MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica

FAN-LI Zhou, LI-PING Chen, YI-ZHONG Wu, JIAN-WAN Ding, JIAN-JUN Zhao, YUN-QING Zhang
CAD Center, Huazhong University of Science and Technology, China
fanli.zhou@gmail.com, chenlp@hustcad.com, cad.wyz@gmail.com, jwdingwh@gmail.com, zhaojj@hustcad.com, zhangyq@hustcad.com

Abstract

MWorks is a platform for modeling and simulation of multi-domain physical systems. It is a modern integrated development environment (IDE) integrating with visual modeling, translator, optimizer, solver and postprocessor. MWorks implements all the syntax and most semantics of Medelica 2.1.

This paper first describes the features of MWorks as a modern IDE, and then gives the detailed descriptions of special Modelica semantic implementation in translator and self-adapting solving strategies in solver of MWorks.

Keywords: IDE; Modeling and Simulation; Multi-domain Physical Systems; Modelica

1 Introduction

MWorks is a general modeling and simulation platform for complex engineering systems which supports visual modeling, automatically translating and solving, as well as convenient postprocessing. The current version is based on Modelica 2.1 and implements all the syntax and most semantics of Modelica.

MWorks has features as follows:

a. With modern integrated development environment styles, it provides friendly user interfaces such as syntax high-lighting, code assist etc.;
b. Based on object-oriented compiler framework, it perfectly supports almost all the syntax and semantics of Modelica;
c. Using self-adapting solving strategies, it can agilely solve differential equations, algebraic equations and discrete equations.

The version 1.0 of MWorks will completely support Modelica 2.2, and the current version is 0.8 which realizes the most semantics of Modelia 2.1.

2 Framework of MWorks

MWorks is a general modeling and simulation platform which consists of studio, translator, optimizer, solver and postprocessor.

The main process is similar to the described in book of Peter Fritzson [1], shown in Figure 1.

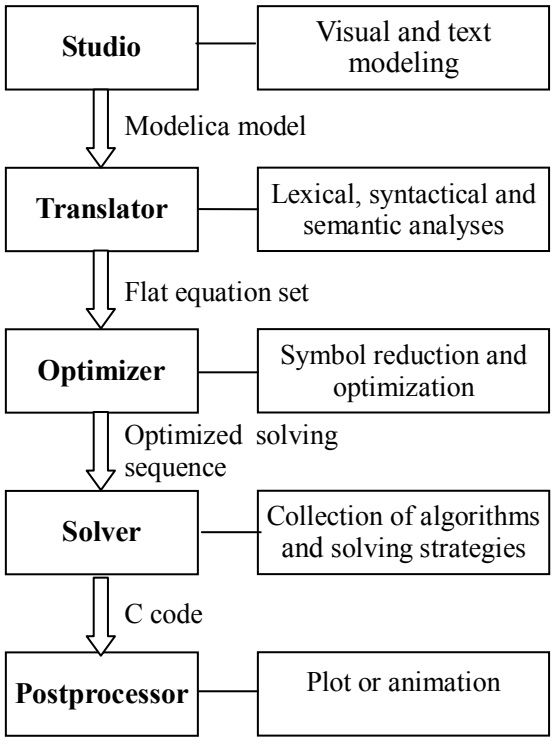


Figure 1. Main process of MWorks

The studio is an integrated development environment which integrates the visual modeling interface with other modules of MWorks. The translator is a compiler of Modelica by which an equation system of model will be generated after lexical, syntactical and semantic analyses. The optimizer completes symbol reduction based on graph theory and gives an

optimized solving sequence. The C code of the model will be emitted after compiling, analyzing and optimizing. The simulation of model is driven by the C code through calling the solver. The solver includes the collection of algorithms for algebraic equations, ordinary differential equations, differential-algebraic equations and discrete equations. Its core is the self-adapting solving strategies. The result of the simulation is displayed on postprocessor in a plot or animation way.

3 MWorks Studio: Visual Modeling Environment and IDE

MWorks Studio is a visual modeling environment which supports drag-drop modeling based on Modelica Standard Library. It is also an integrated development environment integrating with translator, optimizer, solver and postprocessor.

