Classification and Evaluation of Features in Advanced Simulators

F. Breitenecker[1], N. Popper[2]
[1]Vienna University of Technology, Vienna, Austria; [2]DWH Simulation Services, Vienna, Austria

Corresponding Author: Felix Breitenecker, Vienna University of Technology, Inst. f. Analysis and Scientific Computing Wiedner Hauptstrasse 8 - 10, 1040 Wien, Austria; `Felix.Breitenecker@tuwien.ac.at`

Abstract. This contribution elaborates, classifies and compares features of modern simulation systems. First a short overview on the CSSL standard is given, with discussion of standard and extended features for CSSL simulators, and the *ARGESIM Benchmarks - Benchmarks for Modelling and Simulation Techniques* are introduced. . Then main emphasis is put on the developments in the last decade: object-oriented approaches, acausal modelling, physical modelling, structural dynamic systems, modelling standardisations as Modelica and VHDL-AMS, and impacts from computer engineering (e.g. statecharts). Based on solutions of the *ARGESIM Benchmarks*, finally classification and comparison of structural features in modern simulators is given. This classification incorporates also the new development with MATLAB's new physical modelling language Simscape.

1 Classical features of simulators

Development of simulation languages, simulators, simulation systems, etc. is essentially influenced by the CSSL Standard 1968. Although forty years old, the structures defined in CSSL Standard are used up to now. End of 90ties, CSSL extended to implicit systems, while a new modelling language, Modelica, was introduced. In principle, the modelling paradigm changed from signal flow – oriented modelling (explicit systems) to power – oriented modelling (implicit systems), from 'causal' signal modelling to 'acausal' physical modelling. The early CSSS standard determined basic necessary features for a simulator, the late developments to implicit systems fixed extended features for simulation systems – both referred as classical CSSL features.

1.1 CSSL structure for simulation languages and simulation systems

In 1968, the CSSL standard set first challenges for features of simulation systems, defining necessary basic features for simulators and a certain structure for simulators (Figure 1).

The CSSL standard suggests structures and features for a model frame and for an experimental frame. This distinction is based on Zeigler's concept of a strict separation of these two frames. Model frame and experimental frame are the user interfaces for the heart of the simulation system, for the simulator kernel or simulation engine. A translator maps the model description of the model frame into state space notation, which is used by the simulation engine solving the system governing ODEs, whereby the model description at least must be sorted and reformulated as program code function, which can be called from an ODE – solver. The model sorting capability (MS) is an essential classical feature of simulators. The basic structure of a simulator is illustrated in Figure 1; an extended structure with service of discrete elements is given in Figure 2.

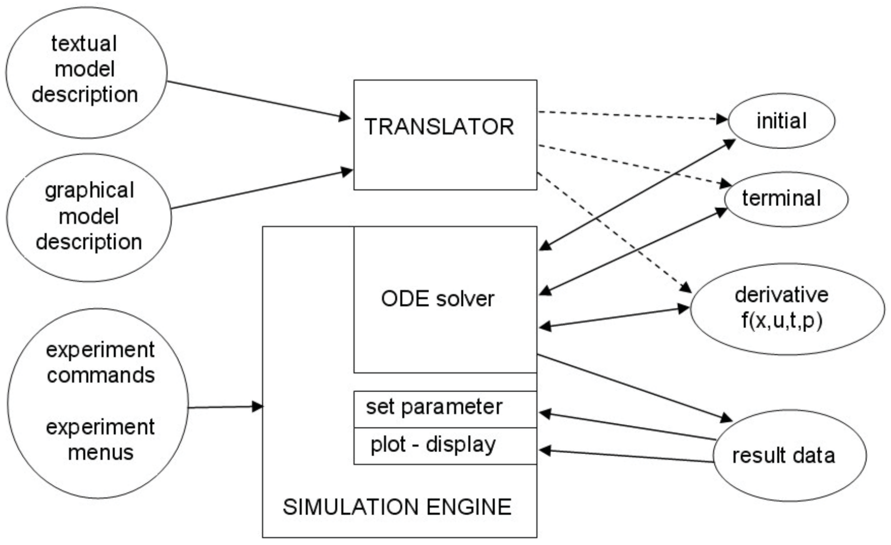


Figure 1. Basic structure of a simulation language due to CSSL Standard.

Between 1980 and 2000 developers put main emphasis on integration of discrete elements into continuous simulation systems, from simple time events to complex state events, and on extending the model description to DAEs. Both extensions are related, because algebraic equations are mainly caused or causing state events by means of state constraints.

While event description (ED) and time event handling (TEH) can be seen as basic feature, state event handling (SEH) and DAE support by means of direct or iterative DAE solvers (DAE) with or without index reduction (IR) became desirable *Extended Features* of simulators, supported directly or indirectly – features to be discussed in more details in the next subsections.

1.2 Implicit Models – Differential-Algebraic Equations – DAE Solvers

For a long time the explicit state space description

$$\dot{\vec{x}}(t)=\vec{f}(\vec{x}(t),\vec{u}(t),t,\vec{p}),\quad \vec{x}(t_0)=\vec{x}_0$$

played the dominant role; additional constraints and implicit models had to be transformed 'manually'. From the 1990s on, the simulators started to take care on these very natural phenomena of implicit structures. Consequently, they started to deal with implicit state space descriptions and constraints, in general with so-called DAE models (differential algebraic equations):

$$F(\dot{\vec{y}}(t),\vec{y}(t),\vec{u}(t),t,\vec{p})=\vec{0}\quad \vec{y}(t_0)=\vec{y}_0$$

The so-called extended state vector $\vec{y}(t)$ can be splitted into the differential state vectors $\vec{x}(t)$ and into the algebraic state vector $\vec{z}(t)$:

$$\begin{aligned}\dot{\vec{x}}(t)&=\vec{f}(\vec{x}(t),\vec{z}(t),\vec{u}(t),t,\vec{p})=0,\quad x(t_0)=\vec{x}_0,\\ g(\vec{x}(t),\vec{z}(t),\vec{u}(t),t,\vec{p})&=0\end{aligned}$$

The above given DAEs can be solved by extended ODE solvers and by implicit DAE solvers. Three different approaches may be used:

1. *Nested Approach*, using classical ODE solver
   a. given x_n, solving first numerically $g(x_n,z_n)=0\Rightarrow z_n=z_n(x_n)=\hat{g}^{-1}(x_n)$,
      e. g. by modified Newton iteration, and
   b. applying ODE method, evolving $x_{n+1}=\Phi_E(x_n,z_n(x_n),t_n)$.
2. *Simultaneous Approach*, using an implicit DAE solver;
   given x_n, solving $g(x_{n+1},z_{n+1})=0$ and $\Phi_I(x_{n+1},x_n,z_{n+1},t_{n+1})=0$ simultaneously.
3. *Symbolic Approach*, determining in advance the explicit form solving
   $g(x,z)=0\Rightarrow z=z(x)=\hat{g}^{-1}(x)$
   by symbolic computations e.g. within the model translator, and using classical ODE solvers.

The *Symbolic Approach* requires a symbolic inversion of the algebraic equations, which in many cases is not possible or not adequate; furthermore the model translator must not only sort equations, it must be able to perform symbolic manipulations on the equations.

The *Nested Approach* – up to now most commonly used – requires a numerical inversion of the algebraic equations: each evaluation of the vector of derivatives (called by the ODE solver) has to start an iterative procedure to solve the algebraic equation. This approach can be very expensive and time-consuming due to these inner iterations. Here classical ODE solvers can be used.

The *Simultaneous Approach* requires an implicit ODE solver – usually an implicit stiff equation solver. Although also working with iterations, these solvers show much more efficiency and provide more flexibility for modelling (DASSL, IDA-DASSL, and LSODE – solvers).

However, hidden is another problem: the 'DAE index' problem. Roughly speaking, a DAE model is of index *n*, if n differentiations of the DAE result in an ODE system (with an increased state space). The implicit ODE solvers for the *Simultaneous Approach* guarantee convergence only in case of DAE index $n=1$. Models with higher DAE index must / should be transformed to models with DAE index $n=1$. This transformation is based on symbolic differentiation and symbolic manipulation of the high index DAE system, and there is no unique solution to this index reduction. The perhaps most efficient procedure is the so-called *Pantelides Algorithm.* Unfortunately, in case of mechanical systems modelling and in case of process technology modelling indeed DAE models with DAE index $n=3$ may occur, so that index reduction may be necessary. Index reduction is a new challenge for the translator of simulators, and still point of discussion.

In graphical model descriptions, implicit model structures are known since long time as algebraic loops: the directed graph of signals has one or more signal feedback loops without any memory operator (integrator, delay, etc). Again, in evaluating the problem of sorting occurs, and the model translator cannot build up the sequence for calculating the derivative vector. Some simulators, e.g. SIMULINK, recognise algebraic loops and treat them as implicit models. When a graphical model contains an algebraic loop, SIMULINK calls a loop solving routine at each time step - SIMULINK makes use of the *Nested Approach* described before. This procedure works well in case of models with DAE index $n = 1$, for higher index problems may occur. In object-oriented simulation systems, like in Dymola, physical a-causal modelling plays an important role, which results in DAEs with sometimes higher index. These systems put emphasis on index reduction (in the translator) to DAEs with index $n = 1$ in order to apply implicit ODE solvers (*Simultaneous Approach).*

1.3 Time Events and State Events

The CSSL standard also defines segments for discrete actions, first mainly used for modelling discrete control. So-called DISCRETE regions or sections manage the communication between discrete and continuous world and compute the discrete model parts.

