Eclipse Support for Design and Requirements Engineering Based on ModelicaML

Adrian Pop, Vasile Băluță, Peter Fritzson
Programming Environments Lab
Department of Computer and Information Science
Linköping University,
SE-581 83 Linköping, Sweden
{adrpo,x07vasba,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. The ModelicaML UML profile integrates Modelica and UML to support engineering of whole products. In this paper we present the Eclipse ModelicaML implementation and integration with the MDT Eclipse plugin, with emphasis on requirements support.

Keywords: Requirements, Modeling and Simulation, Modelica, Eclipse, UML, SysML, ModelicaML.

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, the implementation of the ModelicaML UML profile for Modelica in Eclipse is presented, with emphasis on requirement diagrams.

1.1 SysML

The Unified Modeling Language (UML) has been created to assist software development processes by providing means to capture software system structure and behavior. This has 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 experts, 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 a 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 managaging complex system design and allows its early validation and verification.

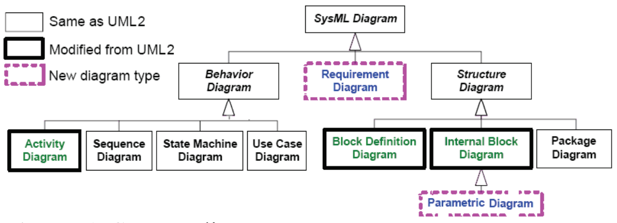


Figure 1. SysML diagram taxonomy.

The taxonomy of SysML diagrams is presented in Figure 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; allow composition of requirements; and support 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).

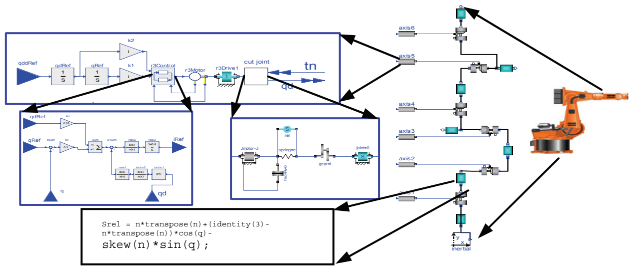
1.2 Modelica

Modelica [2][3] is a modern language for equation-based object-oriented mathematical modeling primarily of physical systems. Several tools, ranging from open-source such 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 in-put/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.

A component may internally consist of other connected components, i.e., hierarchical modeling as in Figure 2.



```
Srel = n*transpose(n)+(identity(3)-
n*transpose(n))*cos(q)-
skew(n)*sin(q);
```

Figure 2. Hierarchical model of an industrial robot.

1.3 SysML vs. Modelica

The System Modeling Language (SysML) has been proposed 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 without complex transformations and additional information provided for SysML models. In that respect, SysML is rather incomplete compared to Modelica.

Modelica was also 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 that 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.

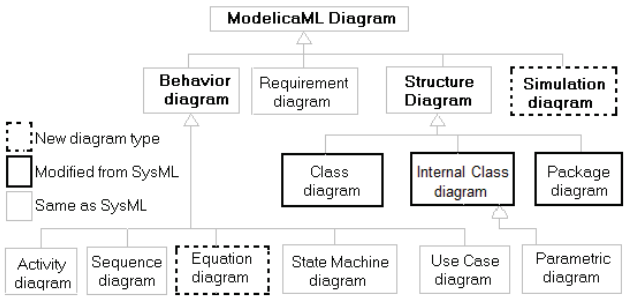


Figure 3. ModelicaML diagrams overview.

2. The ModelicaML UML profile

Before we present the Eclipse integration, we briefly describe the ModelicaML profile.

With respect to SysML, ModelicaML reuses, extends and provides several new diagrams. The ModelicaML diagram overview is shown in Figure 3. Diagrams are grouped into four categories: Structure, Behavior, Simulation and Requirement. The description of the ModelicaML profile is presented in [8], [16] and [17]. 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, variables, etc.
- Using ModelicaML diagrams it is possible to describe most of the aspects of a system being designed and thus support system development process phases such as requirements analysis, design,

implementation, verification, validation and integration.
- The profile supports mathematical modeling with equations since equations specify behavior of a system. Algorithm sections are also supported.
- Simulation diagrams are introduced to model and document simulation parameters and simulation results in a consistent and usable way.
- The ModelicaML meta-model is consistent with SysML in order to provide SysML-to-ModelicaML conversion.