As a developing tool, this studio provides many modern IDE styles to promote the users' conveniences just as Eclipse or Microsoft Visual Studio does, such as real-time syntax highlighting, content assist, code formatting, outlining etc..

The snapshot of MWorks Studio is shown as Figure 2.

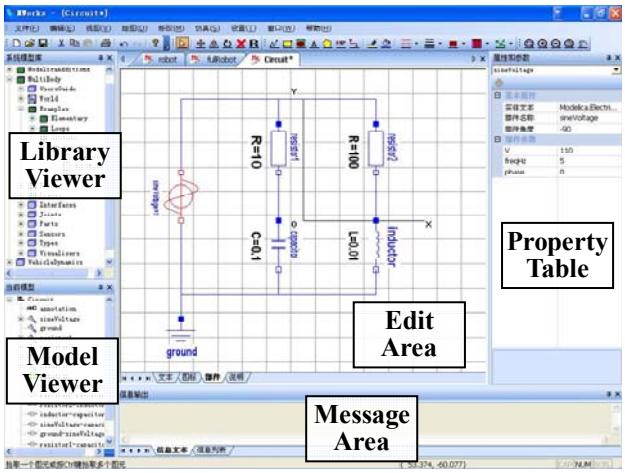


Figure 2. Snapshot of MWorks Studio

The library viewer illustrates all predefined system libraries, all loaded user libraries and other top models in the memory. The model viewer shows all components of the current model. The edit area is visual modeling, text modeling, icon-editing or information area, and the status is chosen by tag. The property table displays all properties of selected element in the model, and the properties can be edited here. The message area displays all messages in the checking, translating, or simulating, including status and error messages. The error can automatically be located by double clicking error message.

The auxiliary functions of real-time syntax highlighting, content assist, code formatting and outlining are provided in the text modeling status.

4 MWorks Translator: Modelica Compiler and Equations Generator

The tasks of translator are to perform lexical, syntactical and semantic analyses for the model files and generate equation systems for the models. They are accomplished by three time parsing: lexical and syntactical parsing, resolving of model and instantiation of model.

The design and implementation of translator are based on object-oriented framework, which is obtained in the first parsing. The designs of all the semantic mechanisms, which are implemented in the second and third parsing, are also based on the framework to perfectly support Modelica.

4.1 Three Times Parsing

Three times parsing should be done for a complete translating of the main model for simulaton in MWorks.

4.1.1 Lexical and Syntactical Analysis

The lexical and syntactical analysis is performed in the first parsing by using ANTLR tool. The ANTLR is a convenient, object-oriented, automotive lexical and syntactical analysis tool [2]. The result of the first parsing is Document Object Model (DOM) tree that is object-oriented container presentation of Abstract Syntax Tree (AST). The class hierarchy of DOM in MWorks is shown as Figure 3.

The class hierarchy of DOM is abstracted from the EBNF description of Modelica language specification (MLS) [3]. Each class in the hierarchy is consistent with the corresponding element in the EBNF. Three main class groups are noticeable in the hierarchy. Element, expression and behavior are respective abstract base class of the three groups. All operations of the translator are based on the DOM, from semantic checking to generating equation system.

4.1.2 Semantic Resolving

The semantic resolving based on DOM tree is the main content of the second parsing, including collecting information for checking types and resolving extends clauses, modifications (general modifica-

tions and redeclarations), outer-inner matches, connect clauses, and so on. The realization of UBD (use before declaration) is easy in the second parsing based on DOM.

