

# Towards Unified System Modeling with the ModelicaML UML Profile

Adrian Pop, David Akhvlediani, Peter Fritzson

Programming Environments Lab,  
Department of Computer and Information Science  
Linköping University, SE-581 83 Linköping, Sweden  
{adrpo, petfr}@ida.liu.se

**Abstract.** In order to support the development of complex products, modeling tools and processes need to support co-design of software and hardware in an integrated way. Modelica is the major object-oriented mathematical modeling language for component-oriented modeling of complex physical systems and UML is the dominant graphical modeling notation for software. In this paper we propose ModelicaML UML profile as an integration of Modelica and UML. The profile combines the major UML diagrams with Modelica graphic connection diagrams and is based on the System Modeling Language (SysML) profile.

## 1 Introduction

The development in system modeling has come to the point where complete modeling of systems is possible, e.g. the complete propulsion system, fuel system, hydraulic actuation system, etc., including embedded software can be modeled and simulated concurrently. This does not mean that all components are dealt with down to the very smallest details of their behavior. It does, however, mean that all functionality is modeled, at least qualitatively.

In this paper, a UML profile for Modelica, named ModelicaML, is proposed. The ModelicaML UML profile is based on the OMG SysML™ (Systems Modeling Language) profile and reuses its artifacts required for system specification. SysML diagrams are also extended to support all Modelica constructs. We argue that with ModelicaML system engineers are able to specify entire systems, starting from requirements, continuing with behavior and finally perform system simulations.

## 2 SysML and Modelica

The Unified Modeling Language (UML) has been created to assist software development processes by providing means to capture software system structure and behavior. This evolved into the main standard for Model Driven Development [5].

The System Modeling Language (SysML) [4] is a graphical modeling language for systems engineering applications. SysML was developed by systems engineering ex-

perts, and was adopted by OMG in 2006. SysML is built on top of UML and tailored to the needs of system engineers by supporting specification, analysis, design, verification and validation of broad range of systems and system-of-systems.

The main goal behind SysML is to unify and replace different document-centric approaches in the system engineering field with a single systems modeling language. A single model-centric approach improves communication, assists to manage complex system design and allows its early validation and verification.

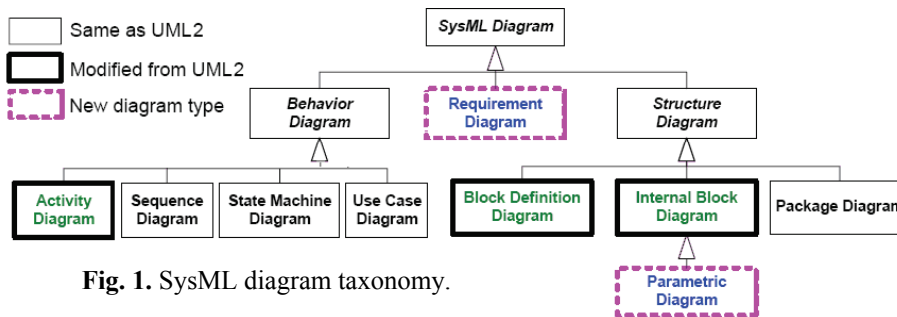


Fig. 1. SysML diagram taxonomy.

The taxonomy of SysML diagrams is presented in Fig. 1. For a full description of SysML see (SysML, 2006) [4]. The major SysML extensions compared to UML are:

- *Requirements diagrams* support requirements presentation in tabular or in graphical notation, allows composition of requirements and supports traceability, verification and “fulfillment of requirements”.
- *Block diagrams* extend the Composite Structure diagram of UML2.0. The purpose of this diagram is to capture system components, their parts and connections between parts. Connections are handled by means of connecting ports which may contain data, material or energy flows.
- *Parametric diagrams* help perform engineering analysis such as performance analysis. Parametric diagrams contain constraint elements, which define mathematical equations, linked to properties of model elements.
- *Activity diagrams* show system behavior as data and control flows. Activity diagrams are similar to Extended Functional Flow Block Diagrams, which are already widely used by system engineers. Activity decomposition is supported. by SysML.
- *Allocations* are used to define mappings between model elements: For example, certain Activities may be allocated to Blocks (to be performed by the block).