For incorporating discrete actions, the simulation engine must interrupt the ODE solver and handle the event. For generality, efficient implementations set up and handle event lists, representing the time instants of discrete actions and the calculations associated with the action, where in-between consecutive discrete actions the ODE solver is to be called. In order to incorporate DAEs and discrete elements, the simulator's translator must now extract from the model description the dynamic differential equations (derivative), the dynamic algebraic equations (algebraic), and the events (event i) with static algebraic equations and event time, as given in Figure 2 (extended structure of a simulation language due to CSSL standard). In principle, initial equations, parameter equations and terminal equations (initial, terminal) are special cases of events at time $t = 0$ and terminal time. Some simulators make use of a modified structure, which puts all discrete actions into one event module, where CASE - constructs distinguish between the different events.

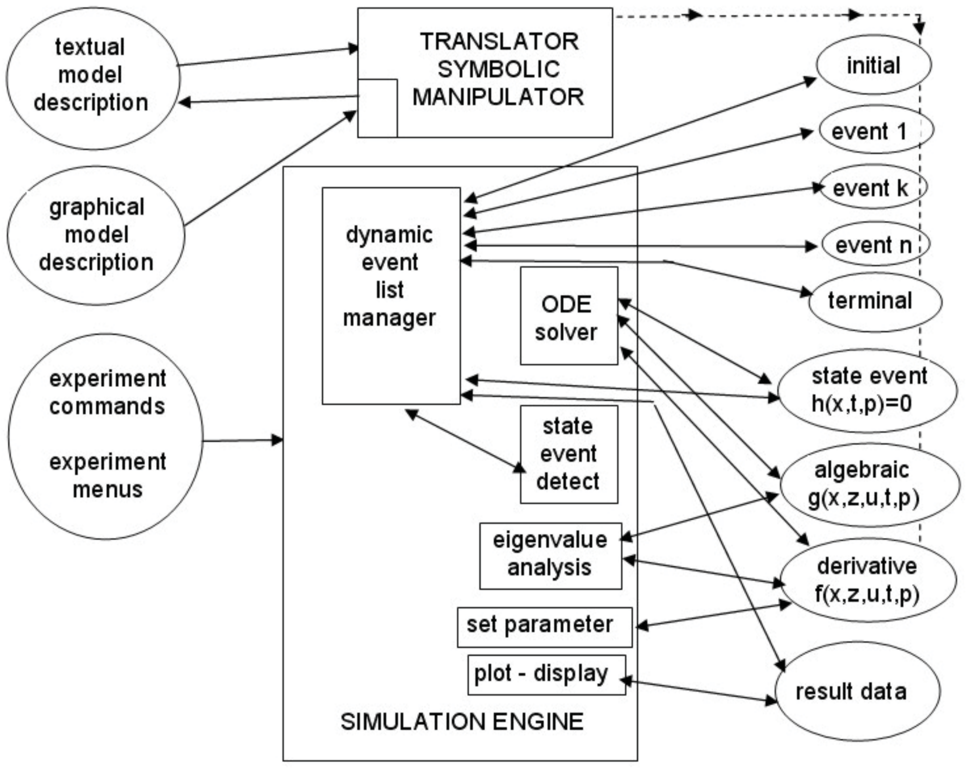


Figure 2. Extended structure of a simulation system due to extensions of the CSSL standard with discrete elements and with DAE modelling.

These so-called *time events* are known in advance, so that scheduling of the time events can be handled easily, e.g. in the same manner than simulators schedule output events. Much more complicated, but defined in CSSL, are the so-called *state events*. Here, a discrete action takes place at a time instant, which is not known in advance, it is only known as a function of the states.

For state events, the classical state space description is extended by the so-called state event function $h(\vec{x}(t), \vec{u}(t), \vec{p})$, the zero of which determines the event:

$$\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t), \vec{u}(t), \vec{p}, t), \quad h(\vec{x}(t), \vec{u}(t), \vec{p}, t) = 0$$

Generally, state events (SE) can be classified in four types:

- **Type 1** – parameters change discontinuously (**SE-P**),
- **Type 2** - inputs change discontinuously (**SE-I**),
- **Type 3** - states change discontinuously (**SE-S**), and
- **Type 4** - state vector dimension changes (**SE-D**), including total change of model equations.

State events type 1 (**SE-P**) could also be formulated by means of IF-THEN-ELSE constructs and by switches in graphical model descriptions, without synchronisation with the ODE solver. The necessity of a state event formulation depends on the accuracy wanted. Big changes in parameters may cause problems for ODE solvers with stepsize control.

State events of type 3 (**SE-S**) are essential state events. They must be located, transformed into a time event, and modelled in discrete model parts. State events of type 4 (**SE-D**) are also essential ones. In principle, they are associated with hybrid modelling: models following each other in consecutive order build up a sequence of dynamic processes. And consequently, the structure of the model itself is dynamic; these so-called structural dynamic systems are at present (2008) discussion of extensions to Modelica, see next chapters.

State events of type 2 (**SE-I**) are not really state events, they are time events. They are usually put in the list of state events, if a synchronisation of the ODE solver with an input jump should be forced.

As example, we consider the pendulum with constraints. This example is basis for detailed investigations in ARGESIM Benchmark C7 *Constrained Pendulum*. Let φ define pendulum angle, and l, m and d parameters for length, mass, and damping. If the pendulum is swinging, it may hit a pin positioned at angle φ_p with distance l_p from the point of suspension. In this case, the pendulum swings on with the position of the pin as new point of rotation. The shortened length is $l_s = l - l_p$. and the angular velocity $\dot{\varphi}$ is changed from $\dot{\varphi}$ to $\dot{\varphi} \cdot l / l_s$ at position φ_p, etc. These discontinuous changes are state events, not known in advance.

With event function notation, the model for *Constrained Pendulum* is given by

$$\dot{\varphi}_1 = \varphi_2, \ \dot{\varphi}_2 = -\frac{g}{l}\sin\varphi_1 - \frac{d}{m}\varphi_2,$$
$$h(\varphi_1, \varphi_2) = \varphi_1 - \varphi_p = 0$$

The example involves two different events: change of length parameter (**SE-P**), and change of state (**SE-S**), i.e. angular velocity).

In general, the handling of a state event requires four steps:

1. Detection of the event, usually by checking the change of the sign of *h(x)* within the solver step over $[t_i, t_{i+1}]$
2. Localisation of the event by a proper algorithm determining the time t^* when the event occurs and performing the last solver step over $[t_i, t^*]$
3. Service of the event: calculating / setting new parameters, inputs and states; switching to new equations
4. Restart of the ODE solver at time t^* with solver step over $[\ t^* = t_{i+1}, t_{i+2}]$

State events are facing simulators with severe problems. Up to now, the simulation engine had to call independent algorithms, now a root finder for the state event function *h* needs results from the ODE solver, and the ODE solver calls the root finder by checking the sign of *h*. For finding the root of the state event function *h(x)*, either interpolative algorithms (MATLAB/Simulink) or iterative algorithms are used (ACSL, Dymola).

Figure 2 (extended structure of a simulation language due to CSSL standard) also shows the necessary extensions for incorporating state events. The simulator's translator must extract from the model description additionally the state event functions (state event j) with the associated event action – only one state event shown in the figure). In the simulator kernel, the static event management must be made dynamically: state events are dynamically handled and transformed to time events. In principle, the kernel of the simulation engine has become an event handler, managing a complex event list with feedbacks. It is to be noted, that different state events may influence each other, if they are close in time – in worst case, the event finders run in a deadlock. Again, modified implementations are found. It makes sense to separate the module for state event function and the module for the associated event – which may be a single module, or which may be put into a common time event module.

In case of a structural change of the system equations (state event of type 4 – **SE-D**), simulators usually can manage only fixed structures of the state space. The technique used is to ‘freeze’ the states that are bound by conditions causing the event. In case of a complete change of equations, both systems are calculated together, freezing one according to the event. One way around is to make use of the experimental frame: the simulation engine only detects and localises the event, and updates the system until the event time. Then control is given back to the experimental frame. The state event is now serviced in the experimental frame, using features of the environment. Then a new simulation run is restarted (modelling of the structural changes in the experimental frame).

```
PROGRAM constrained pendulum
CONSTANT m = 1.02, g = 9.81, d =0.2
CONSTANT lf=1, lp=0.7
DERIVATIVE dynamics
  ddphi = -g*sin(phi)/l - d*dphi/m
  dphi  = integ ( ddphi, dphi0)
  phi   = integ ( dphi, phi0)
  SCHEDULE hit   .XN. (phi-phip)
  SCHEDULE leave .XP. (phi-phip)
END ! of dynamics

DISCRETE hit
  l = ls; dphi = dphi*lf/ls
END ! of hit

DISCRETE leave
  l = lf; dphi = dphi*ls/lf
END ! of leave

END ! of constrained pendulum
```

Listing 1.ACSL model description for *Constrained Pendulum*, dynamics and hit/release of pendulum.