2.1 Package Diagrams

ModelicaML extends SysML 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 snapshot of the Modelica Standard Library hierarchy is shown in Figure 4 using UML notation. Package nodes in the hierarchy are connected via the package containment link as in the example in Figure 4.

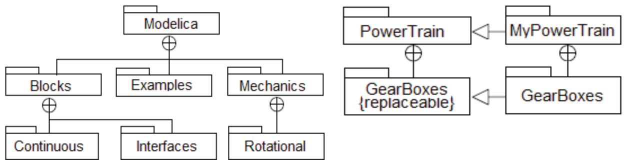


Figure 4. Package hierarchy modeling.

2.2 Modelica Class Diagrams in ModelicaML

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.

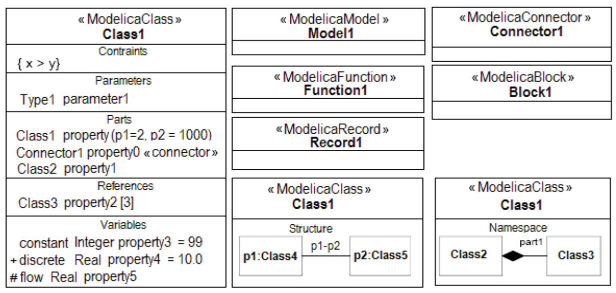
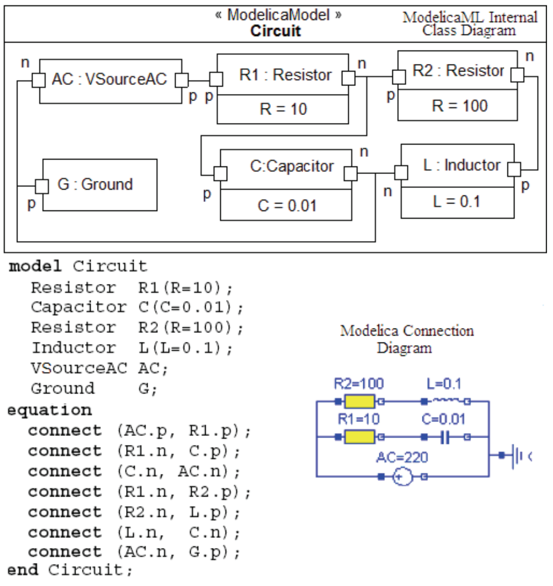


Figure 5. ModelicaML Class Diagrams.

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 ModelicaML and graphical notation of ModelicaML class definitions is shown in Figure 5. 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. 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.

The ModelicaML Internal Class Diagram is based on the SysML Internal Block Diagram. The Modelica Class Diagram defines Modelica classes and relationships between classes, like generalizations, association and dependencies, whereas a ModelicaML Internal Class Diagram shows the internal structure of a class in terms of parts and connections. The ModelicaML Internal Class Diagram is similar to the Modelica connection diagram, which presents parts in a graphical (icon) form. An example Modelica model presented as an Internal Class diagram is shown in Figure 6



```
model Circuit
  Resistor    R1(R=10);
  Capacitor   C(C=0.01);
  Resistor    R2(R=100);
  Inductor    L(L=0.1);
  VSourceAC  AC;
  Ground      G;
equation
  connect (AC.p, R1.p);
  connect (R1.n, C.p);
  connect (C.n, AC.n);
  connect (R1.n, R2.p);
  connect (R2.n, L.p);
  connect (L.n,  C.n);
  connect (AC.n, G.p);
end Circuit;
```

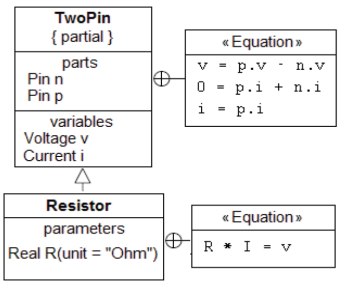
Figure 6. ModelicaML Internal Class Diagram vs. Modelica Connection Diagram.