SysML block definitions (Fig. 2) can include properties to specify block parts, values and references to other blocks. A separate compartment is dedicated for each of these features. To describe the behavior of a block the “Operations” compartment is reused from UML and it lists operations that describe certain behavior. SysML defines a special form of an optional compartment for constraint definitions owned by a block. A “Namespace” compartment may appear if nested block definitions exist for a block. A “Structure” compartment may appear to show internal parts and connections between parts within a block definition.

SysML defines two types of ports: standard ports and flow ports. Standard ports, which are reused from UML, are service-oriented ports required or provided by a block. Flow ports specify interaction points through which items may flow between

blocks, and between blocks and environment. A flow port definition may include single item specification or complex flow specification through the FlowSpecification interface; flow ports define what “can” flow between the block and its environment. Flow direction can be specified for a flow port in SysML. SysML also defines a notion of Item flows that specify “what” does flow in a particular usage context.

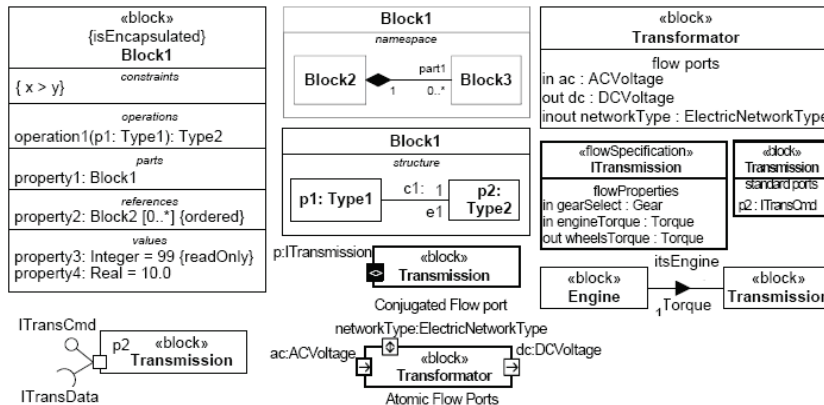


Fig. 2. SysML block definitions.

## 2.1 Modelica

Modelica [2] [3] is a modern language for equation-based object-oriented mathematical modeling primarily of physical systems. Several tools, ranging from open-source as OpenModelica [1], to commercial like Dymola [11] or MathModelica [10] support the Modelica specification.

The language allows defining models in a declarative manner, modularly and hierarchically and combining various formalisms expressible in the more general Modelica formalism. The multidomain capability of Modelica allows combining electrical, mechanical, hydraulic, thermodynamic, etc., model components within the same application model. In short, Modelica has improvements in several important areas:

- *Object-oriented mathematical modeling.* This technique makes it possible to create model components, which are employed to support hierarchical structuring, reuse, and evolution of large and complex models covering multiple technology domains.
- *Physical modeling of multiple application domains.* Model components can correspond to physical objects in the real world, in contrast to established techniques that require conversion to “signal” blocks with fixed input/output causality. In Modelica the structure of the model naturally correspond to the structure of the physical system in contrast to block-oriented modeling tools.
- *Acausal modeling.* Modeling is based on *equations* instead of assignment statements as in traditional input/output block abstractions. Direct use of equations significantly increases re-usability of model components, since components adapt to the data flow context in which they are used.

Hierarchical system architectures can easily be described with Modelica thanks to its powerful component model. The *Components* are connected via the *connection mechanism* realized by the Modelica system, which can be visualized in connection diagrams. The *component framework* realizes components and connections, and ensures that communication works over the connections. For systems composed of *acausal* components with behavior specified by equations, the direction of data flow, i.e., the *causality* is initially unspecified for connections between those components and the causality is automatically deduced by the compiler when needed. Components have well-defined *interfaces* consisting of ports, also known as *connectors*, to the external world. A component may internally consist of other connected components, i.e., *hierarchical modeling* as in Fig. 3.