The *Constrained Pendulum* example involves a state event of type 1 (**SE-P**) and type 3 (**SE-S**). A classical ACSL model description works with two discrete sections `hit` and `leave`, representing the two different modes, both called from the dynamic equations in the derivative section (Listing 1). Dymola defines events and their scheduling implicitly by WHEN – or IF - constructs in the dynamic model description (Listing 2). In case of more complex event descriptions, the WHEN – or IF – clauses are put into an ALGORITHM section similar to ACSL’s DISCRETE section.

```
  WHEN phi-phip=0
       AND phi>phip
 THEN l = ls;
      dphi = dphi*lf/ls
```

Listing 2. Dymola model description for *Constrained Pendulum*, hit of pendulum.

In graphical model descriptions, we are faced with the problem that calculations at discrete time instants are difficult to formulate. For the detection of the event, SIMULINK provides the `HIT CROSSING` block (in new Simulink version implicitly defined). This block starts state event detection (interpolation method) depending on the input, the state event function, and outputs a trigger signal, which may call a triggered subsystem servicing the event. It is to be noted, that discrete elements with time events and state events and DAEs may also change the structure of the model.

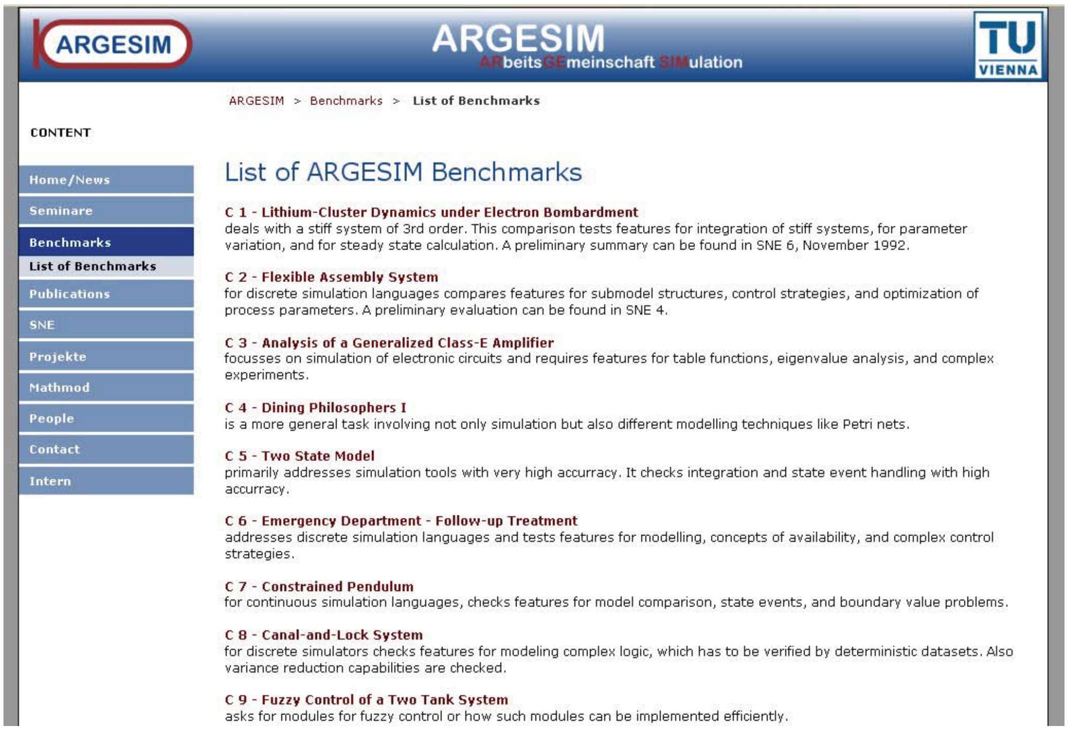
2 ARGESIM Benchmarks on Modelling and Simulation Approaches and Techniques

In 1990, the journal ***SNE – Simulation News Europe*** – started a series on *Comparison of Simulation Software*, which has been developed to *Benchmarks for Modelling and Simulation Approaches and Techniques*. Up to now, 21 comparisons and benchmarks have been defined, and about 250 solutions have been published – being a very valuable source for discussing and documenting various aspects of modelling and simulation approaches.

For the evaluation and comparing features of simulation systems and for the feature comparison documented in this contribution, the following benchmarks were mainly used:

- C 1 - Lithium-Cluster Dynamics under Electron Bombardment
- C 3 - Analysis of a Generalized Class-E Amplifier
- C 5 - Two State Model
- C 7 - Constrained Pendulum
- C 9 - Fuzzy Control of a Two Tank System
- CP1 - Parallel Comparison
- C 11 - SCARA Robot
- C 12 - Collision Processes in Rows of Spheres
- C 13 - Crane Crab with Embedded Control
- C 15 - Clearance Identification
- C 17 - Spatial Dynamics of SIR-Type Epidemic
- C 18 - Neural Networks versus Transfer Functions - Identification of Nonlinear Systems
- C 19 - Pollution in Groundwater Flow
- C 20 – Hybrid Modelling

In this contribution, examples mainly concentrate on Benchmark C5 *Constrained Pendulum*, because it is a small model and comparison results can be documented in a concentrated manner. At present (December 2008), further benchmarks are in preparation, among them *C 20 – Hybrid Modelling*. This comparison is a benchmark of new type, which not only concentrates on one specific model; C20 evaluates hybrid modelling approaches by means of different systems, from simple to more complex: bouncing ball, switching circuit, and rotor-stator dynamics. Basic idea is to compare the hybrid modelling techniques on basis of classifications discussed in this contribution in section 4.



List of ARGESIM Benchmarks

C 1 - Lithium-Cluster Dynamics under Electron Bombardment
deals with a stiff system of 3rd order. This comparison tests features for integration of stiff systems, for parameter variation, and for steady state calculation. A preliminary summary can be found in SNE 6, November 1992.

C 2 - Flexible Assembly System
for discrete simulation languages compares features for submodel structures, control strategies, and optimization of process parameters. A preliminary evaluation can be found in SNE 4.

C 3 - Analysis of a Generalized Class-E Amplifier
focusses on simulation of electronic circuits and requires features for table functions, eigenvalue analysis, and complex experiments.

C 4 - Dining Philosophers I
is a more general task involving not only simulation but also different modelling techniques like Petri nets.

C 5 - Two State Model
primarily addresses simulation tools with very high accurracy. It checks integration and state event handling with high accurracy.

C 6 - Emergency Department - Follow-up Treatment
addresses discrete simulation languages and tests features for modelling, concepts of availability, and complex control strategies.

C 7 - Constrained Pendulum
for continuous simulation languages, checks features for model comparison, state events, and boundary value problems.

C 8 - Canal-and-Lock System
for discrete simulators checks features for modeling complex logic, which has to be verified by deterministic datasets. Also variance reduction capabilities are checked.

C 9 - Fuzzy Control of a Two Tank System
asks for modules for fuzzy control or how such modules can be implemented efficiently.

Figure 3. ARGESIM / SNE webpage with definition of ARGESIM Benchmarks; www.argesim.org.

Administration and publication of benchmark definitions and benchmark solutions is done via web, via ARGE-SIM webpage and SNE webpage www.argesim.org (Figure 3).

3 Comparison of Classical-Feature availability in selected simulator

Model sorting (MS), event description (ED), time event handling (THE), state event handling (SEH), and DAE support (DAE) with or without index reduction (IR) became desirable *Classical Features* of simulators, supported directly or indirectly in simulation systems. Evaluation of ARGESIM Benchmark solutions allow to evaluate the availability of these features in simulation software. Table 1 compares the availability of these features in the MATLAB / Simulink System, in ACSL and in Dymola.

	MS - Model Sorting	ED -Event Description	TEH – Time Event handling	SHE - State Event Handling	DAE - DAE Solver	IR - Index Reduction
MATLAB	no	no	no	(yes)	(yes)	no
Simulink	yes	(yes)	yes	(yes)	(yes)	no
MATLAB / Simulink	yes	yes	yes	yes	(yes)	no
ACSL	yes	yes	yes	yes	yes	no
Dymola	yes	yes	yes	yes	yes	yes

Table 1. Availability of *Classical Features* in selected simulators;
yes / no – available / not available
(yes) / (no) – available, , but difficult to use / yet not available, but foreseen or way around.