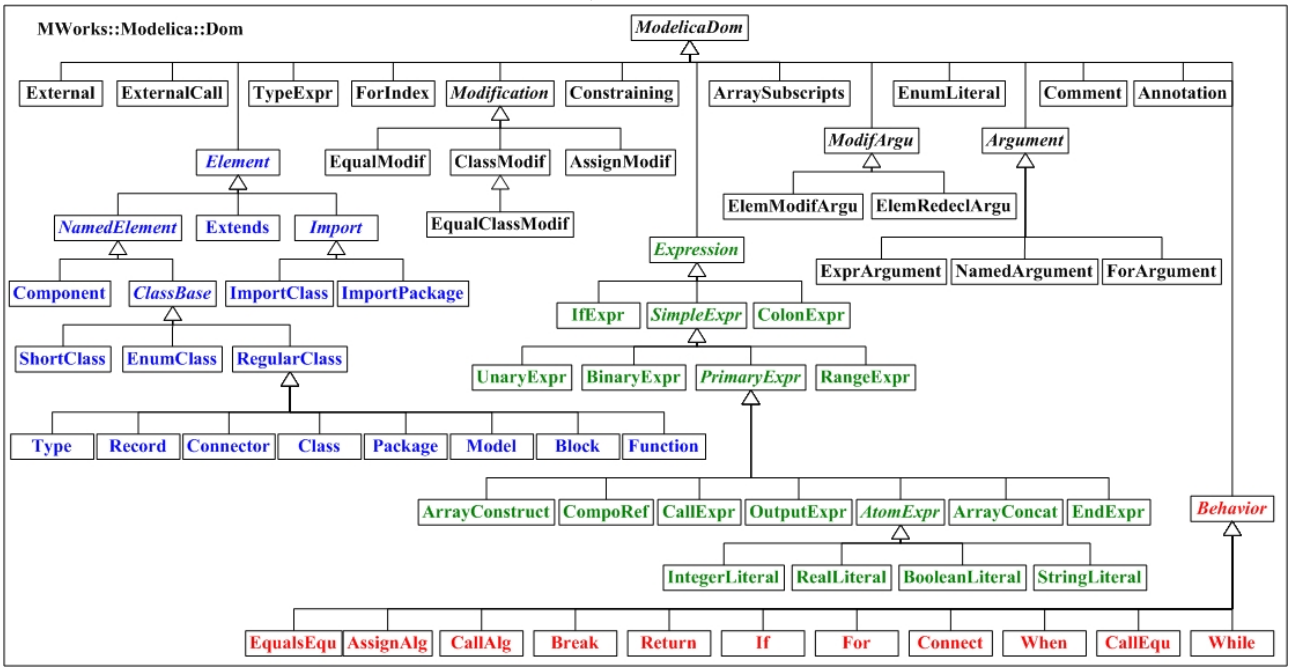


Figure 3. Class hierarchy of DOM

The implementation of semantic resolving is mainly concentrated into three classes in DOM: RegularClass, Component and ShortClass. Their resolving algorithms are as follows.

***Algorithm 1**: Resolving of Regular Class*

1. Checking circular type definition (circular inheritance);

2. Validating type prefixes;

3. Resolving outer-inner (matching inner element for outer element);

4. Resolving extends clauses (looking up the base class);

5. Checking repeated names of the named elements;

6. Resolving member components (calling resolving of component);

7. Checking type restriction;

8. Deducing type prefixes;

9. Resolving modifications (collecting information for modifications, including redeclaratoins);

10. Resolving connections (collecting information for generating connection equations).

The nested classes are not resolved in the resolving of regular class, and they are resolved when they are used as types, such as component types, reference types of short classes or base classes.

***Algorithm 2**: Resolving of Component*

1. Validating component prefixes;

2. Looking up type of the component;

3. Resolving the type if unresolved (calling resolving of type);

4. Checking the validity of the type;

5. Considering redeclaration of the type;

6. Resolving type expression of the component (expanding the type of the component type);

7. Deducing component prefixes;

8. Resolving modifications (collecting information for modifications, including redeclaratoins).

The resolving of component is called when resolving its parent regular class.

***Algorithm 3:** Resolving of Short Class*

1. Checking circular type definition (circular reference);

2. Validating type prefixes;

3. Looking up reference type;

4. Resolving the type if unresolved (calling resolving of type);

5. Considering redeclaration of the type;

6. Resolving type expression of the short class (expanding the type of the short class);

7. Deducing type prefixes;

8. Resolving modifications (collecting information for modifications, including redeclaratoins).

The resolving of short class is called when resolving the element that uses the short class as type of

component, reference type of short class or base class of regular class.

4.1.3 Instantiation

The semantic resolving prepares necessary information for instantiation which is performed in the third parsing. The purpose of instantiation is mainly to generate equation system (continuous equations and discrete events) for the main model.

Similar to the resolving, the implementation of instantiation is concentrated into the same three classes in DOM. The algorithms are as follows.