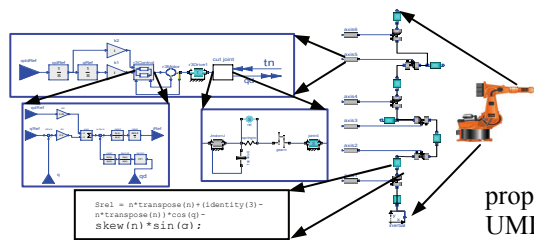


Fig. 3. Hierarchical model of an industrial robot.

## 2.2 SysML vs. Modelica

The System Modeling Language (SysML) has recently been proposed and defined as an extension of UML targeting at systems engineers. As

previously stated, the goal of SysML is to unify different

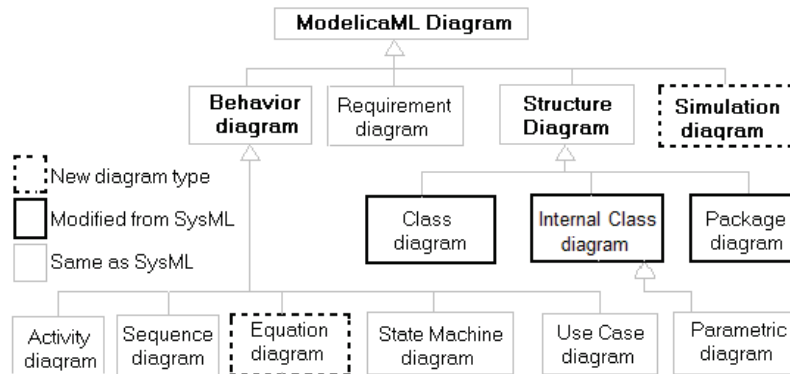
approaches and languages used by system engineers into a single standard. SysML models may span different domains, for example, electrical, mechanical and software. Even if SysML provides means to describe system behavior like Activity and State Chart Diagrams, the precise behavior can not be described and simulated. In that respect, SysML is rather incomplete compared to Modelica.

Modelica also, was created to unify and extend object-oriented mathematical modeling languages. It has powerful means for describing precise component behavior and functionality in a declarative way. Modelica models can be graphically composed using Modelica connection diagrams which depict the structure of designed system. However, complex system design is more than just a component assembly. In order to build a complex system, system engineers have to gather requirements, specify system components, define system structure, define design alternatives, describe overall system behavior and perform its validation and verification.

## 3 ModelicaML: a UML profile for Modelica

ModelicaML reuses several diagrams types from SysML without any extension, extends some of them, and also provides several new ones. The ModelicaML diagram overview is shown in Fig. 4. Diagrams are grouped into four categories: Structure, Behavior, Simulation and Requirement. In the following we present the most important ModelicaML profile diagrams. The full description of the ModelicaML profile is presented in [8]. The most important properties of the ModelicaML profile are outlined in the following:

- The ModelicaML profile supports modeling with all Modelica constructs and properties i.e. restricted classes, equations, generics, discrete variables, etc.
- Using ModelicaML diagrams it is possible to describe multiple aspects of a system being designed and thus support system development process phases such as requirements analysis, design, implementation, verification, validation and integration.
- ModelicaML is partly based on SysML, but reuses and extends its elements.
- The profile supports mathematical modeling with equations since equations specify behavior of a (Modelica) system. Algorithm sections are also supported.
- Simulation diagrams are introduced to model and document simulation parameters and results in a consistent and usable way.
- The ModelicaML meta-model is consistent with SysML in order to provide SysML-to-ModelicaML conversion.