Usually Modelica models are presented graphically via Modelica connection diagrams (Figure 6, 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 ModelicaML 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.

2.3 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.

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 a less powerful means of domain model description than Modelica equations, which also include certain kinds of equations which cannot be modeled using constraint blocks, i.e.: if-, for-, and when- equations.



```
partial class TwoPin
  Pin p, n;
  Voltage v;
  Current i;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;

class Resistor
  extends TwoPin;
  parameter Real R(unit = "Ohm");
equation
  R * I = v;
end Resistor;
```

Figure 7. Equation modeling example with a ModelicaML Class Diagram.

2.4 Simulation Diagram

ModelicaML also introduces a new diagram type, called Simulation Diagram (Figure 8), used for simulation experiment modeling. Simulation is usually performed by a simulation tool which allows parameter setting, variable selection for output and plotting. The Simulation Diagram can 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.

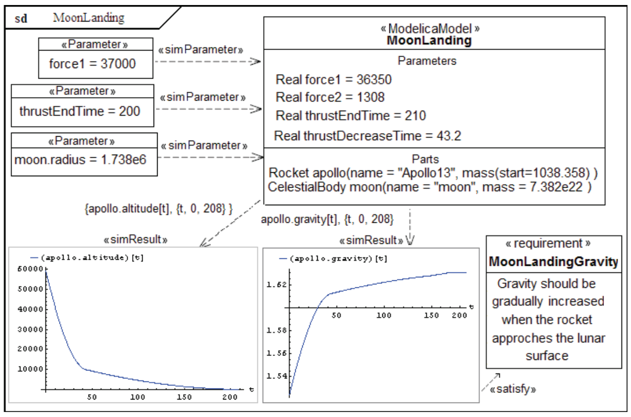


Figure 8. Simulation Diagram example.

2.5 Requirement Diagrams

Requirement diagrams have been included in ModelicaML from SysML without any modification. Requirement Diagrams have several attributes: ID, Level, Status, Name and Description. Requirements support hierarchical modeling, i.e., more specific requirements can derived from more general ones.

Requirements can be linked to any other ModelicaML element via `satisfies` or `satisfiedBy` relations. Any tool implementing the ModelicaML profile can be used to build, query, trace, and manage requirements.

The Modelica language does not support a standard requirements representation. Since the ModelicaML profile supports requirements, we need a way to save these requirements in a Modelica file. Requirements can be represented in Modelica in several ways which we will describe in detail in the next section.

3. ModelicaML in Eclipse

Eclipse [14] is an open source framework for creating extensible integrated development environments (IDEs). One of the goals of the Eclipse platform is to avoid duplicating common code that is needed to implement a powerful integrated environment for the development of software. By allowing third parties to easily extend the platform via the plugin concept, the amount of new code that needs to be written is decreased. For the development of our prototype we have used several Eclipse frameworks:

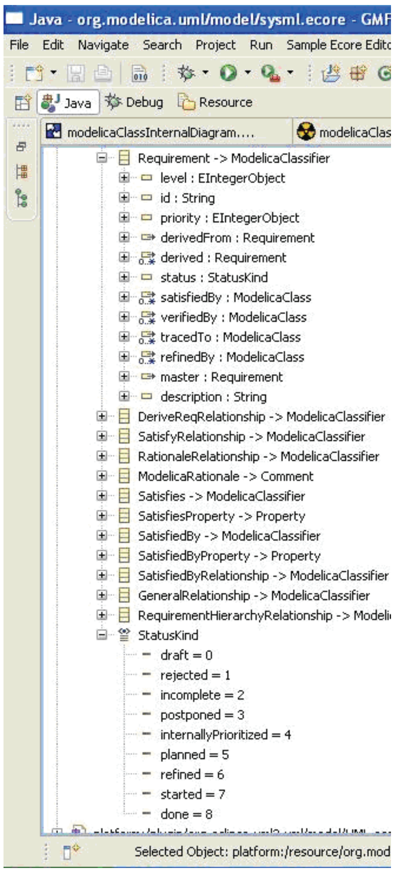
- EMF – Eclipse Modeling Framework [18] is an Eclipse framework for building domain-specific model implementations. The EMF implementation is based on the Meta Object Facility (MOF) standard and implements the "Essential MOF" (EMOF) part of a standard. EMF is used by the GMF and UML2 frameworks.
- GMF – Graphical Modeling Framework [20] provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF [19]. GMF consists of tooling, gen-

erative and runtime parts, depends on the EMF and GEF frameworks and also on other EMF related tools.

