.1, Section 7.1.3 for an overview table.

Attributes

- visibility: VisibilityKind [1]
  When an extends statement appears in a protected section of a «modelicaClassDefinition», then all components of the parent class are protected. Default value is *public*.

- modification: String [0..*]
  An inherited Modelica class can be locally modified. The modifications are defined by this attribute in Modelica syntax. Each modification (as specified in the Modelica concrete syntax as a comma-separated expression) is specified as a separate instance of this attribute. Default value is *null*.

- arraySize: String [0..*] {ordered}
  One can specify an array size for an inherited Modelica class. This attribute is an ordered list of strings, each of which must be a Modelica expressions that evaluates to an integer. The i^{th} element in the ordered list corresponds to size of the the multi-dimensional array in the i^{th} dimension. The default value is *null*.

Associations

No additional associations.

Constraints

[1] Both the *source* and *target* of a «modelicaExtends» relation must be typed to instances of a specialization of «modelicaClassDefinition».

[2] The *visibility* attribute of «modelicaExtends» can only take on values of *public* or *protected*.

1.11 «modelicaDer»

Extensions

- Dependency (from UML4SysML)

Abstract Syntax

- See Figure 3.

Description

The `der` clause in Modelica identifies a function as a partial derivative of another function (Modelica Spec. 3.1, Section 12.7.2). It establishes a relationship between two functions and is therefore modeled as an extension of Dependency in SysML4Modelica. It requires as attributes a list of variables with respect to which the partial derivative is taken.

Attributes

- variable: String [1..*]
  A list of variables with respect to which the partial derivative is taken. At least one variable must be specified. No default value is specified.

Associations

No additional associations.

Page : 12 Line : 331 Author : Chris Paredis 03/15/2010
Should this be [0..1]?

Constraints

[1] Both the *source* and *target* of a «modelicaDer» relation must be typed to instances of «modelicaFunction».

1.12 «modelicaConstrainedBy»

Extensions

- Dependency (from UML4SysML)

Abstract Syntax

- See Figure 3.

Description

In a replaceable declaration in Modelica, one can specify a `constrainedBy` clause. The semantics of this construct are explained in more detail in the Modelica Spec. 3.1, Section 7.3.2.

Attributes

- modification: String [0..*]
  A Modelica class that constrains a replaceable declaration can be locally modified. The modifications are defined by this attribute in Modelica syntax. Each modification (as specified in the Modelica concrete syntax as a comma-separated expression) is specified as a separate instance of this attribute. Default value is *null*.

Associations

No additional associations.

Constraints

[1] Both the *source* and *target* of a «modelicaConstrainedBy» relation must be typed to instances of a specialization of «modelicaClassDefinition».

1.13 Short Class Definitions

Modelica provides a short-hand notation for definition of classes. It is equivalent to an inheritance construct, and is therefore redundant and not supported separately in the SysML4Modelica profile.

1.14 Illustrative Examples

A Modelica model for a translational Mass is defined as a specialization of PartialRigid, as is illustrated in Figure 4. (Model extracted from the Modelica Standard Library – Modelica.Mechanics.Translational.Components.Mass). The corresponding Modelica model is shown below.

Page : 13 Line : 366 Author : Chris Paredis 03/10/2010
Do we need this section?

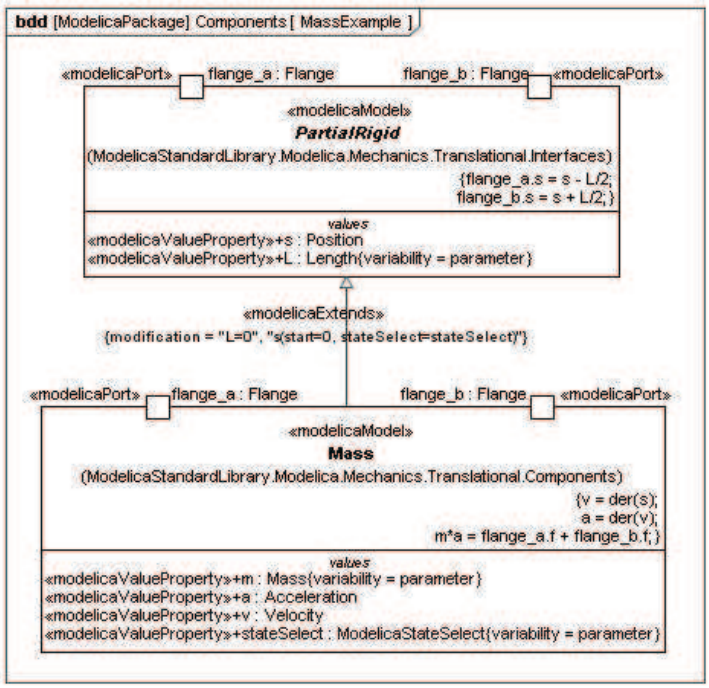


Figure 4: SysML4Modelica model for translational mass

The corresponding Modelica models (with annotations omitted):

```
model Mass "Sliding mass with inertia"
  parameter SI.Mass m(min=0, start=1) "mass of the sliding mass";
  parameter StateSelect stateSelect=StateSelect.default
    "Priority to use s and v as states";
  extends Translational.Interfaces.PartialRigid(L=0,s(start=0,
stateSelect=stateSelect));
  SI.Velocity v(start=0, stateSelect=stateSelect)
    "absolute velocity of component";
  SI.Acceleration a(start=0) "absolute acceleration of component";
equation
  v = der(s);
  a = der(v);
  m*a = flange_a.f + flange_b.f;
end Mass;

partial model Modelica.Mechanics.Translational.Interfaces.PartialRigid
  "Rigid connection of two translational 1D flanges "
  SI.Position s "Absolute position of center of component
                 (s = flange_a.s + L/2 = flange_b.s - L/2)";
  parameter SI.Length L(start=0) "Length of component, from left flange to
                                  right flange (= flange_b.s - flange_a.s)";
  Flange_a flange_a "Left flange of translational component";
  Flange_b flange_b "Right flange of translational component";
equation
  flange_a.s = s - L/2;
  flange_b.s = s + L/2;
end PartialRigid;
```

Remarks:

- The name of each model (Mass and PartialRigid) appear as the Name attribute of the corresponding «mod-

elicaModel»

- The parameters of each model (m and stateSelect) appear as «modelicaValueProperty» in «modelicaModel»
- The connectors flange_a and flange_b appear as ports stereotyped to «modelicaPort»
- The equation sections are each mapped to a «modelicaEquation», which are shown as any other constraint in the concrete graphical syntax

2 Predefined Types

2.1 Overview

The following predefined types are available in the Modelica language (Modelica Spec. 3.1, Section 4.8): Real Type, Integer Type, Boolean Type, String Type, Enumeration Types, StateSelect, ExternalObject, and Graphical Annotation Type (See Chapter 5). These primitive types are defined as predefined types in SysML4Modelica::Types::ModelicaPredefinedTypes. Although these types have direct counterparts in SysML, they are redefined to account for the additional attributes associated with them in Modelica. Note that in Modelica, the properties such as "start", "quantity", etc., are not really equivalent to user-defined complex data-types. For instance, if one defines "Real x;" then one cannot refer to "x.min" in an equation. The only way one can specify a value for these special properties is as part of a type definition or local modification: e.g., "Real x(start=1, unit="m");

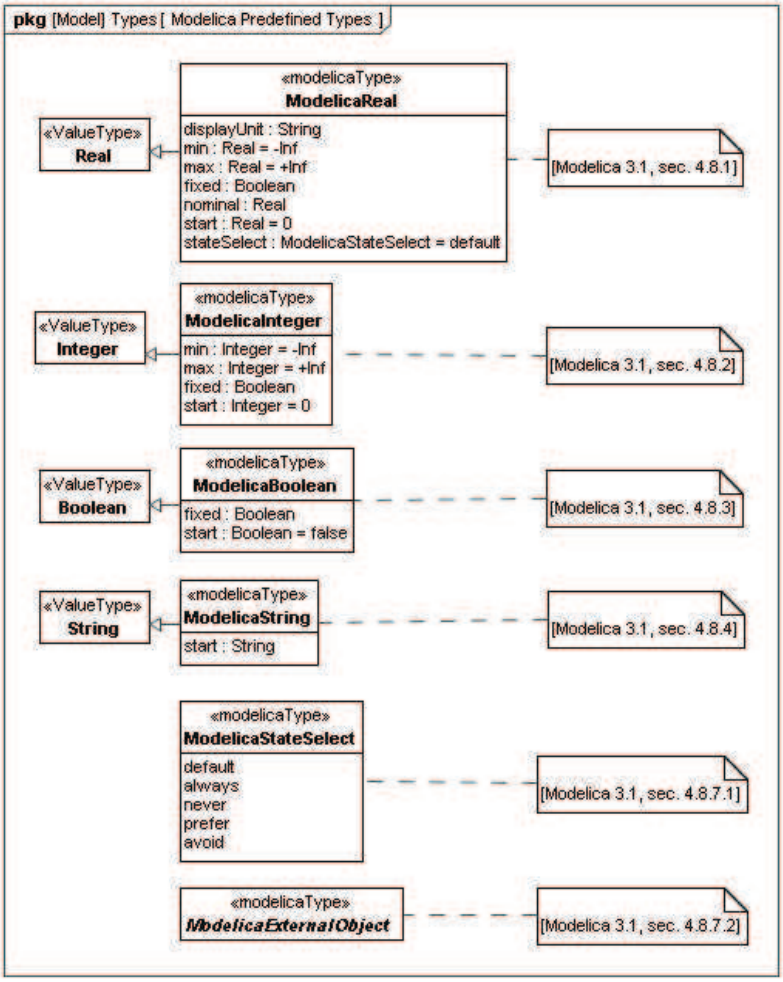


Figure 5: Package diagram with an overview of the Predefined Modelica Types

2.2 ModelicaReal

Instantiation

- SysML4Modelica::Classes::ModelicaType

Generalizations

- SysML::Blocks::Real

Abstract Syntax

- See Figure 5.

Description

The predefined type `Real` in Modelica includes a variety of attributes besides its actual value (Modelica 3.1, section 4.8.1). In SysML4Modelica, these attributes are defined in ModelicaReal, a specialization of the primitive type SysML::Blocks::Real. As a result of this specialization, ModelicaReal, inherits the attributes: quantityKind and unit, which correspond to the Modelica attributes `quantity` and `unit`, respectively. Additional attributes are listed below.

Attributes

- displayUnit: String [0..1]
  In addition to the actual units, a ModelicaReal can have a units used for display in a tool's graphical user interface or in plots. These units are defined in this attribute as a string. Default value is *null*.

- min: Real [1]
  The minimum value the ModelicaReal variable can take on. Default value is *-Inf*.