**Fig. 4.** ModelicaML diagrams overview.

Three SysML diagram types have been partly reused and changed for the ModelicaML profile. The rest of the diagram types we used in ModelicaML unchanged:

- The SysML Block Definition Diagram has been updated and renamed to *Modelica Class Diagram*.
- The SysML Internal Block Diagram has been updated and renamed to *Modelica Internal Class Diagram* (some of the SysML constructs are disabled).
- The *Package Diagram* has been changed in order to fully support the Modelica language (i.e. Modelica package constants, Generic Packages, etc).
- Other SysML diagram types such as Use Case Diagram, Activity Diagrams and Allocations, and State Machine Diagrams are included in ModelicaML without modifications. ModelicaML reuses Sequence Diagrams from SysML and changes the semantics of message passing. Modelica doesn't support method declaration within a single class but supports declaration of functions as a restricted class type.

Thus, the following diagram types are available in the ModelicaML profile:

- The *Modelica Class Diagram* usually describes class definitions and their relationships such as inheritance and containment.

- The *Modelica Internal Class Diagram* describes the internal class structure and interconnections between parts.
- The *Package Diagram* groups logically connected user defined elements into packages. In ModelicaML the primarily purpose of this diagram is to support the specifics of the Modelica packages.
- Activity, Sequence, State Machine, Use Case, Parametric and Requirements diagrams have been reused without modification from SysML.
- Two new diagrams, *Simulation Diagram* and *Equation Diagram*, not present in SysML, have been included in the ModelicaML profile.

### 3.1 Package Diagram

A UML Package is a general purpose model element for grouping other elements within a separate namespace. With a help of packages, designers are able group elements to correspond to different structures/views of a system. ModelicaML extends UML packages in order to support Modelica packaging features, in particular: package inheritance, generic packages, constant declaration within a package, package “instantiation” and renaming import (see [2] for Modelica packages details).

A diagram which contains package elements and their relationships is called a Package Diagram. Modelica packages have a hierarchical structure containing package elements as nodes. In Modelica, packages are used to structure model elements into libraries. A snapshot of the Modelica Standard Library hierarchy is shown in Fig. 5 using UML notation. Package nodes in the hierarchy are connected via the package containment link as in the example in Fig. 6.

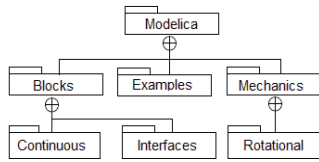


Fig. 5. Package hierarchy modeling.

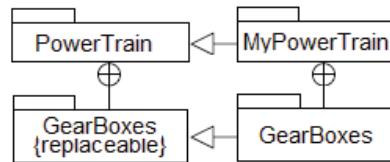


Fig. 6. Package hierarchy modeling

### 3.2 Modelica Class Diagrams

Modelica uses restricted classes such as `class`, `model`, `block`, `connector`, `function` and `record` to describe a system. Modelica classes have essentially the same semantics as SysML blocks specified in [4] and provide a general-purpose capability to model systems as hierarchies of modular components. ModelicaML extends SysML blocks by defining features which are relevant or unique to Modelica.

The purpose of the Modelica Class Diagram is to show features of Modelica classes and relationships between classes. Additional kind of dependencies and associations between model elements may also be shown in a Modelica Class Diagram. For example, behavior description constructs – equations, may be associated with particular Modelica Classes. The detailed description of structural features of Modeli-

caML is provided below. ModelicaML structural extensions are defined based on the SysML block definition outlined in section 2.

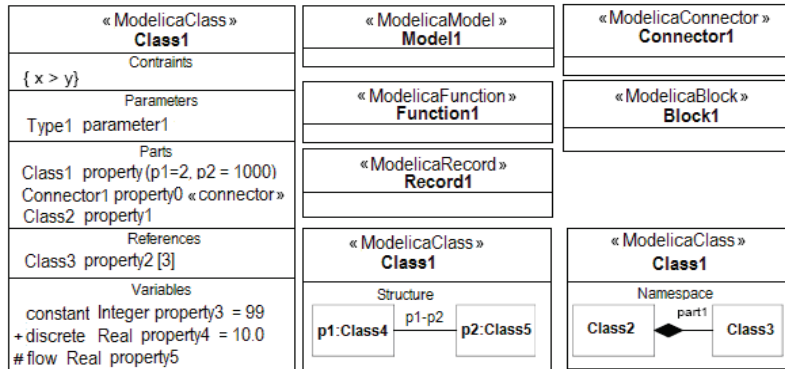


Fig. 7. ModelicaML class definitions.