- The UML2 Eclipse meta-model implementation. The UML2 Eclipse project is an EMF based implementation of a UML2 meta-model for the Eclipse Platform to support development of UML modeling tools. The UML2 project doesn't aim to provide any graphical modeling or diagram interchange capabilities as it only implements UML abstract syntax. UML2 Tools focus on editing capabilities.

3.1 The ModelicaML GMF Model

The Eclipse editor is created from a GMF model of the ModelicaML profile. The model describes the existing elements and their properties. As an example, in Figure 9 we present the Requirement Diagram element and its properties.



Java - org.modelica.uml/model/sysml.ecore - GMF

```
Requirement -> ModelicaClassifier
    level : EIntegerObject
    id : String
    priority : EIntegerObject
    derivedFrom : Requirement
    derived : Requirement
    status : StatusKind
    satisfiedBy : ModelicaClass
    verifiedBy : ModelicaClass
    tracedTo : ModelicaClass
    refinedBy : ModelicaClass
    master : Requirement
    description : String
DeriveReqRelationship -> ModelicaClassifier
SatisfyRelationship -> ModelicaClassifier
RationaleRelationship -> ModelicaClassifier
ModelicaRationale -> Comment
Satisfies -> ModelicaClassifier
SatisfiesProperty -> Property
SatisfiedBy -> ModelicaClassifier
SatisfiedByProperty -> Property
SatisfiedByRelationship -> ModelicaClassifier
GeneralRelationship -> ModelicaClassifier
RequirementHierarchyRelationship -> Modeli
StatusKind
    draft = 0
    rejected = 1
    incomplete = 2
    postponed = 3
    internallyPrioritized = 4
    planned = 5
    refined = 6
    started = 7
    done = 8
```

Selected Object: platform:/resource/org.mod

Figure 9. ModelicaML GMF Model (Requirements)

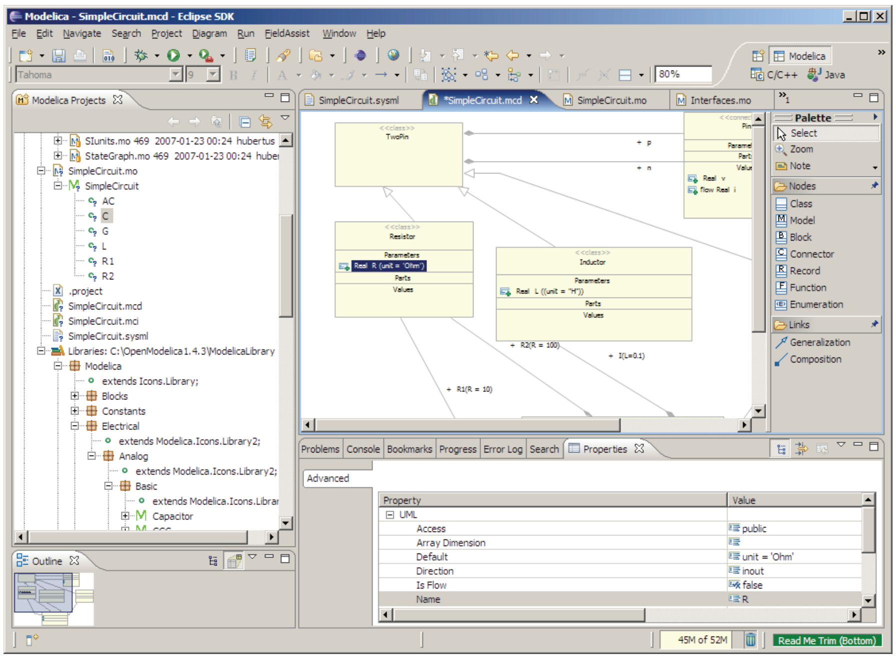
From the GMF model an editor that supports common operations on that model is automatically generated. The generated code can be extended to deal with issues specific to ModelicaML.