In Table 1, the availability of features is indicated by ‘yes’ and ‘no’; a ‘yes’ in parenthesis ‘(yes)’ means, that the feature is complex to use. MS - ‘Model Sorting’, is a standard feature of a simulator – but missing in MATLAB (in principle, MATLAB cannot be called a simulator). On the other hand, MATLAB’s ODE solvers offer limited features for DAEs (systems with mass matrix) and an integration stop on event condition, so that SHE and DAE get a (‘yes’). In Simulink, event descriptions are possible by means of triggered subsystems, so that ED gets a ‘(yes)’ because of complexity. A combination of MATLAB and Simulink suggest putting the event description and handling at MATLAB level, so that ED and SHE get both a ‘yes’. DAE solving is based on modified ODE solvers, using the nested approach (see before), so DE gets only a ‘(yes)’ for all MATLAB/Simulink combinations. Time events are not supported in MATLAB, but they are basic feature in Simulink.

ACSL is a classical simulator with sophisticated state event handling, and since version 10 (2001) DAEs can be modelled directly by the residuum construct, and they are solved by the DASSL algorithm (a well-known direct DAE solver, based on the simultaneous approach), or by modified ODE solvers (nested approach) – so ‘yes’ for ED, SHE, and DAE. In case of DAE index $n = 1$, the DASSL algorithm guarantees convergence, in case of higher index integration may fail. ACSL does not perform index reduction (IR ‘no’). ACSL comes with a sophisticated state event handling, so that all kind of events can be modelled and handled in a comfortable manner.

Dymola is a modern simulator, implemented in C, and based on physical modelling. Model description may be given by implicit laws, symbolic manipulations extract a proper ODE or DAE state space system, with index reduction for high index DAE systems – all classical features are available. Dymola started a new area in modelling and simulation of continuous and hybrid systems (see Section 4).

4 Physical modelling and state chart modelling

There are three basic developments to extend the structure of simulators. First, the extension from ODEs to DAE stimulated the evolvement of *Physical Modelling* – modelling based on laws and physical ‘modules’, textually und graphically – Dymola started the development. Second, influences from computer engineering suggest use of UML – *Unified Modelling Language*, especially UML the use of UML state charts for discrete events. And third, as consequence of the hybrid decomposition of models by state charts, and influenced by experiences from co-simulation, handling of Structural Dynamic Systems became important.

4.1 Physical modelling

In the 1990s, many attempts have been made to improve and to extend the CSSL structure, especially for the task of mathematical modelling. The basic problem was the state space description, which limited the construction of modular and flexible modelling libraries. Two developments helped to overcome this problem. On modelling level, the idea of physical modelling gave new input, and on implementation level, the object-oriented view helped to leave the constraints of input/output relations.

In physical modelling, a typical procedure for modelling is to cut a system into subsystems and to account for the behaviour at the interfaces. Balances of mass, energy and momentum and material equations model each subsystem. The complete model is obtained by combining the descriptions of the subsystems and the interfaces. This approach requires a modelling paradigm different to classical input/output modelling. A model is considered as a constraint between system variables, which leads naturally to DAE descriptions. The approach is very convenient for building reusable model libraries.

In 1996, the situation was thus similar to the mid 1960s when CSSL was defined as a unification of the techniques and ideas of many different simulation programs. An international effort was initiated in September 1996 for bringing together expertise in object-oriented physical modelling (port based modelling) and defining a modern uniform modelling language – mainly driven by the developers of Dymola. The new modelling language is called *Modelica*. Modelica is intended for modelling within many application domains such as electrical circuits, multibody systems, drive trains, hydraulics, thermo-dynamical systems, and chemical processes etc. It supports several modelling formalisms: ordinary differential equations, differential-algebraic equations, bond graphs, finite state automata, and Petri nets etc.

Modelica is intended to serve as a standard format so that models arising in different domains can be exchanged between tools and users. Modelica is a not a simulator, Modelica is a modelling language, supporting and generating mathematical models in physical domains. When the development of Modelica started, also a competitive development, the extension of VHDL towards VHDL-AMS was initiated. Both modelling languages aimed for general-purpose use, but VHDL-AMS mainly addresses circuit design, and Modelica covers the broader area of physical modelling; modelling constructs such as Petri nets and finite automata could broaden the application area, as soon as suitable simulators can read the model definitions.

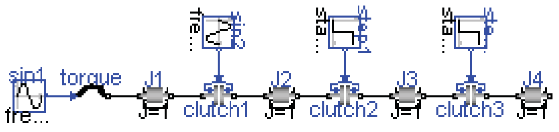


Figure 4. Graphical Modelica model for coupled clutches.

Modelica offers a textual and graphical modelling concept, where the connections of physical blocks are bidirectional physical couplings, and not directed flow. An example demonstrates how drive trains are modelled. The drive train consists of four inertias and three clutches, where the clutches are controlled by input signals (Figure 4). The graphical model layout corresponds with a textual model representation, shown in Listing 3 (abbreviated, simplified).

```
encapsulated model CoupledClutches; "Drive train"
  parameter SI.Frequency freqHz=0.2; ….
  Rotational.Inertia J1(J=1,phi(ic=0),w(ic=10));
  Rotational.Torque torque;
  Rotational.Clutch clutch1(peak=1.1, fn_max=20);
  Rotational.Inertia J3(J=1); ...........................................
equation
  connect(sin1.outPort, torque.inPort);
  connect(torque.flange_b, J1.flange_a);
  connect(J1.flange_b, clutch1.flange_a);
      ............................................. . .
  connect(step2.outPort, clutch3.inPort);
end CoupledClutches;
```

Listing 3. Dymola model description for coupled clutches, definition and connection of mechanical elements.

The translator from Modelica into the target simulator must not only be able to sort equations, it must be able to process the implicit equations symbolically and to perform DAE index reduction (or a way around). Up to now – similar to VHDL-AMS – some simulation systems understand Modelica (2008; generic – new simulator with Modelica modelling, extension - Modelica modelling interface for existing simulator):

- Dymola from Dynasim (generic),
- MathModelica from MathCore Engineering (generic)
- SimulationX from ISI (generic/extension)
- Scilab/Scicos (extension)
- MapleSim (extension, announced)
- Open Modelica - since 2004 the University of Lyngby develops an provides an open Modelica simulation environment (generic),
- Mosilab - Fraunhofer Gesellschaft develops a generic Modelica simulator, which supports dynamic variable structures (generic)
- Dymola / Modelica blocks in Simulink
- Simscape – the very new physical modelling language in MATLAB/Simulink is very similar to Modelica

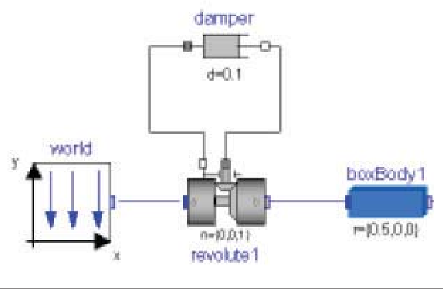
As Modelica also incorporates graphical model elements, the user may choose between textual modelling, graphical modelling, and modelling using elements from an application library. Furthermore, graphical and textual modelling may be mixed in various kinds. The minimal modelling environment is a text editor; a comfortable modelling environment offers a graphical modelling editor.

The *Constrained Pendulum* example can be formulated in Modelica textually as a physical law for angular acceleration. The event with parameter change is put into an `algorithm` section, defining and scheduling the parameter event **SE-P** (Listing 4). As, instead of angular velocity, the tangential velocity is used as state variable, the second state event **SE-S** 'vanishes'.

```
equation /*pendulum*/
  v = length*der(phi);
  vdot = der(v);
  mass*vdot/length + mass*g*sin(phi)
  +damping*v = 0;
algorithm
 if (phi<=phipin) then length:=ls; end if;
 if (phi>phipin) then length:=l1; end if;
```

Listing 4. Dymola model description for *Constrained Pendulum*, dynamics and state events.

Modelica allows combining textual and graphical modelling. For the *Constrained Pendulum* example, the basic physical dynamics could be modelled graphically with joint and mass elements, and the event of length change is described in an `algorithm` section, with variables interfacing to the predefined variables in the graphical model part (Figure 5).



```
algorithm
if (revolute1.phi
     <= phipin then
    revolute1.length:=ls;
end if;
if (revolute1.phi
     < phipin then
    revolute1.length:=l1;
end if;
```

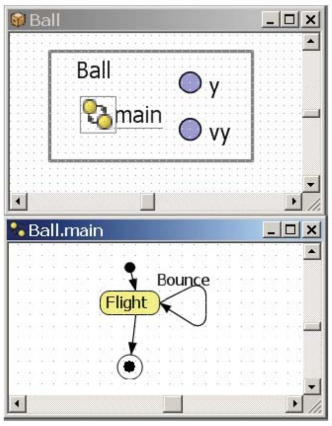
Figure 5. Mixed graphical / textual Dymola model for *Constrained Pendulum*

4.2 UML State Chart Modelling

In the end of the 1990s, computer science initiated a new development for modelling discontinuous changes. The *Unified Modelling Language* (UML) is one of the most important standards for specification and design of object oriented systems. This standard was tuned for real time applications in the form of a new proposal, *UML Real-Time* (UML-RT). By means of UML-RT, objects can hold the dynamic behaviour of an ODE.