### 3.2.1 Modelica Class Definition

The graphical notation of ModelicaML class definitions is shown in Fig. 7. Each class definition is adorned with a stereotype name that indicates the class type it represents. The ModelicaML Class Definition has several compartments to group its features: parameters, parts, variables. We designed the parameters compartment separately from variables because the parameters need to be assigned values in order to simulate a model (see the Simulation Diagram later on). Some compartments are visible by default; some are optional and may be shown on ModelicaML Class Diagram with the help of a tool. Property signatures follow the Modelica textual syntax and not the SysML original syntax, reused from UML. A ModelicaML/SysML tool may allow users to choose between UML or Modelica style textual signature presentation. Using Modelica syntax on a diagram has the advantage of being more compatible with Modelica and being more straightforward for Modelica users. The Modelica syntax is quite simple to learn even for users not acquainted with Modelica.

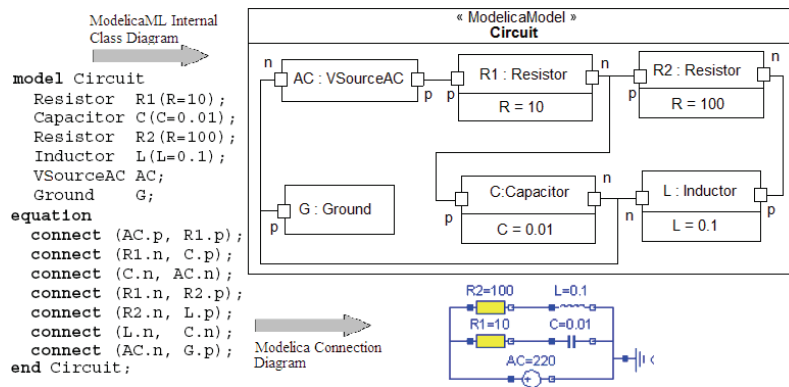
ModelicaML provides extensions to SysML in order to support the full set of Modelica constructs and features. For example, ModelicaML defines unique class definition types ModelicaClass, ModelicaModel, ModelicaBlock, ModelicaConnector, ModelicaFunction and ModelicaRecord that correspond to `class`, `model`, `block`, `connector`, `function` and `record` restricted Modelica classes. We included the Modelica specific restricted classes because a modeling tool needs to impose their semantic restrictions (for example a record cannot have equations, etc).

### 3.2.2 Modelica Internal Class Diagram

The Modelica Internal Class Diagram is based on the SysML Internal Block Diagram but the connections are based on ModelicaConnector. The Modelica Class Diagram defines Modelica classes and relationships between classes, like generalizations, association and dependencies, whereas a Modelica Internal Class Diagram shows the internal structure of a class in terms of parts and connections. The Modelica Internal Class Diagram is similar to Modelica connection diagram, which presents parts in a

graphical (icon) form. An example Modelica model presented as a Modelica Internal Class diagram is shown in Fig. 8.

Usually Modelica models are presented graphically via Modelica connection diagrams (Fig. 8, bottom). Such diagrams are created by the modeler using a graphic connection editor by connecting together components from available libraries. Since both diagram types are used to compose models and serve the same purpose, we briefly compare the Modelica connection diagram to the Modelica Internal Class Diagram. The main advantage of the Modelica connection diagram over the Internal Class Diagram is that it has better visual comprehension as components are shown via domain-specific icons known to application modelers. Another advantage is that Modelica library developers are able to predefine connector locations on an icon, which are related to the semantics of the component. In the case of a ModelicaML Internal Class Diagram a SysML/ModelicaML tool should somehow point out at which side of a rectangular presentation of a part to place a port (connector).