3.2 The ModelicaML Eclipse Editor

The Modelica Development Tooling (MDT) [7] Eclipse plugin is part of the OpenModelica system [1] and provides an environment for working with Modelica projects. The following features are available within the editor:

- Browsing support and wizards for creating Modelica projects, packages, Modelica Standard Libr.
- Syntax color highlighting, syntax and semantic checking.
- Code assistance for packages and function calls
- Support for MetaModelica meta-programming extensions to standard Modelica
- Debugging support for Modelica and MetaModelica algorithmic code.

We have extended the MDT plugin with a design view to facilitate ModelicaML integration. The ModelicaML integrated design environment where SysML/ ModelicaML diagrams are created is shown in Figure 10. It consists of a diagram file browser (left), diagram editor (middle), tool palette (right), properties editor (bottom) and a diagram outline (bottom left).



Property	Value
UML	
Access	public
Array Dimension	
Default	unit = 'Ohm'
Direction	inout
Is Flow	false
Name	R

Figure 10. ModelicaML Eclipse based design environment with a Class diagram.

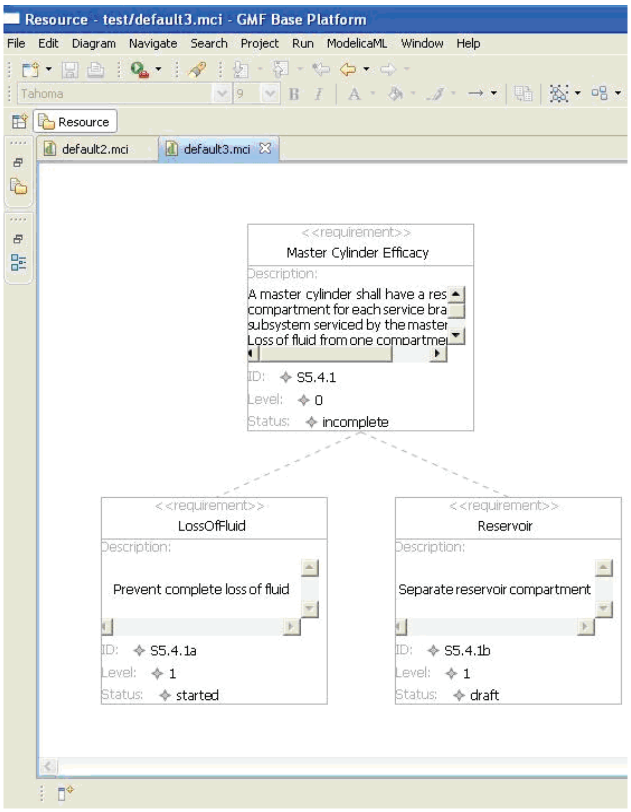
The Project Browser lists all Modelica files. The Diagram Editor is a tool where diagrams can be created and graphical elements laid out. It has the following graphical features: Graphical elements like Modelica Class, Model, or Requirement can be picked up from a Tool palette and created in a Diagram editor pane in a drag-and-drop way. Elements in a palette are grouped by Standard tools (zooming, note, etc), Nodes and Links elements. The tool palette for Modelica Class and Internal Class diagram contains different sets of elements. The Property Editor can be used for changing the properties of the object selected in the diagram editor pane. Property elements vary depending on the type of chosen object.

The ModelicaML diagrams can be automatically generated from Modelica source files. The integrated tool can also generate Modelica source code from ModelicaML diagrams. However, the implementation of the Modelica code generation and ModelicaML diagram generation is, at the moment, in an experimental stage. In the current implementation ModelicaML diagrams are saved both in Modelica form and also the XMI dialect written to XML files. Further work is needed to save diagram position information within Modelica source code as annotations.