- max: Real [1]
  The maximum value the ModelicaReal variable can take on. Default value is *+Inf*.

- start: Real [1]
  The value of the ModelicaReal variable at the beginning of a simulation. The meaning of this variable depends on the value of the attribute `fixed`. If `fixed=false`, then it is to be interpreted as an initial guess from which may be deviated in order to satisfy all the algebraic constraints. If `fixed=true`, then the variable is required to equal its start value. Default value is *0*.

- fixed: Boolean [1]
  This attribute qualifies the meaning of the attribute `start`. If `fixed=false`, then `start` is to be interpreted as an initial guess from which may be deviated in order to satisfy all the algebraic constraints. If `fixed=true`, then the variable is required to equal its `start` value. Default value is *true* for parameters and constants, and *false* for all other variables.

- nominal: Real [0..1]
  The value of this attribute may be used by the solver for scaling purposes. Default value is *null*.

- stateSelect: StateSelect [1]
  The value of this attribute determines how a Modelica solver should select state variables for the system of Differential Algebraic Equations (Modelica Spec. 3.1, Section 4.8.7.1). Default value is *StateSelect.default*.

Associations

No additional associations.

Constraints

No additional associations.

2.3 ModelicaInteger

Instantiation
- SysML4Modelica::Classes::ModelicaType

Generalizations
- SysML::Blocks::Integer

Abstract Syntax
- See Figure 5.

Description

The predefined type `Integer` in Modelica includes a variety of attributes besides its actual value (Modelica Spec. 3.1, Section 4.8.2). In SysML4Modelica, these attributes are defined in ModelicaInteger, a specialization of the primitive type SysML::Blocks::Integer. As a result of this specialization, ModelicaInteger, inherits the attribute: quantityKind, which correspond to the Modelica attribute `quantity`. Additional attributes are listed below.

Attributes
- min: Integer [1]
  The minimum value the ModelicaInteger variable can take on. Default value is -Inf.

- max: Integer [1]
  The maximum value the ModelicaInteger variable can take on. Default value is +Inf.

- start: Integer [1]
  The value of the ModelicaInteger variable at the beginning of a simulation. The meaning of this variable depends on the value of the attribute `fixed`. If `fixed=false`, then it is to be interpreted as an initial guess from which may be deviated in order to satisfy all the algebraic constraints. If `fixed=true`, then the variable is required to equal its start value. Default value is *0*.

- fixed: Boolean [1]
  This attribute qualifies the meaning of the attribute `start`. If `fixed=false`, then `start` is to be interpreted as an initial guess from which may be deviated in order to satisfy all the algebraic constraints. If `fixed=true`, then the variable is required to equal its `start` value. Default value is *true* for parameters and constants, and *false* for all other variables.

Associations

No additional associations.

Constraints

No additional associations.

2.4 ModelicaBoolean

Instantiation
- SysML4Modelica::Classes::ModelicaType

Generalizations
- SysML::Blocks::Boolean

Abstract Syntax
- See Figure 5.

Description

The predefined type `Boolean` in Modelica includes a variety of attributes besides its actual value (Modelica 3.1, section 4.8.3). In SysML4Modelica, these attributes are defined in ModelicaBoolean, a specialization of the primitive type SysML::Blocks::Boolean. As a result of this specialization, ModelicaBoolean, inherits the attribute, quant-

ityKind, which correspond to the Modelica attribute `quantity`. Additional attributes are listed below.

Attributes

- start: Boolean [1]
  The value of the ModelicaBoolean variable at the beginning of a simulation. The meaning of this variable depends on the value of the attribute `fixed`. If `fixed=false`, then it is to be interpreted as an initial guess from which may be deviated in order to satisfy all the algebraic constraints. If `fixed=true`, then the variable is required to equal its start value. Default value is *false*.

- fixed: Boolean [1]
  This attribute qualifies the meaning of the attribute `start`. If `fixed=false`, then `start` is to be interpreted as an initial guess from which may be deviated in order to satisfy all the algebraic constraints. If `fixed=true`, then the variable is required to equal its `start` value. Default value is *true* for parameters and constants, and *false* for all other variables.

Associations

No additional associations.

Constraints

No additional associations.

2.5 ModelicaString

Instantiation

- SysML4Modelica::Classes::ModelicaType

Generalizations

- SysML::Blocks::String

Abstract Syntax

- See Figure 5.

Description

The predefined type `String` in Modelica includes a variety of attributes besides its actual value (Modelica 3.1, section 4.8.4). In SysML4Modelica, these attributes are defined in ModelicaString, a specialization of the primitive type SysML::Blocks::String. As a result of this specialization, ModelicaString inherits the attribute, quantityKind, which correspond to the Modelica attributes `quantity`. In addition, a start value can be specified.

Attributes

- start: String [1]
  The value of the ModelicaReal variable at the beginning of a simulation. The meaning of this variable depends on the value of the attribute `fixed`. If `fixed=false`, then it is to be interpreted as an initial guess from which may be deviated in order to satisfy all the algebraic constraints. If `fixed=true`, then the variable is required to equal its start value. Default value is *0*.

Associations

No additional associations.

Constraints

No additional associations.

2.6 ModelicaStateSelect

Instantiation

- SysML4Modelica::Classes::ModelicaType

Generalizations

No generalizations.

Abstract Syntax

- See Figure 5.

Description

The predefined type `ModelicaStateSelect` is the type of the attribute stateSelect of ModelicaReal. It is an enumeration used to provide guidance to the Modelica solver tool for selecting appropriate state variables (See Modelica Spec. 3.1, section 4.8.7.1).

Associations

No additional associations.

Constraints

No additional associations.

2.7 ModelicaExternalObject

Instantiation

- SysML4Modelica::Classes::ModelicaType

Generalizations

No generalizations.

Abstract Syntax

- See Figure 5.

Description

The predefined type `ModelicaExternalObject` is an abstract type used to indicate that a ModelicaType that specializes it refers to an object defined in an external language such as C or FORTRAN (See Modelica Spec. 3.1, section 12.9.7 for details).

Associations

No additional associations.

Constraints

[1] The value of the attribute *isAbstract* (and hence *isPartial*) must be *true*.

3 Component Declarations

3.1 Overview

In the Modelica language, instances (or usages) of a class are referred to as “Components”. In SysML, these can be mapped to Block Properties, such as Value Property, Part Property, or Port.[1] Modelica does not distinguish explicitly between Value Properties, Parts, or Ports. Instead, whether a component is interpreted as a Value Property, Part or Port depends on the restricted type to which the usage has been typed. If the usage is of restricted type `class`, `model`, or `block` then it is mapped to a «modelicaPart»; if it is of restricted type `connector` then it is mapped to a «modelicaPort»; and if it is of restricted type `record` or `type` then it is mapped to «modelicaValueProperty». In addition, the stereotype «modelicaFunctionParameter» is introduced to represent components of restricted type `record` or `type` that are used in a `function` (this is necessary because a Modelica function is mapped to a SysML FunctionBehavior which has parameters rather than properties). The restricted types `package` and `function` are

[1] Note that Modelica does not have the equivalent of a reference property — properties are never shared.

not considered here because they cannot be instantiated.

Depending on the type of restricted type, a Modelica Component declaration allows for a variety of options (modifications or additional specifications). These additional options are captured as attributes of the corresponding SysML4Modelica stereotypes, as show in Figure 6. To define the possible values these options can assume, several enumerations are defined, as shown in Figure 7. The following production rules define Modelica Components declarations:

```
component_clause:
   type_prefix type_specifier [ array_subscripts ] component_list

type_prefix :
   [ flow ]
   [ discrete | parameter | constant ] [ input | output ]

type_specifier :
   name

component_list :
   component_declaration { "," component_declaration }

component_declaration :
   declaration [ conditional_attribute ] comment

conditional_attribute:
   if expression

declaration :
   IDENT [ array_subscripts ] [ modification ]
```

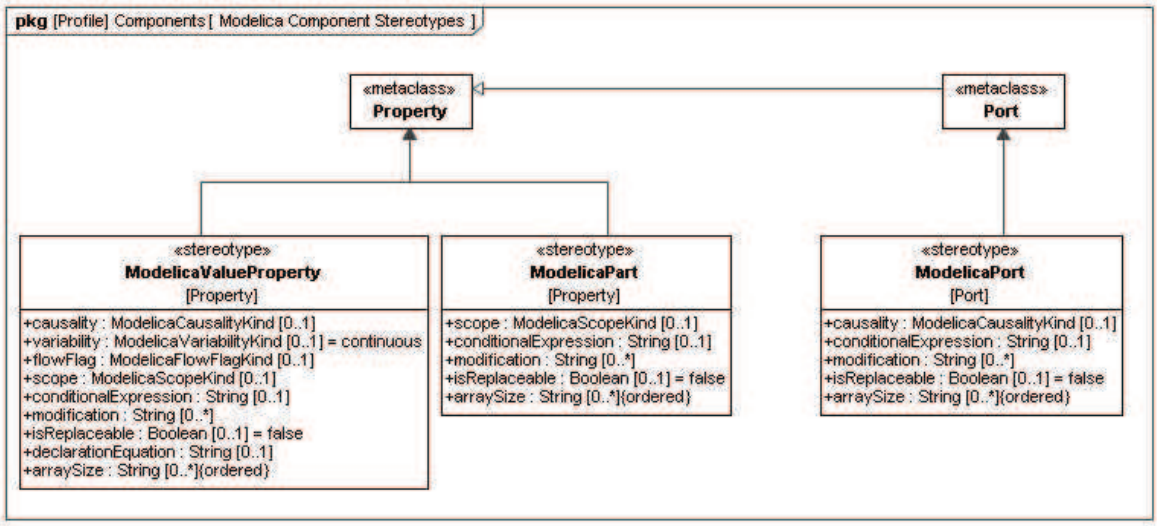


Figure 6: Package diagram with an overview of the stereotypes for Modelica Components

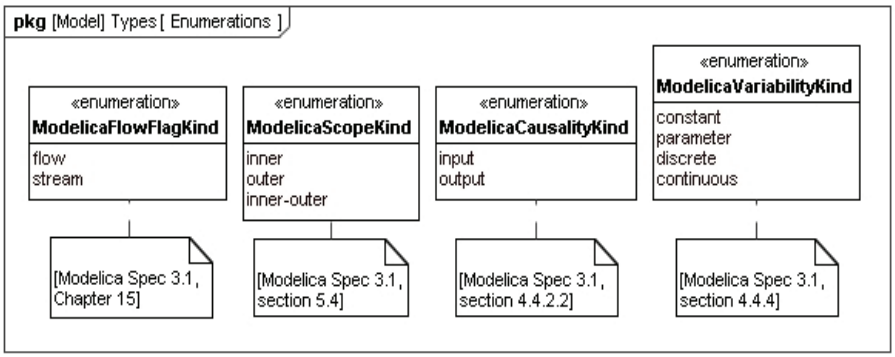


Figure 7: Package diagram with enumerations used in Modelica Component definitions

Table 2:The applicable attributes for Modelica Components.