**Fig. 8.** ModelicaML Internal Class vs. Modelica Connection Diagram.

One of the advantages of the Internal Class Diagram is that it directly supports nested structures. However, nested structures are also available behind the icons in a Modelica connection diagram, thus using the drawing area more effectively.

The main advantage of the Internal Class Diagram is that it highlights top-level Modelica model parameters and variables specification in separate compartments.

Other SysML elements, such as Activities and Requirements which do not exist in Modelica but are very important for additional model specification can be combined with both Internal Class Diagram and Modelica connection diagrams.

### 3.4 Parametric Diagrams vs. Equation Diagrams

SysML defines Constraint blocks which specify mathematical expressions, like equations, to constrain physical properties of a system. Constraint blocks are defined in the Block Definition diagram and can be packaged into domain-specific libraries for later reuse. There is a special diagram type called Parametric Diagram which relates block parameters with certain constraints blocks. The Parametric Diagram is included in ModelicaML without any modifications to keep the compatibility with SysML.



The Modelica class behavior is usually described by equations, which also constrain Modelica class parameters, and have a domain-specific usage. SysML constraint blocks are less powerful means of domain model description than Modelica equations. Modelica equations include some type of equations, which cannot be modeled using Constraint blocks, i.e.: `if`, `for`, `when` equations. Also, modeling complexity is an issue, as for example in Fig. 9 there are only four equations, and the diagram is already quite complex. However, grouping constraint blocks into libraries can be useful for system engineers who use Modelica and SysML. SysML Parametric diagram may be used during the initial design phase, when equations related to a class are being identified using Parametric Diagrams and finally associated (via an Equation Diagram) with a Modelica class or set of classes.

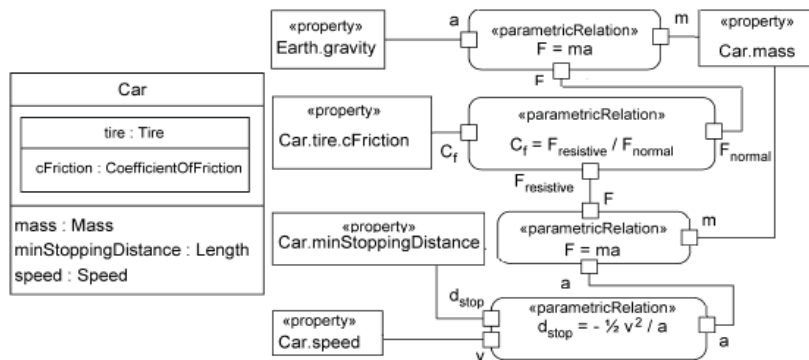


Fig. 9. Parametric Diagram Example

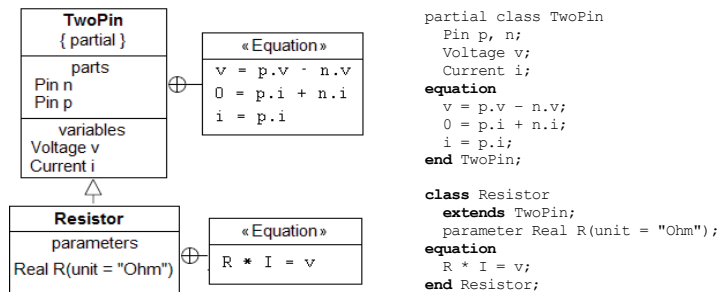


Fig. 10. Equation modeling example with a Modelica Class Diagram.

In Fig. 10, Fig. 11 we present examples of behavior specification via Equation Diagrams in ModelicaML. Equations do not prescribe a certain data flow direction which means that the order in which equations appear in a model do not influence their meaning and semantics. The only requirement for a system of equations is to be solvable. For further details about Modelica equations, see [2]. Besides simple equality equations, Modelica allows other kind of equations be presented within a model. For each of such kind of equations (i.e. when/if/initial equations) ModelicaML defines a graphical construct. It's up to designer to decide whether to use simple equations block representation or specific construct for equation modeling. Algorithm sections are modeled similar to equations, as text.