Algorithm 4: *Instantiation of Regular Class*

1. Merging modifications (defining valid modifiers, including redeclarations);

2. Instantiating base classes (calling instantiation of the base class);

3. Instantiating member components (calling instantiation of the component);

4. Instantiating equation clauses to generate equations;

5. Instantiating algorithm clauses to generate equations or definition of function (if regular class is function the instantiation should be handled specially).

According to the regular class being container of elements, the instantiation of regular class is the console of calling all element instantiations. Additionally, it generates the direct equations of the regular class itself.

Algorithm 5: *Instantiation of Component*

1. Dealing with component prefixes;

2. Looking up redeclarated component or type if redeclaration is valid;

3. Looking up inner type if the component type is outer;

4. Evaluating subscripts of component type if the component is an array;

5. Merging modifications (defining valid modifiers, including redeclarations);

6. Generating variables for built-in component (considering array) or calling instantiation of type for complex component.

The instantiation of component is mainly to generate variables for built-in component or to call the instantiation of type for complex component.

Algorithm 6: *Instantiation of Short Class*

1. Looking up inner type if the reference type is outer;

2. Merging modifications (defining valid modifiers, including redeclarations);

3. Instantiating reference type (calling instantiation of the reference type).

The instantiation of short class is mainly to call the instantiation of the reference type.

4.2 Lookup Mechanism

Lookup is the most basic mechanism for Modelica translating. The type resolving of component declarations, extends clauses and import clauses depends on standard static lookup, and the outer-inner matching depends on dynamic lookup. Besides both of them, the special lookup mechanisms are necessary for supporting modification and array. All of the lookup mechanisms are implemented as appropriate interfaces in DOM class hierarchy.

The static lookup mechanism, which is described in the Modelica language specification, is induced as two virtual interfaces of DOM class NamedElement and one top static interface of the root context class. The two virtual interfaces are lookup_type() and lookup_comp(), and the static interface is lookup_type_from_top(). The interface lookup_type() is used in the lookup of component type, base type or reference type of short class; The interface lookup_comp() is called in the lookup of component in expression or modification; The interface lookup_type_from_top() is specially designed for the lookup of import package or class.

The dynamic lookup, i.e. matching of outer and inner, is very complicated because of the freedom of the situation and kind of the outer element. The outer element may appear anywhere in model, and may be type or component. The implementation of the match depends on stack of both of type and component in the resolving and instantiation of the main model.

Special lookup mechanisms are imported to support the resolving of modification and the evaluation of array subscripts. The imported interface is to lookup replaced type or component for resolving modification and lookup modifier in the modifier container for evaluating value of array subscripts. The modifier container is built in the resolving of modification. Not only array subscripts but also all parameters, their values are evaluated by looking up modifier when the evaluation is necessary.

According to these lookup mechanisms, many Modelica semantics get perfectly implemented, such as circular extends check and complex redeclaration semantics.

4.3 Modelica Semantic Mechanisms

To support modeling and simulation of complex engineering physical systems, Modelica defines complex semantics, and it can be abstracted as the following basic mechanisms: type, extends, general modification (except redeclaration), redeclaration, outer-inner, connect, array and algorithm mechanism.

Some of these mechanisms are very complex, especially, some of them may cause couple effects. For examples, the couple influence of redeclaration and out-inner must be considered everywhere in implementing translator.

The core of the type mechanism is type checking, in which the short class and array should be considered. The type resolving is done in the second parsing and the type checking is finished in the third parsing.

The extends mechanism is easily realized based on lookup mechanism.

The implementation of modification is done by collecting modification to build modifier container in the resolving and merging modifiers to define valid modifiers in the instantiation.

The process of matching of outer-inner is similar to the modification, which is realized by operations in both resolving and instantiation.

It is handled according to the steps described in Modelica specification to translate connect clauses to direct equations.

Array is also very complicated. Comparing with general programming languages, Modelica allows variables as subscripts, and must translate the array to single equation for solving, especially, it allow applying modification to array. If no constraints were set to modification of array, the compiling would be extremely complicated, such as modification to array subscripts. MWorks chooses appropriate simplification just like Dymola does.

Algorithm in model is not same as equation, and the flow analysis should be done for its generating equations. Though function is a class with special algorithm, but generating equations for the algorithms in function is thoroughly different from the algorithms in model. The functions are directly translated into C functions.

Based on DOM class hierarchy, these mechanisms have been dealt with well, and the couple mechanisms are also considered.