Attribute Name	«modelicaValueProperty»	«modelicaPart»	«modelicaPort»
visibility	•	•	
causality	•		•
variability	•		
flowFlag	•		
scope	•	•	
conditionalExpression	•	•	•
isFinal	•	•	•
modification	•	•	•
isReplaceable	•	•	•
declarationEquation	•		
arraySize	•	•	•

3.2 «modelicaValueProperty»

Extensions

- Property (from UML4SysML)

Abstract Syntax

- See Figure 6.

Description

If a Modelica Component is of restricted type `record` or `type` then it is mapped to a «modelicaValueProperty», which is the equivalent of a Value Property in SysML.

Attributes

- visibility: VisibilityKind [1]
  This attribute is inherited from the meta-class Property. In the context of the SysML4Modelica profile, it is limited to the values *public* or *protected*. A protected «modelicaValueProperty» cannot be modified or replaced in specializations or modifications. The members of a protected «modelicaValueProperty» cannot be accessed using the dot-notation. Default value is *public*.

- causality: ModelicaCausalityKind [0..1]
  A «modelicaValueProperty» can be defined as being an input or output (Modelica Spec. 3.1, Section 4.4.2.2). Default value is *null*, which means that the property is neither an input or output.

- variability: ModelicaVariabilityKind [1]
  A «modelicaValueProperty» can be defined as being constant, parameter, discrete or continuous (Modelica Spec. 3.1, section 4.4.3 and 4.4.4). Default value is *continuous*.

- flowFlag: ModelicaFlowFlagKind [0..1]
  This attribute can only be applied to variables that are a subtype of ModelicaReal. It can only be used inside «modelicaConnector» or to define a Type. The attribute *causality* must be *null* when *flowFlag=flow* or *stream*. Default value is *null*.

- scope: ModelicaScopeKind [0..1]
  A Modelica element declared with the prefix `outer` references an element instance with the same name but using the prefix `inner`, which is nearest in the enclosing instance hierarchy of the outer element declaration (Modelica Spec. 3.1, Section 5.4). Default value is *null*.

- conditionalExpression: String [0..1]
  When defined, this attribute contains an expression in Modelica syntax that must evaluate to *true* or *false*. Only if the expression evaluates to true is the the corresponding «modelicaValueProperty» instantiated (Modelica Spec. 3.1, Section 4.4.5). Default value is *null*.

- modification: String [0..*]
  A «modelicaValueProperty» may have a type that is locally modified. Rather than capturing the detailed semantics of such modifications in the SysML4Modelica profile, currently, the modifications are only captured as a set of strings in the Modelica syntax; each string corresponds to a single modification of a component declaration of the modified class (Modelica Spec. 3.1, Section 7.2). Default value is *null*.

- isReplaceable: Boolean [0..1]
  A «modelicaValueProperty» may be defined as `replaceable`. One can then `redeclare` such a «modelicaValueProperty» in extended classes or in modifications (Modelica Spec. 3.1, Section 7.3). Default value is *false*.

- declarationEquation: String [0..1]
  When defined, this attribute contains an expression in Modelica syntax that must evaluate to the same type as the «modelicaValueProperty» itself. A declaration equation refers to the shorthand notation in Modelica in which an equation corresponding to a component is defined in the equation section. The value of the attribute is the right-hand-expression of the equations. The "=" sign is omitted, i.e., it is implicit. Default value is *null*.

- isFinal: Boolean [0..1]
  A Modelica element declared with the prefix `final` cannot be modified in redeclarations or modifications (Modelica Spec. 3.1, Section 7.2.6). Default value is *null*.

- arraySize: String [0..*] {ordered}
  This attribute is an ordered list of strings, each of which must be a Modelica expressions that evaluates to an integer. The i^{th} element in the ordered list corresponds to size of the the multi-dimensional array in the i^{th} dimension. The default value is *null*.

Associations

No additional associations.

Constraints

No additional constraints.

3.3 «modelicaPart»

Extensions

- Property (from UML4SysML)

Abstract Syntax

- See Figure 6.

Description

If a Modelica Component is of restricted type `class`, `model`, or `block`, it is mapped to a «modelicaPart», which is the equivalent of a Part Property in SysML.

Attributes

- visibility: VisibilityKind [1]
  This attribute is inherited from the meta-class Property. In the context of the SysML4Modelica profile, it is limited to the values *public* or *protected*. A protected «modelicaPart» cannot be modified or replaced in specializations or modifications. The members of a protected «modelicaPart» cannot be accessed using the dot-notation. Default value is *public*.

- scope: ModelicaScopeKind [0..1]
  A Modelica element declared with the prefix `outer` references an element instance with the same name but using the prefix `inner`, which is nearest in the enclosing instance hierarchy of the outer element declaration (Modelica Spec. 3.1, Section 5.4). Default value is *null*.

- conditionalExpression: String [0..1]
  When defined, this attribute contains an expression in Modelica syntax that must evaluate to *true* or *false*. Only if the expression evaluates to true is the the corresponding «modelicaPart» instantiated (Modelica Spec. 3.1, Section 4.4.5). Default value is *null*.

- modification: String [0..*]
  A «modelicaPart» may have a type that is locally modified. Rather than capturing the detailed semantics of such modifications in the SysML4Modelica profile, currently, the modifications are only captured as a set of strings in the Modelica syntax; each string corresponds to a single modification of a component declaration of the modified class (Modelica Spec. 3.1, Section 7.2). Default value is *null*.

- isReplaceable: Boolean [0..1]
  A «modelicaPart» may be defined as `replaceable`. One can then `redeclare` such a «modelicaPart» in extended classes or in modifications (Modelica Spec. 3.1, Section 7.3). Default value is *false*.

- isFinal: Boolean [0..1]
  A Modelica element declared with the prefix `final` cannot be modified in redeclarations or modifications (Modelica Spec. 3.1, Section 7.2.6). Default value is *null*.

- arraySize: String [0..*] {ordered}
  This attribute is an ordered list of strings, each of which must be a Modelica expressions that evaluates to an integer. The i^{th} element in the ordered list corresponds to size of the the multi-dimensional array in the i^{th} dimension. The default value is *null*.

Associations

No additional associations.

Constraints

No additional constraints.

3.4 «modelicaPort»

Extensions

- Port (from UML4SysML)

Abstract Syntax

- See Figure 6.

Description

If a Modelica Component is of restricted type `connector`, it is mapped to a «modelicaPort», which is the equivalent of a Port Property in SysML.

Attributes

- causality: ModelicaCausalityKind [0..1]
  A «modelicaPort» can be defined as being an input or output (Modelica Spec. 3.1, Section 4.4.2.2). Default value is *null*, which means that the property is neither an input or output.

- scope: ModelicaScopeKind [0..1]
  A Modelica element declared with the prefix `outer` references an element instance with the same name but using the prefix `inner`, which is nearest in the enclosing instance hierarchy of the outer element declaration (Modelica Spec. 3.1, Section 5.4). Default value is *null*.

- conditionalExpression: String [0..1]
  When defined, this attribute contains an expression in Modelica syntax that must evaluate to *true* or *false*. Only if the expression evaluates to true is the the corresponding «modelicaPort» instantiated (Modelica Spec. 3.1, Section 4.4.5). Default value is *null*.

- isFinal: Boolean [0..1]
  A Modelica element declared with the prefix `final` cannot be modified in redeclarations or modifications (Modelica Spec. 3.1, Section 7.2.6). Default value is *null*.

- modification: String [0..*]
  A «modelicaPort» may have a type that is locally modified. Rather than capturing the detailed semantics of such modifications in the SysML4Modelica profile, currently, the modifications are only captured as a set of strings in the Modelica syntax; each string corresponds to a single modification of a component declaration of the modified class (Modelica Spec. 3.1, Section 7.2). Default value is *null*.

- isReplaceable: Boolean [0..1]
  A «modelicaPort» may be defined as `replaceable`. One can then `redeclare` such a «modelicaPort» in extended classes or in modifications (Modelica Spec. 3.1, Section 7.3). Default value is *false*.

- arraySize: String [0..*] {ordered}
  This attribute is an ordered list of strings, each of which must be a Modelica expressions that evaluates to an integer. The i^{th} element in the ordered list corresponds to size of the the multi-dimensional array in the i^{th} dimension. The default value is *null*.

Associations

No additional associations.

Constraints

No additional constraints.

3.5 «modelicaFunctionParameter»

Extensions

- Parameter (from UML4SysML)

Abstract Syntax

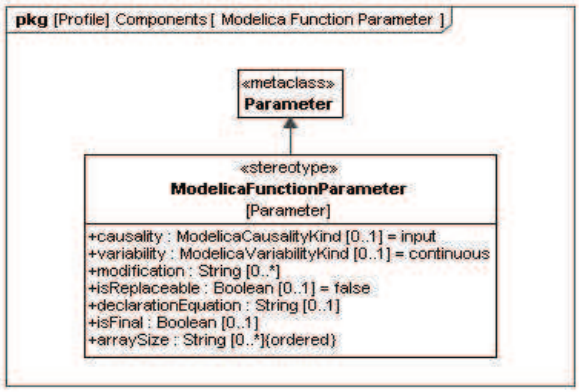


Figure 8: Definition of the «modelicaFunctionParameter» stereotype

Description

A Modelica restricted class function, can also contain can contain Modelica component declarations. These delcarations must be of either restricted type «modelicaType» or «modelicaRecord». Because «modelicaFunction» does not derive from «block» (as all the other restricted classes do), the stereotype «modelicaValueProperty» cannot be applied here. Instead, an equivalent (but more restricted) stereotype for functions is created: «modelicaFunctionParameter».

Attributes

- causality: ModelicaCausalityKind [0..1]
  A «modelicaFunctionParameter» can be defined as being an input or output (Modelica Spec. 3.1, Section 4.4.2.2). Default value is *null*, which means that the parameter is neither an input or output.

- isFinal: Boolean [0..1]
  A Modelica element declared with the prefix `final` cannot be modified in redeclarations or modifications (Modelica Spec. 3.1, Section 7.2.6). Default value is *null*.

- modification: String [0..*]
  A «modelicaFunctionParameter» may have a type that is locally modified. Rather than capturing the detailed semantics of such modifications in the SysML4Modelica profile, currently, the modifications are only captured as a set of strings in the Modelica syntax; each string corresponds to a single modification of a component declaration of the modified class (Modelica Spec. 3.1, Section 7.2). Default value is *null*.

- isReplaceable: Boolean [0..1]
  A «modelicaFunctionParameter» may be defined as `replaceable`. One can then `redeclare` such a «modelicaPort» in extended classes or in modifications (Modelica Spec. 3.1, Section 7.3). Default value is *false*.