In 1999, a simulation research group at the Technical University of St. Petersburg used this approach in combination with a hybrid state machine for the development of a hybrid simulator (*MVS*), from 2000 on available commercially as simulator *AnyLogic*. The modelling language of AnyLogic is an extension of UML-RT; the main building block is the *Active Object*. Active objects have internal structure and behaviour, and allow encapsulating of other objects to any desired depth. Active objects interact with their surroundings through boundary objects: ports for discrete communication, and variables for continuous communication. The activities within an object are usually defined by state charts (extended state machine). While discrete model parts are described state charts, events, timers and messages, the continuous models are described by ODEs and DAEs in CSSL-type notation and with state charts within an object.

An AnyLogic implementation of the well-known *Bouncing Ball* example (see *ARGESIM Benchmark C20*) shows a simple use of state chart modelling (Figure 6). The model equations are defined in the active object ball, together with the state chart ball.main. This state chart describes the interruption of the state flight (without any equations) by the event bounce (**SE-P** and **SE-S** event) defined by condition and action.



```
Equations
d(y)/dt=vy
d(vy)/dt=-g
```

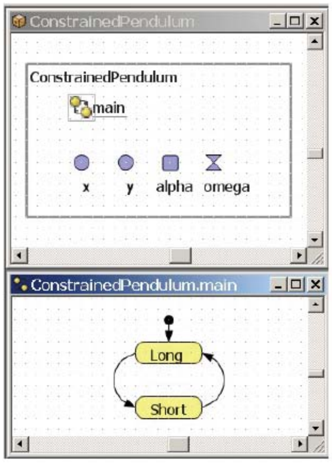
```
Change event
y<=0 && vy < 0

Action
vy = - k * vy;
```

Figure 6. AnyLogic mixed graphical / textual model for the *Bouncing Ball*.

AnyLogic influenced further developments for hybrid and structural dynamic systems, and led to a discussion in the Modelica community with respect to a proper implementation of state charts in Modelica. State charts are to be seen as comfortable way to describe complex WHEN – and IF – constructs, being part of the model, but state charts may also control different models from a higher level. A minor problem is the fact, that the state chart notation is not really standardised; AnyLogic makes use of the Harel state chart type.

An AnyLogic implementation for the *Constrained Pendulum* may follow the implementation for the bouncing ball (Figure 6). An primary active object (Constrained Pendulum)'holds' the equations for the pendulum, together with a state chart (main) switching between short and long pendulum. The state chart nodes are empty; the arcs define the events (Figure 7). Internally, AnyLogic restarts at each hit the same pendulum model (trivial hybrid decomposition).



```
Equations
d(alpha)/dt = omega
d(omega)/dt =
    (-g*sin(alpha)-
      mu*omega)/l
x = l*sin(alpha)
y = l*cos(alpha)
```

```
Change eventLong
(alpha>=alphaN)||
(alpha<=alphaN)

Action
l=ll
omega=omega*ls/ll
```

Figure 7. AnyLogic mixed graphical / textual model for *Constrained Pendulum*, simple implementation.

5 Hybrid modelling and structural-dynamic modelling

Continuous simulation and discrete simulation have different roots, but they are using the same method, the analysis in the time domain. During the last decades a broad variety of model frames (model descriptions) has been developed. In continuous and hybrid simulation, the explicit or implicit state space description is used as common denominator. This state space may be described textually, or by signal-oriented graphic blocks (e.g. SIMULINK), or by physically based block descriptions (Modelica, VHDL-AMS).

Hybrid systems – systems with state events of essential types, often come together with a change of the dimension of the state space, then called *Structural-dynamic Systems*. The dynamic change of the state space is caused by a state event of type **SE-D**. In contrary to state events **SE-P** and **SE-S**, states and derivatives may change continuously and differentiable in case of structural change. In principle, structural-dynamic systems can be seen from two extreme viewpoints. The one says, in a maximal state space, state events switch on and off algebraic conditions, which freeze certain states for certain periods. The other one says that a global discrete state space controls local models with fixed state spaces, whereby the local models may be also discrete or static.

These viewpoints derive two different approaches for structural dynamic systems modelling, the

- maximal state space, and the
- *hybrid* decomposition.

5.1 Maximal state space for structural-dynamic systems – internal events

Most implementations of physically based model descriptions support a big monolithic model description, derived from laws, ODEs, DAEs, state event functions and internal events. The state space is maximal and static, index reduction in combination with constraints keep a consistent state space. For instance, Dymola, OpenModelica, and VHDL-AMS follow this approach. This approach can be classified with respect to event implementation. The approach handles all events of any kind (SE-P, SE-S, and SE-D) within the ODE solver frame, also events which change the state space dimension (change of degree of freedoms) – consequently called *internal events*.

Using the classical state chart notation, *internal state events I-SE* caused by the model schedule the model itself, with usually different re-initialisations (depending on the event type I-SE-P, I-SES, I-SE-D; Figure 8). For instance, VHDL-AMS and Dymola follow this approach, handling also DAE models with index higher than 1; discrete model parts are only supported at event level. ACSL and MATLAB / Simulink generate also a maximal state space.

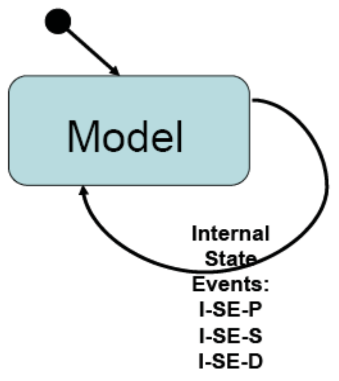


Figure 8. State chart control of *Internal Events* for switching in one singular model (one model instance).

5.2 Hybrid decomposition for structural-dynamic systems – external events

The hybrid decomposition approach makes use of *external events* (E-SE), which control the sequence and the serial coupling of one model or of more models. A convenient tool for switching between models is a state chart, driven by the *external events* – which itself are generated by the models. Following e.g. the UML-RT notation, control for continuous models and for discrete actions can by modelled by state charts. Figure 9 shows the hybrid coupling of two models, which may be extended to an arbitrary number of models, with possible events E-SE-P, E-SE-S, and ESE-D. As special case, this technique may be also used for serial conditional 'execution' of one model – Figure 10 (only for SE-P and SE-S).

This approach additionally allows not only dynamically changing state spaces, but also different model types, like ODEs, linear ODEs (to be analysed by linear theory), PDEs, co-simulation, etc. to be processed in serial or also in parallel, so that also co-simulation can be formulated based on external events. The approach allows handling all events also outside the ODE solver frame. After an event, a very new model can be started. This procedure may make sense especially in case of events of type SE-D and SE-S. As consequence, consecutive models of different state spaces may be used.

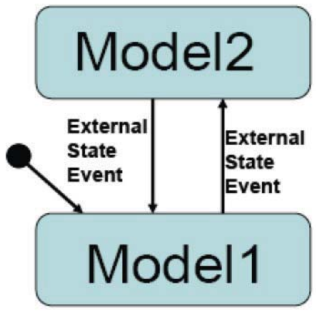


Figure 9. State chart control of *External Events* for switching between two different models.

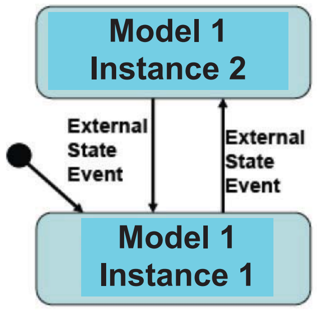


Figure 10. State chart control of *External Events* for switching between two different instances of one model.

Figure 11 shows a structure for a simulator supporting structural dynamic modelling and simulation. The figure summarises the outlined ideas by extending the CSSL structure by control model, external events and multiple models. The main extension is that the translator generates not only one DAE model; he generates several DAE models from the (sub)model descriptions, and external events from the connection model, controlling the model execution sequence in the highest level of the dynamic event list.