With a help of Equation Diagram top-down modeling approach is applied to behavior modeling. First, the primarily equations may be captured, then conditional constructs applied, equations text description substituted with mathematical expressions or even equations refactored by moving to other classes. In the similar way as Modelica classes are grouped by physical domain libraries, common equations can be packaged into domain-specific libraries and be reused during a design process. Moreover, equation constructs shown on Equation Diagram can be linked to Activity elements or with Requirement elements to show that a specific requirement has been fulfilled.

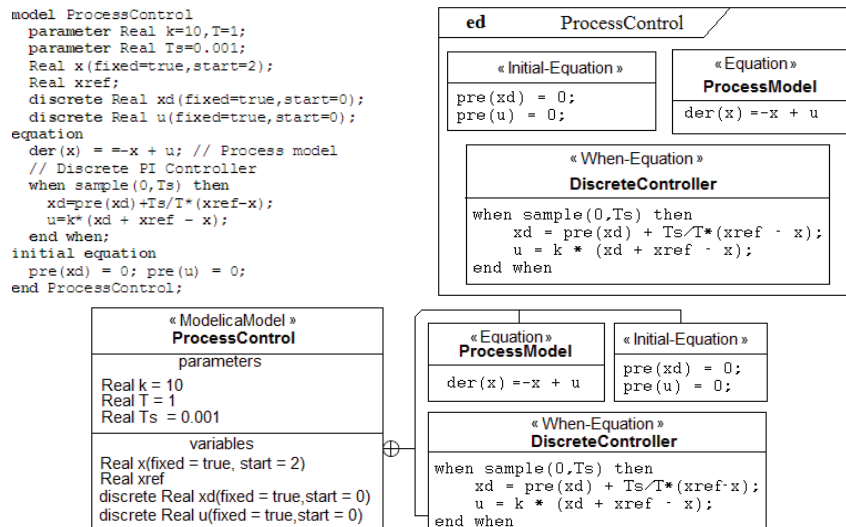


Fig. 11. ModelicaML nested/extern Equation diagrams

### 3.5 Simulation Diagram

ModelicaML introduces a new diagram type, called Simulation Diagram (Fig. 12), used for simulation modeling. Simulation is usually performed by a simulation tool which allows parameter setting, variable selection for output and plotting. The Simulation Diagram may be used to store any simulation experiment, thus helping to keep the history of simulations and its results. When integrated with a modeling and simulation environment, a simulation diagram may be automatically generated.

The Simulation Diagram provides facilities for simulation planning, structured presentation of parameter passing and simulation results. Simulations can be run directly from the Simulation Diagram. Association of simulation results with requirements from a domain expert and additional documentation (e.g. by: Note, Problem Rationale text boxes of SysML) are also supported by the Simulation Diagram. The Simulation Diagram introduces new diagram elements: “Parameter” element and two stereotyped dependency associations, “simParameter” and “simResults”. Parameter values are associated with a class via simParameter for a simulation. Simulation results are associated with a model via simResults which specify which variable is to be plotted and for what time interval.

For simulation purposes, the Simulation Diagram can be integrated with any Modelica modeling and simulation environment. We are currently in the process of designing a ModelicaML development environment which integrates with the OpenModelica modeling and simulation environment.