- declarationEquation: String [0..1]
  When defined, this attribute contains an expression in Modelica syntax that must evaluate to the same type as the «modelicaFunctionParameter» itself. A declaration equation refers to the shorthand notation in Modelica in which an equation corresponding to a component is defined in the equation section. The value of the attribute is the right-hand-expression of the equations. The ":=" sign is omitted, i.e., it is implicit. Default value is *null*.

- isFinal: Boolean [0..1]
  A Modelica element declared with the prefix `final` cannot be modified in redeclarations or modifications (Modelica Spec. 3.1, Section 7.2.6). Default value is *null*.

- arraySize: String [0..*] {ordered}
  This attribute is an ordered list of strings, each of which must be a Modelica expressions that evaluates to an integer. The i^{th} element in the ordered list corresponds to size of the the multi-dimensional array in the i^{th} dimension. The default value is *null*.

Associations

No additional associations.

Constraints

No additional constraints.

4 Equation and Algorithm Sections

4.1 Overview

Equations and Algorithms are the main Modelica constructs for defining behavior of Modelica classes. Modelica distinguishes between declarative equations, which are organized in `equation` sections (Modelica Spec. 3.1, Chapter 8), and imperative algorithms, which are organized in `algorithm` sections (Modelica Spec. 3.1, Chapter 11). The Modelica restricted classes, `class`, `model`, and `block`, can each have zero or more equation and algorithm sections. Modelica `functions` can only have one single algorithm sections (and no equations).

The equations and expressions in equation and algorithm sections are enforced by the solver in every time step --- they must hold at every moment in time. In addition, one can specify equations or expressions that only need do hold at the start of the simulation; they are organized in `initial equation` and `initial algorithm` sections.

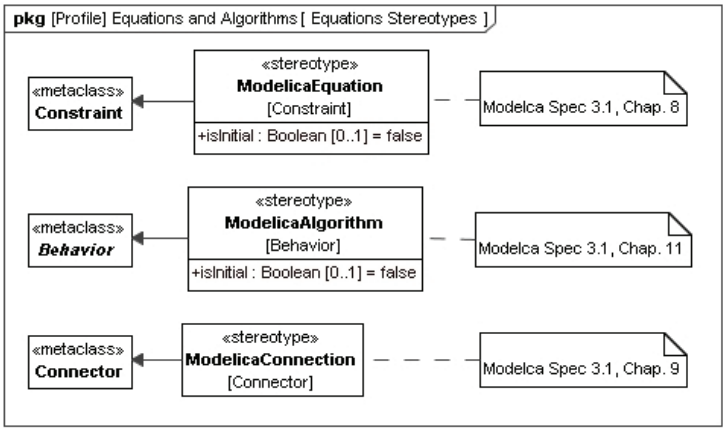


Figure 9: Package diagram with Equation and Algorithm definitions

4.2 «modelicaEquation»

Extensions

- Constraint (from UML4SysML)

Abstract Syntax

- See Figure 9.

Description

Modelica `equation` sections contain declarative equations that must hold at every moment in time. Each model (of restricted class types `class`, `model` or `block`) may contain zero or more equation sections. Given that the equations in these equation sections are declarative, they could be combined into a single section (note: the order in

which declarative equations are defined does not matter). However, the SysML4Modelica mapping allows for each equation section to be modeled by a separate «modelicaEquation».

Modelica `equation` sections may also contain `connect` statements (Modelica Spec., Chapter 9). Although `connect` statements are treated just like other equations in Modelica, they require special attention in SysML4-Modelica. Refer to Section 4.4 from details on «modelicaConnection»s.

Attributes

- isInitial: Boolean [0..1]
  This attribute is *true* when the «modelicaEquation» represents an `initial equation` section in Modelica. The default value is *false*.

Associations

No additional associations

Constraints

No additional constraints

4.3 «modelicaAlgorithm»

Extensions

- Behavior (from UML4SysML)

Abstract Syntax

- See Figure 9.

Description

Modelica `algorithm` sections contain imperative statements that are executed at every moment in time. Each model (of restricted class types `class`, `model` or `block`) may contain zero or more algorithm sections. In addition, a `function` contains at most one algorithm section. Each algorithm section is modeled by a separate «modelicaAlgorithm». To capture the imperative nature of algorithm sections, a «modelicaAlgorithm» extends UML4SysML::Behavior. Only opaque behaviors are currently supported and the algorithm statements are expressed in Modelica syntax in the *Body* of the «modelicaAlgorithm».

Attributes

- isInitial: Boolean [0..1]
  This attribute is *true* when the «modelicaAlgorithm» represents an `initial algorithm` section in Modelica. The default value is *false*.

Associations

No additional associations

Constraints

No additional constraints

4.4 «modelicaConnection»

Extensions

- Connector (from UML4SysML)

Abstract Syntax

- See Figure 9.

Description

In Modelica, a `connection` between two ports typically has Kirchhoff semantics (i.e., across variables are equal, through variables sum to zero), or an output-to-input binding in the case of a signal connection (See Modelica Spec.

3.1, Chapter 9). To capture these same semantics succinctly, a «modelicaConnection» is used. The two arguments of the connect statement correspond to the two ends of the «modelicaConnection». Note that the use of a «modelicaConnection» is optional. The alternative is to represent the connection using a connect statement in Modelica syntax in a «modelicaEquation». If a «modelicaConnection» is used, then the corresponding connect statement must be removed from the «modelicaEquation».

As for all equations, Modelica allows `connect` statements to be used in a parametric fashion, for instance, inside a for loop. Since the parameter values are only resolved at the time of compilation of the Modelica model, a parametrically defined `connect` statement cannot be modeled explicitly in SysML4Modelica. The alternative is to represent such connect statements in Modelica syntax in a «modelicaEquation».

Attributes

No additional attributes

Associations

No additional associations

Constraints

[1] The start and end of a «modelicaConnection» must be a «modelicaPort».

5 Other Related Constructs

5.1 «modelicaSimulation»

Generalizations

- Block (from SysML)

Abstract Syntax

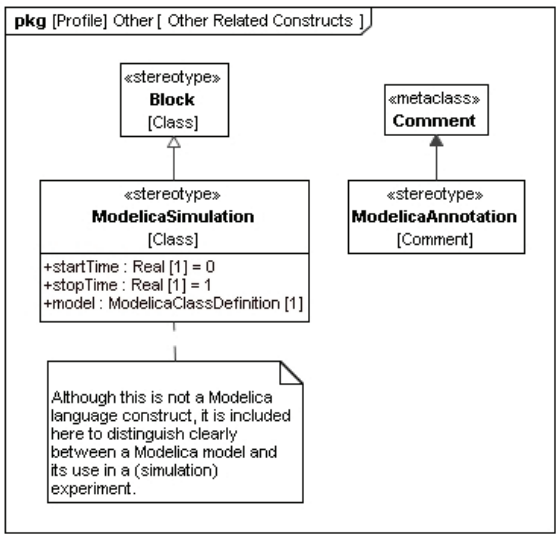


Figure 10: Package diagram with definitions of Modelica-related constructs

Description

A «modelicaSimulation» is not a Modelica language construct. However, it is introduced in order to distinguish between the model and its simulation. A simulation refers to the solution of the initial value problem: the integration of the model over a particular time period starting from a particular initial condition. Since the initial conditions are already defined in the model itself, the only additional information that needs to be provided is the time over which to integrate and the properties of the solver to be used.

Attributes

- startTime: Real [1]
  The time at which the simulation starts. Default value is *0.*

- stopTime: Real [1]
  The time at which the simulation stops. Default value is *1*.

- model: «modelicaClassDefinition» [1]
  The instance of a specialization of «modelicaClassDefinition» that is to be solved. Default value is *null*.

Associations

No additional associations.

Constraints
No additional constraints.

5.2 «modelicaAnnotation»

Extension

- Comment (from UML4SysML)

Abstract Syntax

- See Figure 9.

Description

Any Modelica language construct can be annotated with information about its graphical representation. In addition, guidelines for the compiler can be specified. In SysML4MModelica, these annotations are represented in Modelica syntax as «modelicaComment»s.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Part III Modelica Abstract Syntax

The abstract syntax (AST = abstract syntax tree) of Modelica is not standardized. The abstract syntax described in this document is one possible definition, defined in an extended subset of Modelica (also known as MetaModelica) and used in the OpenModelica specification/implementation of Modelica which originated as a Structural Operational Semantics/Natural Semantics specification, the first version from 1998.
However, even if the abstract syntax of Modelica is not standardized, given the structure of the language as described in this document, the difference in abstract syntax between the different tools are likely to be relatively small. Any difference in terminology or minor differences in structure can be handled with tool-specific transformations that will be performed on the ASTs.

The abstract syntax used in OpenModelica has been designed with several goals in mind:

- Complete representation of all Modelica language constructs.
- Reconstruction of the source code from the AST.
- Use for semantic specification, type checking, and compilation.

Syntax type classes are defined using the ***uniontype*** construct. A union type is the union of all the ***record*** types it contains. Recursive references to a union type are allowed. Components with optional values are declared at instances of the ***Option***<...> parametrized type constructor. In a few cases the ***tuple***<type1,type2,...> type constructor is used. A tuple type can be described as an anonymous record type, where the record type name and the field names are not defined.

In the following all MetaModelica classes (including a short textual description) are listed (version Oct.2009[1] from the OpenModelica SVN). This definition is translated into an OMG MOF-based description (see http://www.omg.org/mof/) using the Eclipse EMF (http://www.eclipse.org/emf/) implementation of a subset of the OMG MOF standard. **The .ecore and .ecorediag files are attached** to this document. Please see the files for details and diagrams.

The mapping between the MetaModelica and the Eclipse EMF (ecore) is defined as following:

- MetaModelica ***package*** is translated **E*Package***
- MetaModelica ***uniontype*** is translated to **E*Class*** (isAbstract)
- MetaModelica ***record*** is translated to **E*Class*** which inherits from the respective EClass that represents the uniontype)
- MetaModelica ***record attributes*** of primitive type is translated to EClass attributes of primitive type
- MetaModelica ***record attributes*** of composite type is translated to EClass EReference to the repsective EClass
- MetaModelica ***types*** are expanded and translated into ***EClasses***
- MetaModelica ***tuples*** are expanded and translated into ***EClasses*** with the prefix “tuple_”

[1] Note: all MetaModelica specific classes are removed.