3.3 Modeling with Requirements

The ModelicaML/MDT Eclipse environment supports modeling with requirements. The following functionality is available in the development environment:

- Hierarchies of requirements can be created.
- Requirements can be traced during the development process.
- Requirements can be queried with respect to any of their attributes.


Resource - test/default3.mci - GMF Base Platform
File Edit Diagram Navigate Search Project Run ModelicaML Window Help
Tahoma
Resource
default2.mci
default3.mci

<<requirement>>
Master Cylinder Efficacy
Description:
A master cylinder shall have a res
compartment for each service bra
subsystem serviced by the master
Loss of fluid from one compartme
ID: S5.4.1
Level: 0
Status: incomplete

<<requirement>>
LossOfFluid
Description:
Prevent complete loss of fluid
ID: S5.4.1a
Level: 1
Status: started

<<requirement>>
Reservoir
Description:
Separate reservoir compartment
ID: S5.4.1b
Level: 1
Status: draft

Figure 11. Modeling with Requirement Diagrams.

Examples of modeling with Requirement Diagrams are presented in Figure 11 and in the Appendix.

3.4 Representing Requirements in Modelica

While the default storage of ModelicaML diagrams is within XML files, our goal is to be able to save this information within normal Modelica files. Having this information within Modelica code would make it accessible to other Modelica and UML tools. Using this information tools could provide additional functionality. For example Modelica tools could display the inheritance hierarchy of a library, display the requirements for a specific class, etc.

To find the best way to encode the ModelicaML diagram information within Modelica we have experienced with several ways of encoding the Requirement Diagrams.

Using Modelica Annotations

A requirement could be saved as an annotation in the following way (the top requirement from Figure 11):

```
type RequirementStatus =
  enumeration(Incomplete, Draft, Started);

annotation(
  Requirement(
    id="S5.4.1",
    level=0,
    status=RequirementStatus.Incomplete,
    name="Master Cylinder Efficacy",
    description="A master cylinder…"));
```

The problem with Modelica annotations is that they can only be present at specific places within code and they are usually tied to a class definition. Because requirements are usually cross cutting is impossible to represent all requirements as such annotations. Another problem using annotations is the representation of hierarchies of requirements. Linking of a class with a requirement via a `satisfy` relation could be possible using Modelica `extends`, but would be cumbersome.

Creating a new Restricted Class: requirement

A requirement could be saved as a standard `class` marked with an `isRequirement` annotation or alternatively using a new restricted class in the following way (the diagrams from Figure 12 in the Appendix):

```
requirement R1
  String name="Master Cylinder Efficiency";
  String id="S5.4.1";
  Integer level=0;
  RequirementStatus status=
     RequirementStatus.Incomplete;
  String description=”A master cylinder
                      shall have…”;
end R1;
```

Now, we can use `extends` over requirements to build hierarchies:

```
requirement R2
  extends R1;
  String name"Loss Of Fluid";
  String id="S5.4.1a";
  Integer level=1;
  RequirementStatus status=
     RequirementStatus.Started;
  String description="Prevent complete
                      loss of fluid";
...
end R2;
```

To link requirements to Modelica elements one can use annotations:

```
model BreakSystem
  annotation(satisfy=R1);
...
end BreakSystem;
```

We believe that the best way to encode requirements within Modelica would be to create a new restricted class. Using this new class, requirements can be fully modeled in Modelica.

We will propose in the Modelica Association to introduce the `requirement` restricted class into the Modelica specification. In Modelica, the requirement class could also have equation or algorithm sections that impose constraints to be verified against the class linked with the requirement.

4. Conclusion and Future Work

In this paper we have presented the integration and implementation of the ModelicaML profile in Eclipse. UML Statecharts and Modelica have previously been combined, see e.g. [9][15]. SysML has already been adopted for system on chip design [13] evaluated for code generation [14], and extended with bond graphs support [12].

The support for Modelica in ModelicaML allows precise definition, specification, and simulation of 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).

Ongoing and future work targets further development and improvement of the Eclipse-based ModelicaML integrated development environment as a part of our Modelica Development Tooling (MDT).

5. Acknowledgements

This work was supported by Vinnova in the Safe & Secure Modeling and Simulation project, by SSF in the RISE project and by the CUGS graduate school at Linköping University.