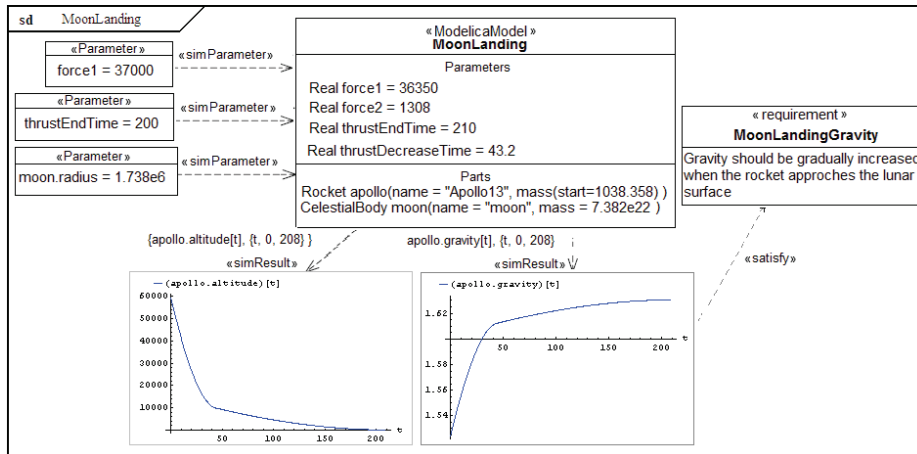


Fig. 12. Simulation Diagram example.

## 4 Conclusion and Future Work

In this paper we propose the ModelicaML profile that integrates Modelica and UML. UML Statecharts and Modelica have been previously integrated, see e.g. [9][15]. SysML is rather new but it was already adopted for system on chip design [13] evaluated for code generation [14] or extended with bond graphs support [12].

The support for Modelica in ModelicaML allows precisely defining, specifying and simulating physical systems. Modelica provides the means for defining behavior for SysML block diagrams while the additional modeling capabilities of SysML provides additional modeling and specification power to Modelica (e.g. requirements and inheritance diagrams, etc).

As a future project we plan to implement an Eclipse-based [6] graphical editor for ModelicaML as a part of our Modelica Development Tooling (MDT) [7].

## References

- [1] P Fritzson, P Aronsson, H Lundval, K Nyström, A Pop, L Saldamli, and D Broman. *The OpenModelica Modeling, Simulation, and Software Development Environment*. In Simulation News Europe, 44/45, Dec 2005. <http://ww.ida.liu.se/projects/OpenModelica>.
- [2] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pp., Wiley-IEEE Press, 2004. See also: <http://www.mathcore.com/drmodelica/>
- [3] The Modelica Association. *The Modelica Language Specification Version 2.2*.
- [4] OMG, *System Modeling Language, (SysML)*, www: <http://www.omgsysml.org>
- [5] OMG : Guide to Model Driven Architecture: The MDA Guide v1.0.1

- [6] Eclipse.Org, www: <http://www.eclipse.org/>
- [7] A Pop, P Fritzson, A Remar, E Jagudin, and D Akhvlediani. *OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging*. In Proc of the Modelica'2006, Vienna, Austria, Sept. 4-5, 2006.
- [8] David Akhvlediani. *Design and implementation of a UML profile for Modelica/SysML*. Final Thesis, LITH-IDA-EX--06/061—SE, April 2007.
- [9] J.Ferreira, J. Estima, *Modeling hybrid systems using Statecharts and Modelica*". 7<sup>th</sup> IEEE Intl. Conf. on Emerging Technologies and Factory Automation, October 1999, Spain.
- [10] MathCore Engineering AB, *MathModelica* <http://www.mathcore.com>
- [11] Dynasim AB, *Dymola*, <http://www.dynasim.com>
- [12] Skander Turki, Thierry Soriano, *A SysML Extension for Bond Graphs Support*, Proc. of the International Conference on Technology and Automation (ICTA), Greece, 2005
- [13] Yves Vanderperren, Wim Dehane, *SysML and Systems Engineering Applied to UML-Based SoC Design*, Proc. of the 2nd UML-SoC Workshop at 42nd DAC, USA, 2005.
- [14] Yves Vanderperren, Wim Dehane, *From UML/SysML to Matlab/Simulink*, Proceedings of the Conference on Design, Automation and Test in Europe (DATE), Munich, 2006.
- [15] André Nordwig. *Formal Integration of Structural Dynamics into the Object-Oriented Modeling of Hybrid Systems*. ESM 2002: 128-134.