– MetaModelica ***Option<...>*** implies the multiplicity 0..1

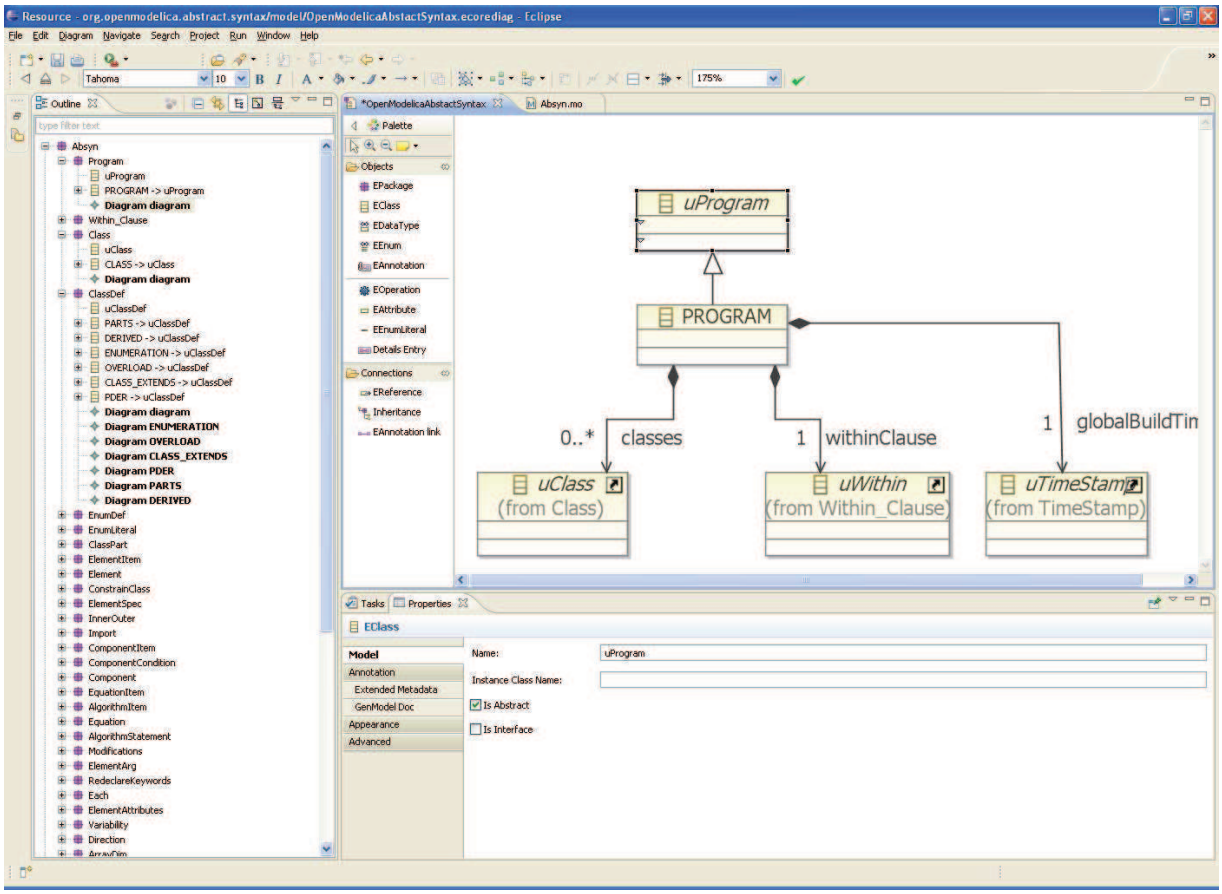
– MetaModelica ***list<...>*** implies the multiplicity 0..*

– MetaModelica `type Ident = String;` is not translated. EString is used directly.

– In order to avoid name clashes between ***EClasses*** representing ***uniontype*** or ***record*** each ***EClass*** that represents a ***uniontype*** has a prefix “u”.

– In order to improve the structure and readability for each MetaModelica ***uniontype*** an ***EPackage*** is created with the same name as the ***uniontype***. This ***EPackage*** includes the ***EClass*** representing the ***uniontype*** and ***EClasses*** representing the ***records*** of the ***uniontype***.

The following figure shows an example of the translation for the MetaModelica ***uniontype*** “program” (The MetaModelica code that also I includes comments and references to the Modelica specification is listed hereafter).



`public uniontype` `Program`
Programs, the top level construct. A program is simply a list of class definitions declared at top level in the source file, combined with a within statement that indicates the hierarchical position of the program."

```
record PROGRAM  "PROGRAM, the top level construct"
  list<Class>  classes "List of classes" ;
  Within       within_ "Within clause" ;
```

```
  end PROGRAM;
end Program;


public uniontype Within "Within Clauses"
```
//See Modelica specification 3.1 **Chapter 13.2.2.3 The within Clause.**

```
  record WITHIN "the within clause"
    Path path "the path for within";
  end WITHIN;

  record TOP end TOP;

end Within;


public uniontype Class
```
"A class definition consists of a name, a flag to indicate if this class is declared as partial, the declared class restriction, and the body of the declaration."
See Modelica specification 3.1 **Chapter 4.5 Class Declarations.**

```
 record CLASS
    Ident name;
    Boolean     partialPrefix   "true if partial" ;
    Boolean     finalPrefix     "true if final" ;
    Boolean     encapsulatedPrefix "true if encapsulated" ;
    Restriction restriction  "Restriction" ;
    ClassDef    body;
  end CLASS;

end Class;


public uniontype ClassDef
```
"The ClassDef type contains the definition part of a class declaration. The definition is either explicit, with a list of parts (public, protected, equation, and algorithm), or it is a definition derived from another class or an enumeration type. For a derived type, the type contains the name of the derived class and an optional array dimension and a list of modifications."
See Modelica specification 3.1 **Chapter 4.5 Class Declarations.**

```
  record PARTS
    list<ClassPart> classParts;
    Option<String>  comment;
  end PARTS;

  record DERIVED
```
See Modelica specification 3.1 **Chapter 4.5.1 Short Class Definitions.**
```
    TypeSpec          typeSpec "typeSpec specification includes array dimen-
sions" ;
    ElementAttributes attributes;
    list<ElementArg>  arguments;
    Option<Comment>   comment;
  end DERIVED;

  record ENUMERATION
```
See Modelica specification 3.1 **Chapter 4.8.5 Enumeration Types.**

```
    EnumDef         enumLiterals;
    Option<Comment> comment;
  end ENUMERATION;

  record OVERLOAD
```
See Modelica specification 3.1 **Chapter 14 Overloaded Operators.**

```
    list<Path>      functionNames;
    Option<Comment> comment;
  end OVERLOAD;

  record CLASS_EXTENDS
```
See Modelica specification 3.1 **Chapter 7.1 Inheritance—Extends Clause.**

```
    Ident            baseClassName  "name of class to extend" ;
    list<ElementArg> modifications  "modifications to be applied to the base
class";
    Option<String>   comment        "comment";
    list<ClassPart>  parts          "class parts";
  end CLASS_EXTENDS;

  record PDER
```
See Modelica specification 3.1 **Chapter 4.5 Class Declarations.**

```
    Path         functionName;
    list<Ident>  vars "derived variables" ;
  end PDER;

end ClassDef;


public uniontype TypeSpec
  record TPATH
    Path path;
    Option<ArrayDim> arrayDim;
  end TPATH;

  record TCOMPLEX
    Path             path;
    list<TypeSpec>   typeSpecs;
    Option<ArrayDim> arrayDim;
  end TCOMPLEX;

end TypeSpec;


public uniontype EnumDef
```
"The definition of an enumeration is either a list of literals or a colon, \':\', which defines a supertype of all enumerations"
See Modelica specification 3.1 **Chapter 4.8.5 Enumeration Types.**

```
  record ENUMLITERALS
    list<EnumLiteral> enumLiterals;
  end ENUMLITERALS;

  record ENUM_COLON end ENUM_COLON;
```

```
end EnumDef;
```

```
public uniontype EnumLiteral
```
"EnumLiteral, which is a name in an enumeration and an optional Comment."
See Modelica specification 3.1 **Chapter 4.8.5 Enumeration Types.**

```
  record ENUMLITERAL
    Ident           literal;
    Option<Comment> comment;
  end ENUMLITERAL;

end EnumLiteral;
```

```
public uniontype ClassPart
```
"A class definition contains several parts. There are public and protected component declarations, type definitions and `extends\' clauses, collectively called elements. There are also equation sections and algorithm sections. The EXTERNAL part is used only by functions which can be declared as external C or FORTRAN functions."

```
  record PUBLIC
```
See Modelica specification 3.1 **Chapter 4.1 Access Control – Public and Protected Elements.**