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 Simu-lation 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), http://www.omgsysml.org

[5] OMG : Guide to Model Driven Architecture: The MDA Guide v1.0.1

[6] Eclipse.Org, 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 Statechars and Modelica". 7th IEEE Intl. Conf. on Emerging Technologies and Factory Automation, October 1999, Spain.

[10] MathCore, MathModelica http://mathcore.com

[11] Dynasim, Dymola, http://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.

[16] Adrian Pop, David Akhlevidiani, Peter Fritzson, Towards Unified System Modeling with the ModelicaML UML Profile, EOOLT'2007 - 1st International Workshop on Equation-Based Object-Oriented Languages and Tools, July 29-August 3, 2007, Berlin, Germany.

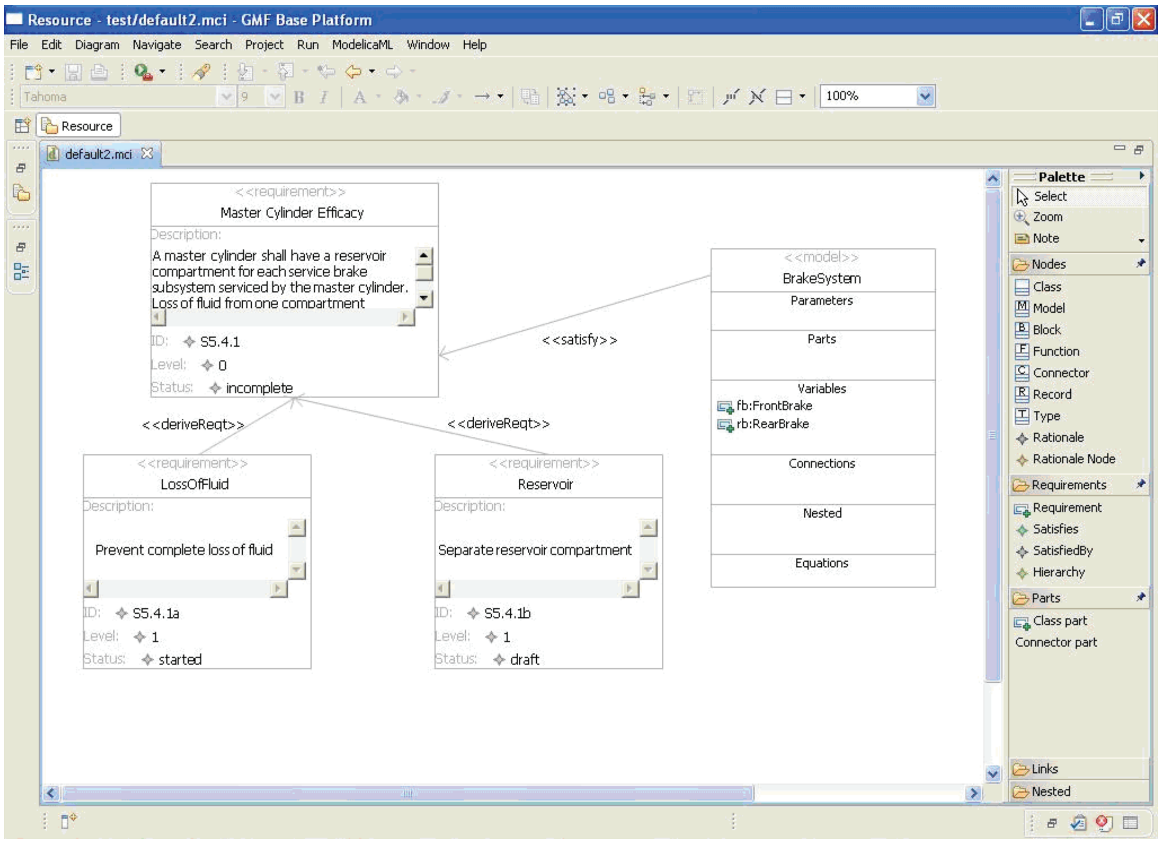
[17] Adrian Pop, David Akhvlediani, Peter Fritzson: Integrated UML and Modelica System Modeling with ModelicaML in Eclipse, 11th IASTED International Conference on Software Engineering and Applications (SEA 2007), November 19-21, 2007, Cambridge, MA, USA