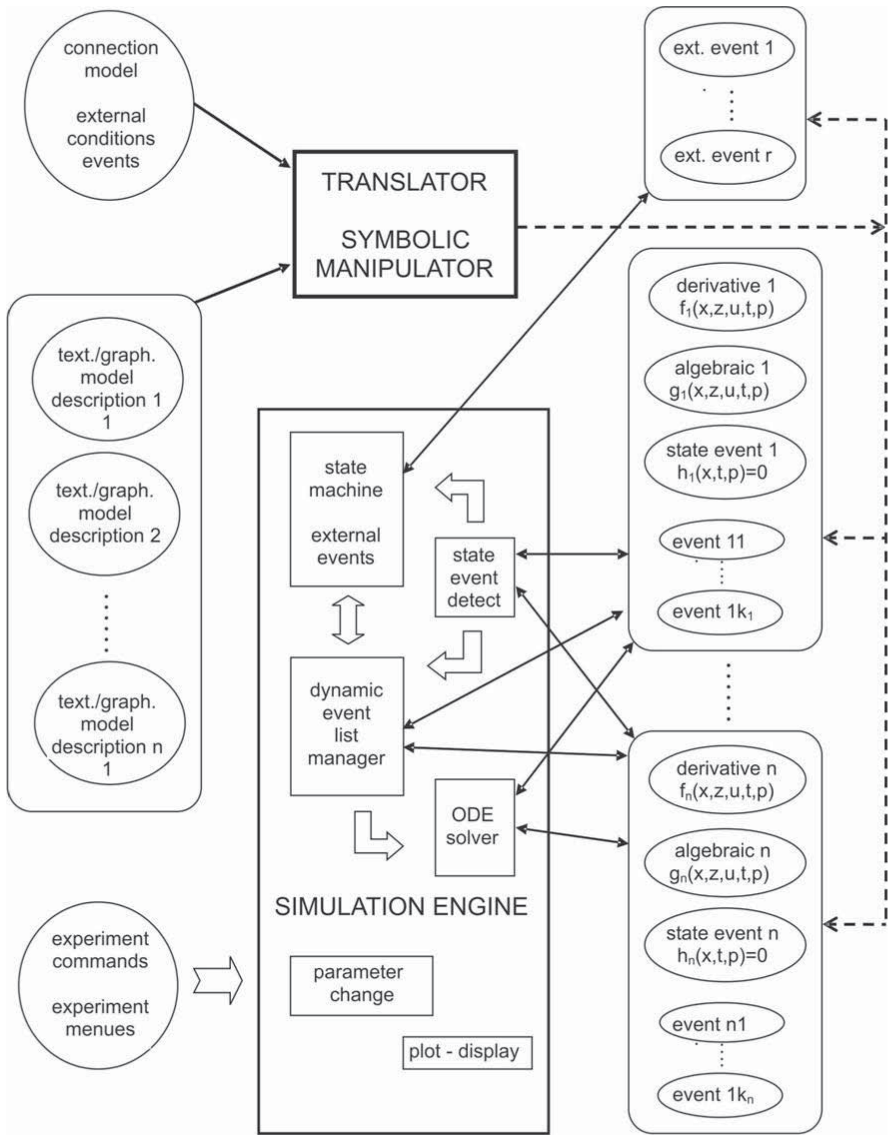

connection model
external conditions events
TRANSLATOR
SYMBOLIC MANIPULATOR
text./graph. model description 1 1
text./graph. model description 2
text./graph. model description n 1
state machine
external events
state event detect
dynamic event list manager
ODE solver
SIMULATION ENGINE
parameter change
plot - display
experiment commands
experiment menues
ext. event 1
ext. event r
derivative 1 $f_1(x,z,u,t,p)$
algebraic 1 $g_1(x,z,u,t,p)$
state event 1 $h_1(x,t,p)=0$
event 11
event $1k_1$
derivative n $f_n(x,z,u,t,p)$
algebraic n $g_n(x,z,u,t,p)$
state event n $h_n(x,t,p)=0$
event n1
event $1k_n$

Figure 11. Extended CSSL-type simulator structure for structural-dynamic systems and for independent consecutive models.

There, all (sub)models may be precompiled, or the new recent state space may be determined and translated to a DAE system in case of the external event (interpretative technique). Clearly, not only ODE solver can make use of the model descriptions (derivatives), but also eigenvalue analysis and steady state calculation may be used and other analysis algorithms. Furthermore, complex experiments can be controlled by external events scheduling the same model in a loop. A simulator structure as proposed in Figure 7 is a very general one, because it allows as well external as ell as internal events, so that hybrid coupling with variable state models of any kind is possible.

5.3 Mixed hybrid approach with external and internal event

A simulator structure as proposed in Figure 11 is a very general one, because it allows as well external as ell as internal events, so that hybrid coupling with variable state models of any kind with internal and external events is possible (Figure 12).

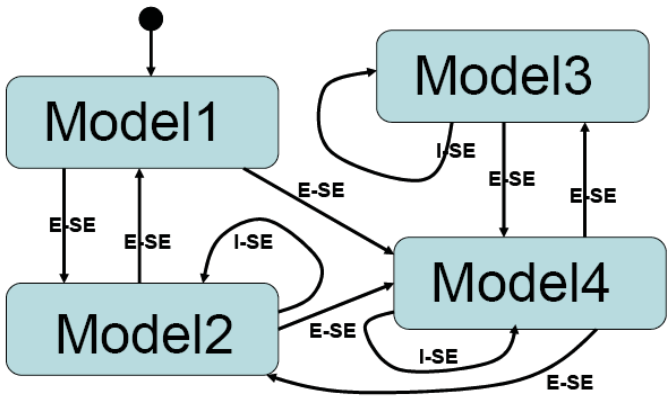


Figure 12. State chart control for different models with *Internal* and *External Events*.

Both approaches have advantages and disadvantages. The classical Dymola approach generates a fast simulation, because of the monolithic program. However, the state space is static. Furthermore, Modelica centres on physical modelling. A hybrid approach handles separate model parts and must control the external events. Consequently, two levels of programs have to be generated: dynamic models, and a control program – today's implementations are interpretative and not compiling, so that simulation times increase - but the overall state space is indeed dynamic.

A challenge for the future lies in the combination of both approaches. The main ideas are:

- Moderate hybrid decomposition
- External and internal events
- Efficient implementation of models and control

For instance, for parameter state events (SE-P) an implementation with an internal event may be sufficient (I-SE-P), for an event of SE-S type implementation with an external event may be advantageous because of easier state re-initialisation (E-SE-S), and for a structural model change (SE-D) an implementation with an external event may be preferred (E-SE-D), because of much easier handling of the dynamic state change – and less necessity for index reduction. An efficient control of the sequence of models can be made by state charts, but also by a well-defined definitions and distinction of IF - and WHEN - constructs, like discussed in extensions of Scilab/Scicos for Modelica models.

6 Structural features in simulators

While the *classical features* discussed before address the CSSL-standard, *structural features* characterise features for physical modelling and for structural dynamic systems. This section investigates the availability or of structural features in some simulators, and summarises the results in Table 3. The features may be classified as follows:

- Support of a-causal physical modelling (sometimes called port-based modelling) at textual (PM-T) or graphical level (PM-G),
- Modelica standard (MOD) for a-causal physical modelling ,
- Decomposition of structural dynamic systems with dynamic features (SD) – features for external events, and
- Support of state chart modelling or a of a similar construct, by means of textual (SC-T) or graphical (SC-G) constructs.

In principle, each combination of the above features is possible. By means of the maximal state space approach, each classic simulator can handle structural dynamic systems, but a-causal modelling may be supported or not, and state chart modelling may be available or not. Simulators with a-causal modelling may support hybrid decomposition or not, and state chart modelling may be available or not. Simulators with features for state chart modelling may support hybrid decomposition or not, and a-causal modelling may be offered or not. In general, interpreter-oriented simulators offer more structural flexibility, but modern software structures would allow also flexibility with precompiled models or with models compiled 'on the fly'.

In addition, of interest are also structural features as

- simulation-driven visualisation (with visualisation objects defined with the model objects; VIS),
- frequency domain analysis and linearization for steady state analysis (FA), and
- extended environment for complex experiments and data pre- and postprocessing (ENV).

6.1 MATLAB / SimuLink / Stateflow / SimScape

The mainly interpretative systems MATLAB / Simulink offer different approaches. First, MATLAB itself allows any kind of static and dynamic decomposition (SD 'yes'), but MATLAB is not a simulator, because the model equations have to be provided in a sorted manner, to be called from an ODE solver (MS 'no'). Second, MATLAB allows hybrid decomposition at MATLAB level with Simulink models. There, from MATLAB different Simulink models are called conditionally, and in Simulink, a state event is determined by the hit-crossing block (terminating the simulation). Simulink is MATLAB's simulation module for block-oriented dynamic models (directed signal graphs), which can be combined with Stateflow, MATLAB's module for event-driven state changes described by state charts (SC-T and SC-G 'yes').

At Simulink level, Stateflow, Simulink's state chart modelling tool, may control different submodels. But Simulink can only work with a maximal state space and does not allow hybrid decomposition (SD 'no). Neither basic MATLAB nor basic Simulink support a-causal modelling. First MATLAB/Simulink modules for physical modelling (e.g. *Hydraulic Blockset* and others, 2004 - 2008) were precompiled to a classical state space (PM-T and PM-G 'no'), Modelica modelling is not supported (MOD 'no').

For DAEs, MATLAB and Simulink offer modified LSODE solvers (implicit solvers) for the nested DAE solving approach. In MATLAB any kind of simulation – driven visualisation can be programmed and used in MATLAB or Simulink or in both, but not based on the model definition blocks (VIS '(yes)'). From the beginning on, MATLAB and Simulink offered frequency analysis (FA 'yes'), and clearly, MATLAB is a very powerful environment for Simulink, Stateflow, for all other Toolboxes, and for MATLAB itself (ENV 'yes').