list<ElementItem> contents ;
```
  end PUBLIC;

  record PROTECTED
```
See Modelica specification 3.1 **Chapter 4.1 Access Control – Public and Protected Elements.**

```
    list<ElementItem> contents;
  end PROTECTED;

  record EQUATIONS
```
See Modelica specification 3.1 **Chapter 8 Equations.**
```
    list<EquationItem> contents;
  end EQUATIONS;

  record INITIALEQUATIONS
```
See Modelica specification 3.1 **Chapter 8.6 Initialization, initial equation, and initial algorithm.**

```
    list<EquationItem> contents;
  end INITIALEQUATIONS;

  record ALGORITHMS
```
See Modelica specification 3.1 **Chapter 11 Statements and Algorithm Sections.**

```
    list<AlgorithmItem> contents;
  end ALGORITHMS;

  record INITIALALGORITHMS
```
See Modelica specification 3.1 **Chapter 8.6 Initialization, initial equation, and initial algorithm.**

```
    list<AlgorithmItem> contents;
  end INITIALALGORITHMS;
```

```
  record EXTERNAL
```
See Modelica specification 3.1 **Chapter 12.9 External Function Interface**.

```
    ExternalDecl externalDecl "externalDecl" ;
    Option<Annotation> annotation_ "annotation" ;
  end EXTERNAL;

end ClassPart;


public uniontype ElementItem
```
"An element item is either an element or an annotation"

```
  record ELEMENTITEM
    Element  element;
  end ELEMENTITEM;

  record ANNOTATIONITEM
    Annotation  annotation_ ;
  end ANNOTATIONITEM;

end ElementItem;


public uniontype Element
```
"Elements: The basic element type in Modelica"

```
  record ELEMENT
    Boolean                    finalPrefix;
    Option<RedeclareKeywords> redeclareKeywords "replaceable, redeclare" ;
    InnerOuter                 innerOuter "inner/outer" ;
    Ident                      name;
    ElementSpec                specification "Actual element specification" ;
    Option<ConstrainClass> constrainClass "constrainClass ; only valid for
classdef and component" ;
  end ELEMENT;

  record DEFINEUNIT
    Ident name;
    list<NamedArg> args;
  end DEFINEUNIT;

  record TEXT
    Option<Ident> optName "optName : optional name of text, e.g. model with
syntax error. We need the name to be able to browse it..." ;
    String string;
    Info info;
  end TEXT;

end Element;


public uniontype ConstrainClass
```
"Constraining type, must be extends".
See Modelica specification 3.1 **Chapter 7.3.2 Constraining Type.**

```
  record CONSTRAINCLASS
    ElementSpec elementSpec "elementSpec ; must be extends" ;
    Option<Comment> comment "comment" ;
  end CONSTRAINCLASS;

end ConstrainClass;
```

```
public uniontype ElementSpec
```
"An element is something that occurs in a public or protected section in a class definition. There is one constructor in the `ElementSpec\' type for each possible element type. There are class definitions (`CLASSDEF\'), `extends\' clauses (`EXTENDS\') and component declarations (`COMPONENTS\'). As an example, if the element `extends TwoPin;\' appears in the source, it is represented in the AST as `EXTENDS(IDENT(\"TwoPin\"),{})\'."

```
  record CLASSDEF
    Boolean replaceable_ "replaceable" ;
    Class class_ "class" ;
  end CLASSDEF;

  record EXTENDS
```
See Modelica specification 3.1 **Chapter 7.1 Inheritance—Extends Clause.**

```
    Path path "path" ;
    list<ElementArg> elementArg "elementArg" ;
    Option<Annotation> annotationOpt "optional annotation";
  end EXTENDS;

  record IMPORT
```
See Modelica specification 3.1 **Chapter 13.2.1 Importing Definitions from a Package.**

```
    Import import_ "import" ;
    Option<Comment> comment "comment" ;
  end IMPORT;

  record COMPONENTS
    ElementAttributes attributes "attributes" ;
    TypeSpec typeSpec "typeSpec" ;
    list<ComponentItem> components "components" ;
  end COMPONENTS;

end ElementSpec;
```

```
public uniontype InnerOuter
```
"One of the keyword inner and outer CAN be given to reference an inner or outer component. Thus there are three disjoint possibilities."
See Modelica specification 3.1 **Chapter 5.4 “Instance Hierarchy Name Lookup of Inner Declarations”** for explanations of inner/outer.

```
  record INNER end INNER;

  record OUTER end OUTER;

  record INNEROUTER end INNEROUTER;

  record UNSPECIFIED end UNSPECIFIED;
```

```
end InnerOuter;


public uniontype Import
```
"Import statements, different kinds"
 // A named import is a import statement to a variable ex;
 // NAMED_IMPORT("SI",Absyn.QUALIFIED("Modelica",Absyn.IDENT("SIunits")));
See Modelica specification 3.1 **Chapter 13.2.1 Importing Definitions from a Package.**

```
  record NAMED_IMPORT
    Ident name "name" ;
    Path path "path" ;
  end NAMED_IMPORT;

  record QUAL_IMPORT
    Path path "path" ;
  end QUAL_IMPORT;

  record UNQUAL_IMPORT
    Path path "path" ;
  end UNQUAL_IMPORT;

end Import;


public uniontype ComponentItem
```
"Collection of component and an optional comment"
See Modelica specification 3.1 **Chapter 4.4.1 Syntax and Examples of Component Declarations.**

```
  record COMPONENTITEM
    Component component "component" ;
    Option<ComponentCondition> condition "condition" ;
    Option<Comment> comment "comment" ;
  end COMPONENTITEM;

end ComponentItem;


public type ComponentCondition = Exp
```
"A componentItem can have a condition that must be fulfilled if the component should be instantiated." ;


```
public uniontype Component
```
"Some kind of Modelica entity (object or variable)"

```
  record COMPONENT
    Ident name "name" ;
    ArrayDim arrayDim "arrayDim ; Array dimensions, if any" ;
    Option<Modification> modification "modification ; Optional modification" ;
  end COMPONENT;

end Component;


public uniontype EquationItem
```

"Several component declarations can be grouped together in one `ElementSpec\' by writing them on the same line in the source. This type contains the information specific to one component."
See Modelica specification 3.1 **Chapter 8 “Equations”.**

```
  record EQUATIONITEM
    Equation equation_ "equation" ;
    Option<Comment> comment "comment" ;
  end EQUATIONITEM;

  record EQUATIONITEMANN
    Annotation annotation_ "annotation" ;
  end EQUATIONITEMANN;

end EquationItem;
```

```
public uniontype AlgorithmItem
```
"Info specific for an algorithm item."
See Modelica specification 3.1 **Chapter 11 “Statements and Algorithm Sections”.**

```
  record ALGORITHMITEM
    Algorithm algorithm_ "algorithm" ;
    Option<Comment> comment "comment" ;
  end ALGORITHMITEM;

  record ALGORITHMITEMANN
    Annotation annotation_ "annotation" ;
  end ALGORITHMITEMANN;

end AlgorithmItem;
```

```
public uniontype Equation
```
"Information on one (kind) of equation, different constructors for different kinds of equations"
See Modelica specification 3.1 **Chapter 8 “Equations”.**

```
  record EQ_IF
    Exp ifExp "ifExp ; Conditional expression" ;
    list<EquationItem> equationTrueItems "equationTrueItems ; true branch" ;
    list<tuple<Exp, list<EquationItem>>> elseIfBranches "elseIfBranches" ;
    list<EquationItem> equationElseItems "equationElseItems Standard 2-side
eqn" ;
  end EQ_IF;

  record EQ_EQUALS
    Exp leftSide "leftSide" ;
    Exp rightSide "rightSide Connect stmt" ;
  end EQ_EQUALS;

  record EQ_CONNECT
    ComponentRef connector1 "connector1" ;
    ComponentRef connector2 "connector2" ;
  end EQ_CONNECT;

  record EQ_FOR
    ForIterators iterators;
```

```
    list<EquationItem> forEquations "forEquations" ;
  end EQ_FOR;

  record EQ_WHEN_E
    Exp whenExp "whenExp" ;
    list<EquationItem> whenEquations "whenEquations" ;
    list<tuple<Exp, list<EquationItem>>> elseWhenEquations "elseWhenEqua-
tions" ;
  end EQ_WHEN_E;

  record EQ_NORETCALL
    ComponentRef functionName "functionName" ;
    FunctionArgs functionArgs "functionArgs; fcalls without return value" ;
  end EQ_NORETCALL;

  record EQ_FAILURE
    EquationItem equ;
  end EQ_FAILURE;

end Equation;


public uniontype Algorithm
```
"The Algorithm type describes one algorithm statement in an algorithm section. It does not describe a whole algorithm. The reason this type is named like this is that the name of the grammar rule for algorithm statements is `algorithm\'."
See Modelica specification 3.1 **Chapter 11 "Statements and Algorithm Sections".**

```
  record ALG_ASSIGN
    Exp assignComponent "assignComponent" ;
    Exp value "value" ;
  end ALG_ASSIGN;

  record ALG_IF
    Exp ifExp "ifExp" ;
    list<AlgorithmItem> trueBranch "trueBranch" ;
    list<tuple<Exp, list<AlgorithmItem>>> elseIfAlgorithmBranch "elseIfAl-
gorithmBranch" ;
    list<AlgorithmItem> elseBranch "elseBranch" ;
  end ALG_IF;

  record ALG_FOR
    ForIterators iterators;
    list<AlgorithmItem> forBody "forBody" ;
  end ALG_FOR;

  record ALG_WHILE
    Exp boolExpr "boolExpr" ;
    list<AlgorithmItem> whileBody "whileBody" ;
  end ALG_WHILE;

  record ALG_WHEN_A
    Exp boolExpr "boolExpr" ;
    list<AlgorithmItem> whenBody "whenBody" ;
    list<tuple<Exp, list<AlgorithmItem>>> elseWhenAlgorithmBranch "elseWhen-
AlgorithmBranch" ;
  end ALG_WHEN_A;
```

```
  record ALG_NORETCALL
    ComponentRef functionCall "functionCall" ;
    FunctionArgs functionArgs "functionArgs; general fcalls without return
value" ;
  end ALG_NORETCALL;

  record ALG_RETURN
  end ALG_RETURN;

  record ALG_BREAK
  end ALG_BREAK;

end Algorithm;
```

```
public uniontype Modification
```
"Modifications are described by the `Modification\' type. There are two forms of modifications: redeclarations and component modifications. - Modifications"
See Modelica specification 3.1 **Chapter 7.2 Modifications.**

```
  record CLASSMOD
    list<ElementArg> elementArgLst;
    Option<Exp> expOption;
  end CLASSMOD;

end Modification;
```

```
public uniontype ElementArg
```
"Wrapper for things that modify elements, modifications and redeclarations"

```
  record MODIFICATION
```
See Modelica specification 3.1 **Chapter 7.2 Modifications.**

```
    Boolean finalItem "finalItem" ;
    Each each_ "each" ;
    ComponentRef componentReg "componentReg" ;
    Option<Modification> modification "modification" ;
    Option<String> comment "comment" ;
  end MODIFICATION;

  record REDECLARATION
```
See Modelica specification 3.1 **Chapter 7.3 Redeclaration.**

```
    Boolean finalItem "finalItem" ;
    RedeclareKeywords redeclareKeywords "redeclare  or replaceable " ;
    Each each_ "each" ;
    ElementSpec elementSpec "elementSpec" ;
    Option<ConstrainClass> constrainClass "class definition or declaration" ;
  end REDECLARATION;

end ElementArg;
```

```
public uniontype RedeclareKeywords
```

"The keywords redeclare and replacable can be given in three different kombinations, each one by themself or the both combined."
See Modelica specification 3.1 **Chapter 7.3 Redeclaration.**

```
  record REDECLARE end REDECLARE;

  record REPLACEABLE end REPLACEABLE;

  record REDECLARE_REPLACEABLE end REDECLARE_REPLACEABLE;

end RedeclareKeywords;
```

```
public uniontype Each
```
"The each keyword can be present in both MODIFICATION\'s and REDECLARATION\'s. - Each attribute"
See Modelica specification 3.1 **Chapter 7.2.5 Modifiers for Array Elements.**

```
  record EACH end EACH;

  record NON_EACH end NON_EACH;

end Each;
```

```
public uniontype ElementAttributes
```
"Component attributes"
See Modelica specification 3.1 **Chapter 4.4.1 Syntax and Examples of Component Declarations.**

```
  record ATTR
    Boolean flowPrefix "flow" ;
    Boolean streamPrefix "stream" ;
    Variability variability "variability ; parameter, constant etc." ;
    Direction direction "direction" ;
    ArrayDim arrayDim "arrayDim" ;
  end ATTR;

end ElementAttributes;
```

```
public uniontype Variability
```
See Modelica specification 3.1 **Chapter 3.8 Variability of Expressions.**

```
  record VAR end VAR;
  record DISCRETE end DISCRETE;
  record PARAM end PARAM;
  record CONST end CONST;

end Variability;
```

```
public uniontype Direction
```
See Modelica specification 3.1 **Chapter 4.4.1 Syntax and Examples of Component Declarations** and **4.4.2.2 Prefix Rules.**

```
  record INPUT end INPUT;
  record OUTPUT end OUTPUT;
```

```
  record BIDIR end BIDIR;

end Direction;
```

public type ArrayDim = list<Subscript>
"Component attributes are properties of components which are applied by type prefixes. As an example, declaring a component as `input Real x;\' will give the attributes `ATTR({},false,VAR,INPUT)\'. Components in Modelica can be scalar or arrays with one or more dimensions. This type is used to indicate the dimensionality of a component or a type definition. Array dimensions" ;

public uniontype Exp
"The Exp uniontype is the container of a Modelica expression. - Expressions"
See Modelica specification 3.1 **Chapter 3 Operators and Expressions.**

```
  record INTEGER
    Integer value;
  end INTEGER;

  record REAL
    Real value;
  end REAL;

  record CREF
    ComponentRef componentReg;
  end CREF;

  record STRING
    String value;
  end STRING;

  record BOOL
    Boolean value;
  end BOOL;

  record BINARY
```
"Binary operations, e.g. a*b"
```
    Exp exp1;
    Operator op;
    Exp exp2;
  end BINARY;

  record UNARY
```
"Unary operations, e.g. -(x)"
```
    Operator op "op" ;
    Exp exp "exp Logical binary operations: and, or" ;
  end UNARY;

  record LBINARY
    Exp exp1 "exp1" ;
    Operator op "op" ;
    Exp exp2 ;
  end LBINARY;

  record LUNARY
```