[18] Eclipse.Org, Eclipse Modeling Framework (EMF), http://www.eclipse.org/emf/

[19] Eclipse.Org, Graphical Editing Framework (GEF), http://www.eclipse.org/gef/

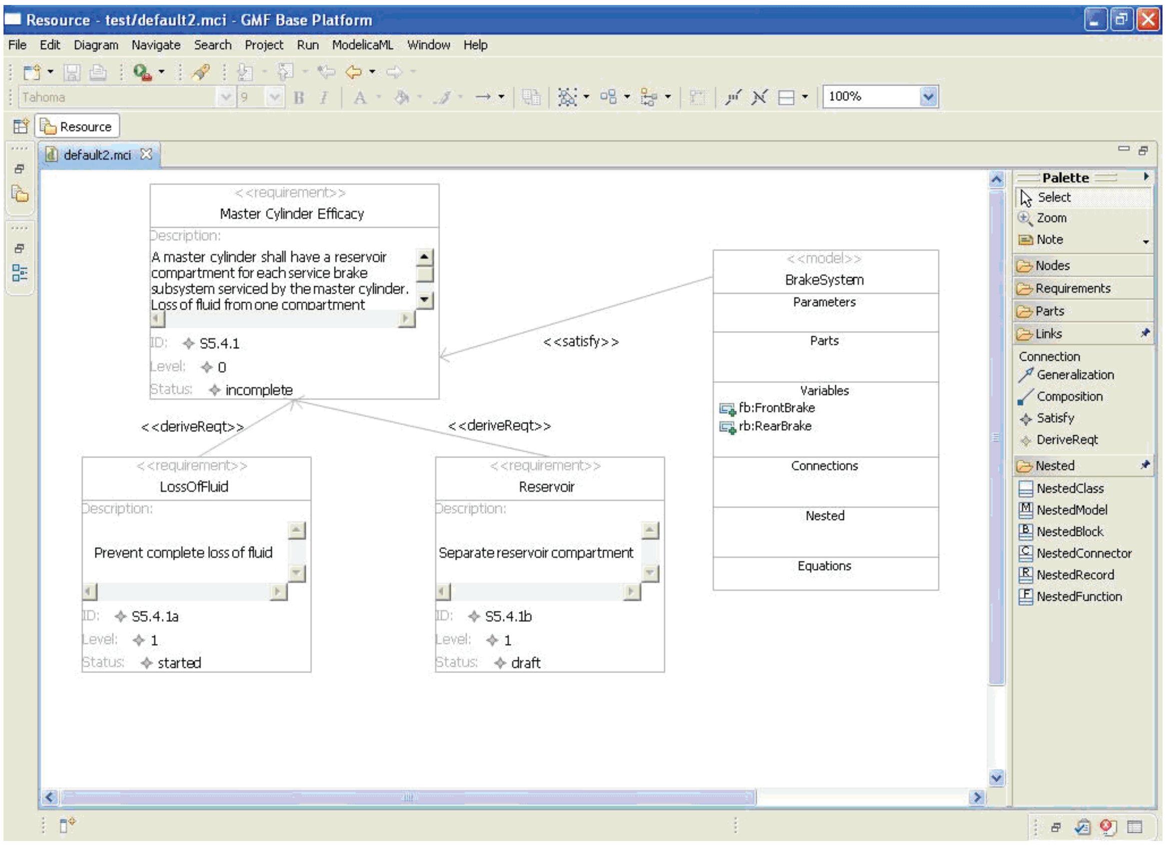
[20] Eclipse.Org, Graphical Modeling Framework (GMF), http://www.eclipse.org/gmf/

Appendix



A master cylinder shall have a reservoir compartment for each service brake subsystem serviced by the master cylinder. Loss of fluid from one compartment

Figure 12. Modeling with requirements (Requirements palette).



A master cylinder shall have a reservoir compartment for each service brake subsystem serviced by the master cylinder. Loss of fluid from one compartment

Figure 13. Modeling with requirements (Connections).