5 MWorks Solver: Collection of Algorithms and Console of Solving Strategies

The solver of MWorks includes two primary modules: collection of algorithms and console of solving strategies. In fact, the optimizer shown as Figure.1 should be a part of the solver. Here it is skipped, and its implementation for MWorks is described in the paper [4].

Solver provides different basic algorithm alternatives for users to select appropriate one. For examples, users can select one-step method series or multi-step method series for ODE/DAE problems. The different basic algorithms for differential equation and algebraic equations are collected in solver.

The consistent interface framework is designed for solver to conveniently call different basic algorithms. Each basic algorithm can be easily integrated into the solver by simply encapsulating it according to the template. Now, a series of algorithms for different kinds of equations have been collected in the solver, such as SUNDIALS [5].

It is the task of console of solving strategies to solve continuous-discrete hybrid problem. Solver controls the solution of problem based on basic algorithms according to information collected in translator.

The solving of continuous-discrete hybrid problem in Modelica is according to the principle of synchronous data flow [6]. All the events are collected in the third parsing, and solver will monitor them in the simulation.

6 Example

Here an example is given to show the visual modeling and simulation in MWorks. The visual model of the example is shown as Figure.1, and the text model is as follows (annotations are skipped),

```
model Circuit
  Modelica.Electrical.Analog.Sources.SineVoltage AC(V=110,
freqHz=5);
  Modelica.Electrical.Analog.Basic.Ground G;
  Modelica.Electrical.Analog.Basic.Resistor R1(R=10);
  Modelica.Electrical.Analog.Basic.Resistor R2(R=100);
  Modelica.Electrical.Analog.Basic.Capacitor C(C=0.01);
  Modelica.Electrical.Analog.Basic.Inductor L(L=0.1);
equation
  connect(AC.p, R1.p);
  connect(R2.p, R1.p);
  connect(R2.n, L.p);
```

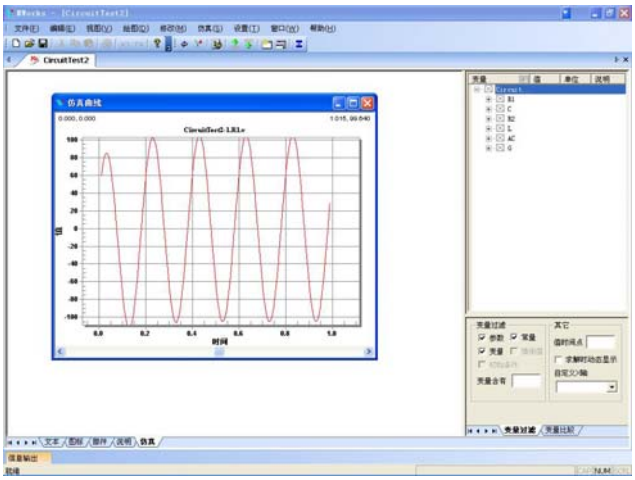
```
  connect(L.n, C.n);
  connect(AC.n, C.n);
  connect(G.p, AC.n);
  connect(R1.n, C.p);
end Circuit;
```