"Logical unary operations: not"

```
    Operator op "op" ;
    Exp exp "exp Relations, e.g. a >= 0" ;
  end LUNARY;

  record RELATION
    Exp exp1 "exp1" ;
    Operator op "op" ;
    Exp exp2  ;
  end RELATION;

  record IFEXP
    Exp ifExp "ifExp" ;
    Exp trueBranch "trueBranch" ;
    Exp elseBranch "elseBranch" ;
    list<tuple<Exp, Exp>> elseIfBranch "elseIfBranch Function calls" ;
  end IFEXP;

  record CALL
    ComponentRef function_ "function" ;
    FunctionArgs functionArgs ;
  end CALL;

  record PARTEVALFUNCTION "Partially evaluated function"
    ComponentRef function_ "function" ;
    FunctionArgs functionArgs ;
  end PARTEVALFUNCTION;

  record ARRAY   "Array construction using {, }, or array"
    list<Exp> arrayExp ;
  end ARRAY;

  record MATRIX  "Matrix construction using {, } "
    list<list<Exp>> matrix ;
  end MATRIX;

  record RANGE  "Range expressions, e.g. 1:10 or 1:0.5:10"
    Exp start "start" ;
    Option<Exp> step "step" ;
    Exp stop "stop";
  end RANGE;

  record TUPLE " Tuples used in function calls returning several values"
    list<Exp> expressions "comma-separated expressions" ;
  end TUPLE;

  record END "array access operator for last element, e.g. a{end}:=1;"
  end END;

end Exp;
```

uniontype FunctionArgs

"The FunctionArgs uniontype consists of a list of positional arguments followed by a list of named arguments (Modelica v2.0)"

See Modelica specification 3.1 **Chapter 12.4 Function Call.**

```
  record FUNCTIONARGS
    list<Exp> args "args" ;
    list<NamedArg> argNames "argNames" ;
  end FUNCTIONARGS;

  record FOR_ITER_FARG
     Exp  exp "iterator expression";
     ForIterators iterators;
  end FOR_ITER_FARG;

end FunctionArgs;
```

uniontype NamedArg

"The NamedArg uniontype consist of an Identifier for the argument and an expression giving the value of the argument"

```
  record NAMEDARG
    Ident argName "argName" ;
    Exp argValue "argValue" ;
  end NAMEDARG;

end NamedArg;
```

uniontype Operator

"Expression operators"

See Modelica specification 3.1 **Chapter 3 Operators and Expressions.**

```
  /* arithmetic operators */
  record ADD       "addition"                    end ADD;
  record SUB       "subtraction"                 end SUB;
  record MUL       "multiplication"              end MUL;
  record DIV       "division"                    end DIV;
  record POW       "power"                       end POW;
  record UPLUS     "unary plus"                  end UPLUS;
  record UMINUS    "unary minus"                 end UMINUS;
  /* element-wise arithmetic operators */
  record ADD_EW    "element-wise addition"       end ADD_EW;
  record SUB_EW    "element-wise subtraction"    end SUB_EW;
  record MUL_EW    "element-wise multiplication" end MUL_EW;
  record DIV_EW    "element-wise division"       end DIV_EW;
  record POW_EW    "element-wise power"          end POW_EW;
  record UPLUS_EW  "element-wise unary minus"    end UPLUS_EW;
  record UMINUS_EW "element-wise unary plus"     end UMINUS_EW;
  /* logical operators */
  record AND       "logical and"                 end AND;
  record OR        "logical or"                  end OR;
  record NOT       "logical not"                 end NOT;
  /* relational operators */
  record LESS      "less than"                   end LESS;
  record LESSEQ    "less than or equal"          end LESSEQ;
  record GREATER   "greater than"                end GREATER;
  record GREATEREQ "greater than or equal"       end GREATEREQ;
  record EQUAL     "relational equal"            end EQUAL;
  record NEQUAL    "relational not equal"        end NEQUAL;
```

```
end Operator;
```

uniontype Subscript

"The Subscript uniontype is used both in array declarations and component references. This might seem strange, but it is inherited from the grammar. The NOSUB constructor means that the dimension size is undefined when used in a declaration, and when it is used in a component reference it means a slice of the whole dimension. - Subscripts"
See Modelica specification 3.1 **Chapter 10.5 Array Indexing.**

```
  record NOSUB end NOSUB;

  record SUBSCRIPT
    Exp subScript "subScript" ;
  end SUBSCRIPT;

end Subscript;
```

uniontype ComponentRef

"A component reference is the fully or partially qualified name of a component. It is represented as a list of identifier- -subscript pairs. - Component references and paths"

```
  record CREF_QUAL
    Ident name "name" ;
    list<Subscript> subScripts "subScripts" ;
    ComponentRef componentRef "componentRef" ;
  end CREF_QUAL;

  record CREF_IDENT
    Ident name "name" ;
    list<Subscript> subscripts "subscripts" ;
  end CREF_IDENT;

  record WILD end WILD;

end ComponentRef;
```

uniontype Path

"The type `Path\', on the other hand, is used to store references to class names, or names inside class definitions."

```
  record QUALIFIED
    Ident name "name" ;
    Path path "path" ;
  end QUALIFIED;

  record IDENT
    Ident name "name" ;
  end IDENT;
```

record FULLYQUALIFIED

"Used during instantiation for names that are fully qualified, i.e. the names are looked up from top scope directly like for instance Modelica.SIunits.Voltage Note: Not created during parsing, only during instantation to speedup/simplify lookup."

```
    Path path;
```

```
  end FULLYQUALIFIED;
end Path;
```

uniontype Restriction

"These constructors each correspond to a different kind of class declaration in Modelica, except the last four, which are used for the predefined types. The parser assigns each class declaration one of the restrictions, and the actual class definition is checked for conformance during translation. The predefined types are created in the Builtin module and are assigned special restrictions."

See Modelica specification 3.1 **Chapter 4.6 Specialized Classes.**

```
  record R_CLASS end R_CLASS;
  record R_MODEL end R_MODEL;
  record R_RECORD end R_RECORD;
  record R_BLOCK end R_BLOCK;
  record R_CONNECTOR "connector class"  end R_CONNECTOR;
  record R_EXP_CONNECTOR "expandable connector class" end R_EXP_CONNECTOR;
  record R_TYPE end R_TYPE;
  record R_PACKAGE end R_PACKAGE;
  record R_FUNCTION end R_FUNCTION;
  record R_ENUMERATION end R_ENUMERATION;
  record R_PREDEFINED_INT end R_PREDEFINED_INT;
  record R_PREDEFINED_REAL end R_PREDEFINED_REAL;
  record R_PREDEFINED_STRING end R_PREDEFINED_STRING;
  record R_PREDEFINED_BOOL end R_PREDEFINED_BOOL;
  record R_PREDEFINED_ENUM end R_PREDEFINED_ENUM;

end Restriction;
```

public uniontype Annotation

"An Annotation is a class_modification.- Annotation"

See Modelica specification 3.1 **Chapter 17 Annotations.**

```
  record ANNOTATION
    list<ElementArg> elementArgs "elementArgs" ;
  end ANNOTATION;

end Annotation;
```

public uniontype Comment

See Modelica specification 3.1 **Chapter 2.2 Comments.**

```
  record COMMENT
    Option<Annotation> annotation_ "annotation" ;
    Option<String> comment "comment" ;
  end COMMENT;

end Comment;
```

public uniontype ExternalDecl

"Declaration of an external function call – ExternalDecl"

See Modelica specification 3.1 **Chapter 12.9 External Function Interface.**

```
  record EXTERNALDECL
```

```
    Option<Ident>         funcName "The name of the external function" ;
    Option<String>        lang     "Language of the external function" ;
    Option<ComponentRef> output_  "output parameter as return value" ;
    list<Exp>             args     "only positional arguments, i.e. expression
list" ;
    Option<Annotation>    annotation_ ;
  end EXTERNALDECL;

end ExternalDecl;
```

```
public type ForIterator = tuple<Ident, Option<Exp>>
```

See Modelica specification 3.1 **Chapter 11.2.2 For-statement** and **Chapter 8.3.2 For-Equations – Repetitive Equation Structures.**

"For Iterator -
  these are used in:
  * for loops where the expression part can be NONE and then the range
    is taken from an array variable that the iterator is used to index,
    see 3.3.3.2 Several Iterators from Modelica Specification.
  * in array iterators where the expression should always be SOME(Exp),
    see 3.4.4.2 Array constructor with iterators from Specification";

```
public type ForIterators = list<ForIterator>
```

"For Iterators -
  these are used in:
  * for loops where the expression part can be NONE and then the range
    is taken from an array variable that the iterator is used to index,
    see 3.3.3.2 Several Iterators from Modelica Specification.
  * in array iterators where the expression should always be SOME(Exp),
    see 3.4.4.2 Array constructor with iterators from Specification";

Part IV – Transformation

This document defines an incomplete mapping between the SysML4Modelica profile defined in Part II and the abstract syntax defined in Part III. As the definitions found in Part II and III evolve, this transformation document will also change. The majority of this mapping document is composed of tables relating elements in the SysML4Modelica profile to elements of the Modelica abstract syntax. It is also intended to be implementation independent.