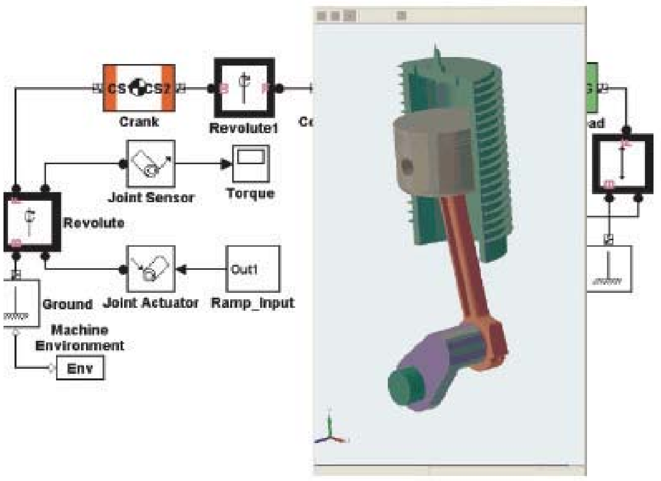
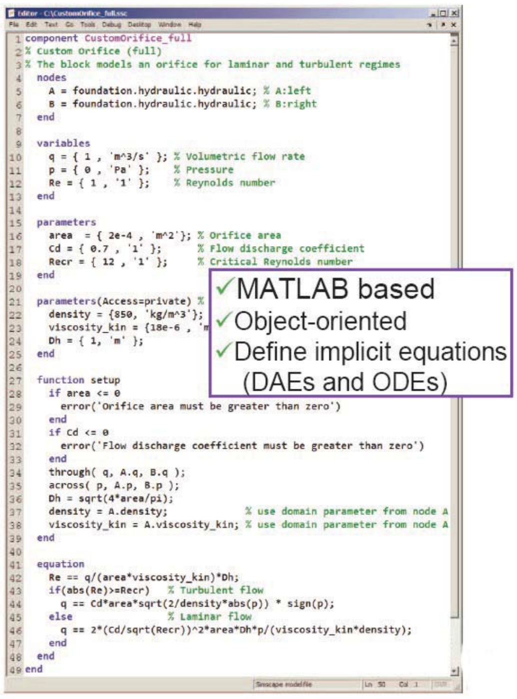


Figure 13. Physical modelling basic libraries in Simulink – physical components modelled in Simscape language.



Editor - C:\CustomOrifice_full.ssc
File Edit Text Go Tools Debug Desktop Window Help
Simscape model file
Ln 50 Col 1
✓MATLAB based
✓Object-oriented
✓Define implicit equations
(DAEs and ODEs)

```
component CustomOrifice_full
% Custom Orifice (full)
% The block models an orifice for laminar and turbulent regimes
  nodes
    A = foundation.hydraulic.hydraulic; % A:left
    B = foundation.hydraulic.hydraulic; % B:right
  end

  variables
    q = { 1 , 'm^3/s' }; % Volumetric flow rate
    p = { 0 , 'Pa' };    % Pressure
    Re = { 1 , '1' };    % Reynolds number
  end

  parameters
    area  = { 2e-4 , 'm^2'}; % Orifice area
    Cd = { 0.7 , '1' };      % Flow discharge coefficient
    Recr = { 12 , '1' };     % Critical Reynolds number
  end

  parameters(Access=private) %
    density = {850, 'kg/m^3'};
    viscosity_kin = {18e-6 , 'm
    Dh = { 1, 'm' };
  end

  function setup
    if area <= 0
      error('Orifice area must be greater than zero')
    end
    if Cd <= 0
      error('Flow discharge coefficient must be greater than zero')
    end
    through( q, A.q, B.q );
    across( p, A.p, B.p );
    Dh = sqrt(4*area/pi);
    density = A.density;             % use domain parameter from node A
    viscosity_kin = A.viscosity_kin; % use domain parameter from node A
  end

  equation
    Re == q/(area*viscosity_kin)*Dh;
    if(abs(Re)>=Recr)   % Turbulent flow
      q == Cd*area*sqrt(2/density*abs(p)) * sign(p);
    else                % Laminar flow
      q == 2*(Cd/sqrt(Recr))^2*area*Dh*p/(viscosity_kin*density);
    end
  end
end
```

Figure 14. Acausal physical modelling in Simscape language – basis for graphic physical components.

In 2008, Mathworks introduced a basic physical modelling language, Simscape, with a basic graphic component library (Figure 13) – based on Simscape language (Figure 14). This modelling language is similar to Modelica, and it is not clear, whether the developers at Mathworks will keep Simscape compatible to Modelica, or whether they will set a competing standard. Simscape is the textual basis for large physical modelling libraries. The Simscape translator is also capable of index reduction (IR 'yes'). So combinations of MATLAB / Simulink / Stateflow with Simscape would offer any modelling possibilities (PM-T and PM-G 'yes', SD 'yes', etc.).

6.2 ACSL

ACSL – Advanced Continuous Simulation Language – has been developed since more than 25 years. ACSL' software structure is a direct mapping of the CSSL structure in Figure 1. The new developers (Aegis Technologies) concentrate on application-oriented simulation solutions, with models are tailor-made for the specific application. Last extensions were a change to C as basic language (instead of FORTRAN), and DAE features using the nested approach with classical solvers, or direct implicit DAE solving with DASSL Code (DAE 'yes', IR 'no'). From the beginning on, steady state calculation, linearization and frequency analysis was a standard feature of ACSL's simulator kernel (FA 'yes'). Since 2000, the environment has been enriched by modules for modelling and environment modules. The first module was a graphical modeller, which seems to make use of physical modelling, but in behind a classical state space is used – PM-T and PM-G 'no'. Furthermore, a simulation-driven visualisation system (third party) is offered (but hard to use) – VIS '(yes)'.

ACSLMath was intended to have same features as MATLAB; available is only a subset, but powerful enough for an extended environment (ENV 'yes'), which can be used for hybrid decomposition of a structural dynamic model in almost the same way than MATLAB does (SD 'yes'). Unfortunately the development of ACSLMath has been stopped. In general, there is no intention to make a-causal physical modelling available, also Modelica is not found in the developers' plans (PM-T, PM-G, and MOD 'no').

6.3 Dymola

Dymola, introduced by F. E. Cellier as a-causal modelling language, and developed to a simulator by H. Elmquist, can be called the mother of Modelica. Dymola is based on a-causal physical modelling and initiated Modelica; consequently, it fully supports Modelica these structural features (PM-T, PM-G, and MOD 'yes'). Together with the model objects, also graphical objects may be defined, so that simulation based pseudo-3D visualisation is available (VIS 'yes'). A key feature of Dymola is the very sophisticated index reduction by the modified Pantelides algorithm, so Dymola handles any DAE system, also with higher index, with bravura (DAE and IR 'yes'). For DAE solving, modified DASSL algorithms are used. In software structure, Dymola is similar to ACSL, using an extended CSSL structure as given in Figure 1 – with the modification that all discrete actions are put into one event module, where CASE - constructs distinguish between the different events (this structure is based on the first simulator engine Dymola used, the DS-Block System of DLR Oberpfaffenhofen).

Dymola comes with a graphical modelling and basic simulation environment, and provides a simple script language as extended environment; new releases offer also optimisation, as built-in function of the simulator. Furthermore, based on Modelica's matrix functions some task of an environment can be performed – so ENV ('yes') – available, but complex/uncomfortable. Dymola offers also a Modelica – compatible state chart library, which allows to model complex conditions (internally translated into `IF-THEN-ELSE` or `WHEN` constructs - SC-T and SC-G '(yes)').

Up to now (2008) the Modelica definition says nothing about structural dynamic systems, and Dymola builds up a maximal state space. And up to now, Modelica does not directly define state charts, and in Dymola a state chart library in basic Modelica notation is available, but working only with internal events within the maximal state space (SD 'no'). For Modelica extension, a working group on hybrid systems has been implemented, in order to discuss and standardise hybrid constructs like state charts, and hybrid decompositions (independent submodels).

6.4 MathModelica

MathModelica, developed by MathCoreAB, was the second simulation system, which understood Modelica modelling. MathModelica is an integrated interactive development, from modelling via simulation to analysis and code integration. As the MathModelica translator is very similar to Dymola's model translator, clearly all related features are available, including index reduction and use of implicit solvers like DASSL (all DAE, IR, PM-T, PM-G and MOD 'yes').

MathModelica follows a software model different to CSSL standard. The user interface consists of a graphical model editor and notebooks. There, a *simulation center* controls and documents experiments in the time domain. Documentation, mathematical type setting, and symbolic formula manipulation are provided via Mathematica, as well as Mathematica acts as extended environment for MathModelica (ENV 'yes') – performing any kind of analysis and visualisation (FA and VIS 'yes'). By means of the Mathematica environment, also a hybrid decomposition of structural dynamic systems is possible, with the same technique like in MATLAB (SD – 'yes').

6.5 Mosilab

Since 2004, Fraunhofer Gesellschaft Dresden develops a generic simulator *Mosilab*, which also initiates an extension to Modelica: multiple models controlled by state automata, coupled in serial and in parallel. Furthermore, Mosilab puts emphasis on co-simulation and simulator coupling, whereby for interfacing the same constructs are used than for hybrid decomposition. Mosilab is a generic Modelica simulator, so all basic features are met (ED, SEH, DAE, PM-T, and PM-G 'yes', and MOD '(yes)' – because of subset implementation at present, 2008). For DAE solving, variants of IDA-DASSL solver are used.