The following is the equations generated by the translator. There are 34 variables and 34 equations excluding parameters and constants.

```
model Circuit
  parameter Real AC.offset = 0;
  parameter Real AC.startTime = 0;
  parameter Integer AC.signalSource.nout = 1;
  parameter Integer AC.signalSource.outPort.n = AC.signalSource.nout;
  parameter Real AC.signalSource.amplitude[1] = AC.V;
  parameter Real AC.signalSource.freqHz[1] = AC.freqHz;
  parameter Real AC.signalSource.phase[1] = AC.phase;
  parameter Real AC.signalSource.offset[1] = AC.offset;
  parameter Real AC.signalSource.startTime[1] = AC.startTime;
  constant Real AC.signalSource.pi = Modelica.Constants.pi;
  parameter Real AC.signalSource.p_amplitude[1] = AC.signalSource.amplitude[1]*1;
  parameter Real AC.signalSource.p_freqHz[1] = AC.signalSource.freqHz[1]*1;
  parameter Real AC.signalSource.p_phase[1] = AC.signalSource.phase[1]*1;
  parameter Real AC.signalSource.p_offset[1] = AC.signalSource.offset[1]*1;
  parameter Real AC.signalSource.p_startTime[1] = AC.signalSource.startTime[1]*1;
  parameter Real AC.V = 110;
  parameter Real AC.phase = 0;
  parameter Real AC.freqHz = 5;
  parameter Real R1.R = 10;
  parameter Real R2.R = 100;
  parameter Real C.C = 0.01;
  parameter Real L.L = 0.1;

  Real AC.v;
  Real AC.i;
  Real AC.p.v;
  Real AC.p.i;
  Real AC.n.v;
  Real AC.n.i;
  Real AC.signalSource.outPort.signal[1];
  Real AC.signalSource.y[1];
  Real G.p.v;
  Real G.p.i;
  Real R1.v;
  Real R1.i;
  Real R1.p.v;
  Real R1.p.i;
  Real R1.n.v;
  Real R1.n.i;
  Real R2.v;
  Real R2.i;
  Real R2.p.v;
  Real R2.p.i;
  Real R2.n.v;
  Real R2.n.i;
  Real C.v;
  Real C.i;
  Real C.p.v;
  Real C.p.i;
  Real C.n.v;
  Real C.n.i;
  Real L.v;
  Real L.i;
  Real L.p.v;
  Real L.p.i;
  Real L.n.v;
  Real L.n.i;

equation

  AC.v=AC.p.v-AC.n.v;
  0=AC.p.i+AC.n.i;
  AC.i=AC.p.i;
  AC.signalSource.y[1] = AC.signalSource.outPort.signal[1];
  AC.signalSource.outPort.signal[1]=AC.signalSource.p_offset[1]+(if time<AC.signalSource.p_startTime[1] then 0 else AC.signalSource.p_amplitude[1]*Modelica.Math.sin(2*AC.signalSource.pi*AC.signalSource.p_freqHz[1]*(time-AC.signalSource.p_startTime[1])+AC.signalSource.p_phase[1]));
  AC.v=AC.signalSource.outPort.signal[1];
  G.p.v=0;
  R1.v=R1.p.v-R1.n.v;
  0=R1.p.i+R1.n.i;
  R1.i=R1.p.i;
  R1.R*R1.i=R1.v;
  R2.v=R2.p.v-R2.n.v;
  0=R2.p.i+R2.n.i;
  R2.i=R2.p.i;
  R2.R*R2.i=R2.v;
  C.v=C.p.v-C.n.v;
  0=C.p.i+C.n.i;
  C.i=C.p.i;
  C.i=C.C*der(C.v);
  L.v=L.p.v-L.n.v;
  0=L.p.i+L.n.i;
  L.i=L.p.i;
  L.L*der(L.i)=L.v;
  R1.p.v = AC.p.v;
  R2.p.v = AC.p.v;
  R2.n.v = L.p.v;
```

C.n.v = AC.n.v;
G.p.v = AC.n.v;
L.n.v = AC.n.v;
R1.n.v = C.p.v;
AC.p.i+R1.p.i+R2.p.i = 0;
L.p.i+R2.n.i = 0;
AC.n.i+C.n.i+G.p.i+L.n.i = 0;
C.p.i+R1.n.i = 0;
end Circuit;

The result is displayed in the postprocessor, shown as Figure. 4.



7 Conclusions

MWorks is a modern IDE for modeling and simulation of multi-domain physical systems based on Modelica. All the syntax and most semantics of Modelica 2.1 have been implemented. The current version of MWorks can validly deal with some problems based on Modelica Standard Library. The coming version 1.0 of MWorks will completely support Modelica 2.2.

References

[1] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press, 2003.

[2] http://www.antlr.org/

[3] http://www.modelica.org/documents/ModelicaSpec21.pdf

[4] Ding Jianwan, Chen Liping, Zhou Fanli. A Component-based Debugging Approach for Detecting Structural Inconsistencies in Declarative Equation based Models. Journal of Computer Science & Technology, 2006, 21(3):450-458

[5] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, C. S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. ACM Transactions on Mathematical Software, 2005, 31(3):363-396.

[6] Otter M., Elmqvist H., Mattsson S.E.. Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. 1999 IEEE Symposium on Computer-Aided Control System Design, CACSD'99, Hawaii, August 22-27, 1999:151-157.