Each mapping table may consist of 4 sections:

1. A general statement describing which element in the SysML profile is being mapped to which element of the Modelica abstract syntax.
2. A **Required** section describing the required conditions necessary to make the transformation valid
3. A **Conditional** section describing possible links between attributes based on conditional expressions
4. An **Attributes** section describing the mapping between any additional attributes

1 Class Definition

1.1 Overview

Table 1: Overview of mapping from new SysML stereotypes to Modelica specialized classes.

SysML4Modelica	Modelica		Attributes
	Abstract Syntax	Concrete Syntax	
Classes::ModelicaClassDefinition	Absyn.Class.Class	N/A	See Below
Specializations:			
Classes::ModelicaClass	Absyn.Class.Class	Class	See Section Error: Reference source not found
Classes::ModelicaModel	Absyn.Class.Class	Model	See Section Error: Reference source not found
Classes::ModelicaRecord	Absyn.Class.Class	Record	See Section Error: Reference source not found
Classes::ModelicaBlock	Absyn.Class.Class	Block	See Section Error: Reference source not found
Classes::ModelicaConnector	Absyn.Class.Class	Connector	See Section Error: Reference source not found
Classes::ModelicaType	Absyn.Class.Class	Type	See Section Error: Reference source not found

Classes::ModelicaPackage	Absyn.Class.Class	Package	See Section Error: Reference source not found
Classes::ModelicaFunction	Absyn.Class.Class	Function	See Section Error: Reference source not found

SysML4Modelica		**Modelica**
Classes::ModelicaClassDefinition	maps to	Absyn.Class.Class
Attributes:		
• IsFinal	always maps to	• finalPrefix
• IsModelicaEncapsulated	always maps to	• encapsulatedPrefix
• IsAbstract	always maps to	• PartialPrefix

1.2 «modelicaClass»

SysML4Modelica		**Modelica**
Classes::ModelicaClass	maps to	Absyn.Class.Class
Required:		
		• restriction **equal to** Restriction.R_Class
Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition		

1.3 «modelicaModel»

SysML4Modelica		**Modelica**
Classes::ModelicaModel	maps to	Absyn.Class.Class
Required:		
		• restriction **equal to** Restriction.R_Model
Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition		

1.4 «modelicaRecord»

SysML4Modelica		**Modelica**
Classes::ModelicaRecord	maps to	Absyn.Class.Class
Required:		
		• restriction **equal to** Restriction.R_Record

Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition

1.5 «modelicaBlock»

SysML4Modelica		Modelica
Classes::ModelicaBlock	maps to	Absyn.Class.Class
Required:		
		• restriction **equal to** Restriction.R_Block
Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition		

1.6 «modelicaConnector»

SysML4Modelica		Modelica
Classes::ModelicaConnector	maps to	Absyn.Class.Class
Conditional:		
• IsExpandable **equal to** false	maps to	• restriction **equal to** Restriction.R_CONNECTOR
• IsExpandable **equal to** true	maps to	• restriction **equal to** Restriction. R_EXP_CONNECTOR
Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition		

1.7 «modelicaType»

SysML4Modelica		Modelica
Classes::ModelicaType	maps to	Absyn.Class.Class
Required:		
		• restriction **equal to** Restriction.R_Type
Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition		

1.8 «modelicaPackage»

SysML4Modelica		Modelica
Classes::ModelicaPackage	maps to	Absyn.Class.Class
Required:		
		• restriction **equal to** Restriction.R_Package
Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition		

1.9 «modelicaFunction»

SysML4Modelica		Modelica
SysML4Modelica::Classes::ModelicaFunction	maps to	Absyn.Class.Class
Required:		
		• restriction **equal to** Restriction.R_Function
Conditional:		
• IsExternal **equal to** true		• **Type of** Class.body **equal to** ClassDef.PARTS • **Type of** Class.blody.classParts **equal to** ClassPart.EXTERNAL • **Type of** Class.body.classParts.externalDecl **equal to** ExternalDecl.EXTERNALDECL
• ModelicaFunction::externalLanguage	Always maps to	• Class.body.classParts.externalDecl.lang
• ModelicaFunction::externalFunctionSpecification	Always maps to	• Class.body.classParts.externalDecl.funcName
Attributes same as SysML4Modelica::Classes::ModelicaClassDefinition		

1.10 «modelicaEnumeration»

Currently not covered in this draft.

1.11 «modelicaExtends»

Currently not covered in this draft.

Short Class Definitions

Currently not covered in this draft.

2 Predefined Types

2.1 Overview

The following primitive types are available in the Modelica language: Real Type, Integer Type, Boolean Type, String Type, Enumeration Types, StateSelect, ExternalObject, Graphical Annotation Types. These primitive types are defined as predefined types in SysML4Modelica::BasicTypes. Although these types have direct counterparts in SysML, they are defined again to account for the additional attributes associated with them in Modelica.

SysML4Modelica	Modelica
BasicTypes	**Predefined Type**
ModelicaReal	Real
ModelicaInteger	Integer
ModelicaBoolean	Boolean
ModelicaString	String
ModelicaEnumeration	Enumeration

SysML4Modelica	Modelica
ModelicaStateSelect	StateSelect
ModelicaExternalObject	ExternalObject
ModelicaAnnotation	Annotation

3 Component Declarations

3.1 Overview

SysML4Modelica	Modelica	Attributes
Component::ModelicaComponent	Absyn.Element.Element	See Section 3.2
Component::ModelicaPart	Absyn.Element.Element	See Section 3.3
Component::ModelicaPort	Absyn.Element.Element	See Section 3.4
Component::ModelicaValueProperty	Absyn.Element.Element	See Section 3.5

3.2 «modelicaComponent»

SysML4Modelica		Modelica
Component::ModelicaComponent	maps to	Absyn.Element.Element
Required:		
		• **Type of** specification **equal to** ElementSpec.COMPONENTS • **Type of** specification.components **equal to** ComponentItem**.**COMPONENTITEM • **Type of** specification.components.component **equal to** Componenet.COMPONENT
Attributes:		
• name	always maps to	• name • specification.components.component.name(?)
• scope	always maps to	• **Type of** innerOuter
• conditionalExpression	always maps to	• specification.components.condition
• modification	always maps to	• specification.components.component.modification
• redeclaration	always maps to	• redeclareKeywords
• arraySize	always maps to	• specification.components.component.arrayDim

3.3 «modelicaPart»

SysML4Modelica		Modelica

Component::ModelicaPart	maps to	Absyn.Element.Element
Required:		
		• **Type of** specification **equal to** ElementSpec.COMPONENTS • Absyn.Class.Class **referenced by** specification.typeSpec has Restriction **not equal to** R_Connector or R_Type • **Type of** specification.components **equal to** ComponentItem**.**COMPONENTITEM • **Type of** specification.components.component **equal to** Componenet.COMPONENT
Attributes:		
• name	always maps to	• name • specification.components.component.name(?)
• Scope	always maps to	• **Type of** innerOuter
• conditionalExpression	always maps to	• specification.components.condition
• modification	always maps to	• specification.components.component.modification
• redeclaration	always maps to	• redeclareKeywords
• arraySize	always maps to	• specification.components.component.arrayDim

3.4 «modelicaPort»

SysML4Modelica		**Modelica**
Component::ModelicaPort	maps to	Absyn.Element.Element
Required:		
		• **Type of** specification **equal to** ElementSpec.COMPONENTS • Absyn.Class.Class **referenced by** specification.typeSpec has restriction **equal to** R_Connector or • **Type of** specification.components **equal to** ComponentItem**.**COMPONENTITEM • **Type of** specification.components.component **equal to** Componenet.COMPONENT
Attributes:		
• name	always maps to	• name • specification.components.compon-

		ent.name(?)
• conditionalExpression	always maps to	• specification.components.condition
• modification	always maps to	• specification.components.component.modification
• redeclaration	always maps to	• redeclareKeywords
• arraySize	always maps to	• specification.components.component.arrayDim

3.5 «modelicaValueProperty»

SysML4Modelica		**Modelica**
Component::ModelicaValueProperty	maps to	Absyn.Element.Element
Required:		
		• **Type of** specification **equal to** ElementSpec.COMPONENTS • Absyn.Class.Class **referenced by** specification.typeSpec has restriction **equal to** R_Type • **Type of** specification.components **equal to** ComponentItem**.**COMPONENTITEM • **Type of** specification.components.component **equal to** Componenet.COMPONENT • **Type of** specification.components.component.attributes **equal to** ElementAttributes.ATTR
Attributes:		
• name	always maps to	• name • specification.components.component.name(?)
• scope	always maps to	• **Type of** innerOuter
• flowFlag	always maps to	• specification.components.component.attributes.flowPrefix
• causality	always maps to	• **Type of** specification.components.component.attributes.direction
• variability	always maps to	• **Type of** specification.components.component.attributes.variability
• conditionalExpression	always maps to	• specification.components.condition
• modification	always maps to	• specification.components.component.modification
• redeclaration	always maps to	• redeclareKeywords
• arraySize	always maps to	• specification.components.component.arrayDim

3.6 «modelicaFunctionParameter»

Currently not covered in this draft.

4 Equation and Algorithm Sections

4.1 Overview

SysML4Modelica	Modelica Abstract Syntax	Attributes
Equations and Algorithms::ModelicaEquation	Absyn.EquationItem.EQUATIONITEM	See Section 4.2
Equations and Algorithms::ModelicaConnection	Absyn.Equation.EQ_CONNECT	See Section 4.3
Equations and Algorithms::ModelicaAlgorithm	Absyn.EquationItem.ALGORITHMITEM	See Section 4.4

4.2 «modelicaEquation»

SysML4Modelica		Modelica
Equations and Algorithms::ModelicaEquation	maps to	Absyn.EquationItem.EQUATIONITEM
Required:		
• constraint.specification	parsed to	• equation
Conditionals:		
• If IsInitial **equal to** false		• EQUATIONITEM **contained in record typed to** ClassPart.EQUATIONS
• If IsInitial **equal to** true		• EQUATIONITEM **contained in record typed to** ClassPart.INITIALEQUATIONS

4.3 «modelicaAlgorithm»

SysML4Modelica		Modelica
Equations and Algorithms::ModelicaAlgorithm	maps to	Absyn.AlgorithmItem.ALGORITHMITEM
Required:		
• constraint.specification	parsed to	• algorithm_
Conditionals:		
• If IsInitial **equal to** false		• ALGORITHMITEM **contained in record typed to** ClassPart.ALGORITHMS
• If IsInitial **equal to** true		• ALGORITHMITEM **contained in record typed to** ClassPart.INITIALALGORITHMS

4.4 «modelicaConnection»

SysML4Modelica		Modelica
Equations and Algorithms::ModelicaConnection	maps to	Absyn.Equation.EQ_CONNECTOR
Required:		
• ConnectorEndA.Role	maps to	• connector1
• ConnectorEndB.Role	maps to	• connector2