Mosilab implements extended state chart modelling, which may be translated directly due to Modelica standard into equivalent IF – THEN constructs, or which can control different models and model executions (SC-T, SC-G, and SD 'yes'). At state chart level, state events of type SE-D control the switching between different models and service the events (E-SE-D). State events affecting a state variable (SE-S type) can be modelled at this external level (E-SE-S type), or also as classic internal event (I-SE-S). Mosilab translates each model separately, and generates a main simulation program out of state charts, controlling the call of the precompiled models and passing data between the models.

Mosilab is in developing, so it supports only a subset of Modelica, and index reduction has not been implemented yet, so that MOD gets a '(yes)' in parenthesis, and IR gets a '(no)'. Index reduction at present not available in Mosilab, but planned (IR '(no)') – has become topic of discussion: case studies show, that hybrid decomposition of structural dynamic systems results mainly in DAE systems of index $n = 1$, so that index reduction may be bypassed..

Mosilab allows very different approaches for modelling and simulation tasks. In a standard Modelica approach, the *Constrained Pendulum* is defined in the MOSILAB equation layer as implicit law; the state event, which appears every time when the rope of the pendulum hits or 'leaves' the pin, is modelled in an `algorithm section` with if (or when) – conditions. A state chart approach replaces the if- conditions in the `algorithm section` with (textual) state chart construct – with same results, but different implementation of the switching algorithm for the state event (Listing 5).

```
event Boolean lengthen(start = false),
              shorten(start = false);
 end
equation
   lengthen=(phi>phipin); shorten=(phi<=phipin);
 equation /* pendulum*/
   v = l1*der(phi); vdot = der(v);
   mass*vdot/l1 + mass*g*sin(phi)+damping*v= 0;
 end / equation
statechart
state LengthSwitch extends State;
   State Short,Long,Initial(isInitial = true);
   transition Long -> Short event shorten
      action length := ls;
   end transition;
   transition Short -> Long event lengthen
      action length := l1;
   end transition;
 end LengthSwitch;
```

Listing 4. Mosilab model for *Constrained Pendulum* – state chart model with *Internal Events* (I-SE-P).

But Mosilab's state chart construct allows also any kind of hybrid composition of models with different state spaces and of different type (from ODEs to PDEs, etc.). Listing 6 shows a Mosilab implementation of the *Constrained Pendulum* making use of two different pendulum models, controlled externally by a state chart. The `transitions` organise the switching between the pendulums (`remove`, `add`).

Mosilab offers also strong support for simulator coupling (e.g. MATLAB) and time-synchronised coupling of external programs. This feature may be used for any kind of visualisation not based the model definition (VIS '(yes)'). For complex experiments, Mosilab allows to mix model frame and experimental frame and sets up a common extended environment (ENV 'yes'), where also frequency analysis can be implemented (FA '(no)').

```
model Long
  equation
     mass*vdot/l1 + mass*g*sin(phi)+damping*v = 0;
end Long;
model Short
  equation
     mass*vdot/ls + mass*g*sin(phi)+damping*v = 0;
end Short;
event discrete Boolean lengthen(start=true),
                       shorten(start=false);
equation
   lengthen = (phi>phipin);
   shorten  = (phi<=phipin);
statechart
 state ChangePendulum extends State;
   State Short,Long,startState(isInitial=true);
   transition Long -> Short event shorten action
      disconnect ….; remove(L); add(K); connect …
   end transition;
   transition Short -> Long event lengthen action
      disconnect …; remove(K); add(L); connect …
   end transition;
  end / equation
end ChangePendulum;
```

Listing 5. Mosilab model for *Constrained Pendulum* – state chart switching between different pendulums models by *External Events* (E-SE-P).

6.6 Open Modelica

The goal of the *Open Modelica* project is to create a complete Modelica modelling, compilation and simulation environment based on free software distributed in binary and source code form.

The whole OpenModelica environment consists of open software (Figure 15): *OMC* – the Open Modelica Compiler translates Modelica models (with index reduction); *OMShell* as interactive session handler is a minimal experiment frame; Modelica models may be set up by a simple text editor or by a graphical model editor (here, for teaching purposes the model editor of MathModelica is allowed to be used!); the purpose of *OMNotebook* is to provide an advanced Modelica environment and teaching tool; the *DrModelica* notebook provides all the examples from P. Fritzson's book on Modelica; the other modules support environment interfacing and Open Modelica development.

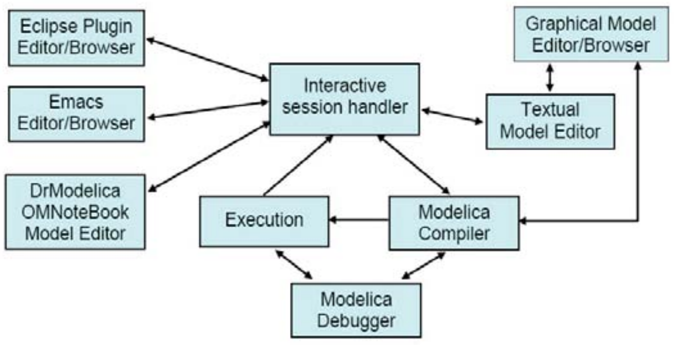

Eclipse Plugin Editor/Browser
Emacs Editor/Browser
DrModelica OMNoteBook Model Editor
Interactive session handler
Graphical Model Editor/Browser
Textual Model Editor
Execution
Modelica Compiler
Modelica Debugger

Figure 15. Software Modules of Open Modelica.

Open Modelica is a generic Modelica simulator, so all basic features are met (ED, SEH, DAE, PM-T, PM-G, IR and MOD 'yes'; for DAE solving, variants of DASSL solver are used). P. Fritzson, the initiator of Open Modelica puts emphasis on discrete events and hybrid modelling, so documentation comes with clear advice for use of IF – and WHEN – clauses in Modelica, and with state chart modules in DrModelica – so SC-T gets 'yes'. Figure 16 and Listing 6 show the equivalence of a state chart and the correct definition as Modelica submodel. For graphical state chart modelling the experimental Modelica state chart library can be used – so SC-G '(yes)'.

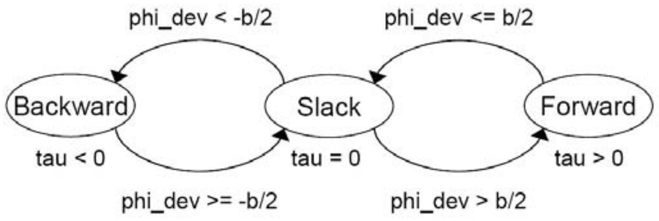


Figure 16. State chart model for backlash function.

```
partial model SimpleBacklash
Boolean backward, slack, for-
ward;
parameter ……
equation
phi_dev = phi_rel - phi_rel0;
 backward = phi_rel < -b/2;
 forward = phi_rel > b/2;
 slack = not (backward or for-
ward);
 tau = if forward then
            c*(phi_dev - b/2)
       else (if backward then
               c*(phi_dev +
b/2)
            else 0);
end SimpleBacklash
```

Listing 6. Modelica implementation for state chart model of backlash function (Figure 16).

The notebook features allow interfaces and extensions of any kind, e.g. for data visualisation and frequency analysis – FA and VIS '(yes)'; they allow also for controlled executive of different models, so that hybrid decomposition of structural dynamic systems is possible – SD '(yes)'.

6.7 SimulationX

SimulationX is a new Modelica simulator developed by ITI simulation, Dresden. This almost generic Modelica simulator is based on ITI's simulation system ITI-SIM, where the generic IT-SIM modelling frame has been replaced by Modelica modelling. From the very beginning on, ITI-SIM concentrated on physical modelling, with a theoretical background from power graphs and bond graphs. Figure 17 shows graphical physical modelling in ITI-SIM – very similar to Modelica graphical modelling. The simulation engine from ITI-SIM drives also SimulationX, using a sophisticated implicit integration scheme, with state event handling.

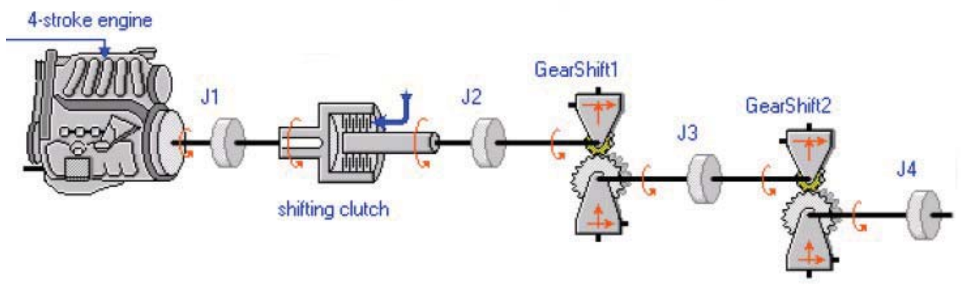


Figure 17. Physical modelling in ITI-SIM / SimulationX.

Consequently, all features related to physical modelling are available: (ED, SEH, DAE, PM-T, PM-G, and MOD 'yes'; index reduction is not really implemented – IR '(no)'. State chart constructs are not directly supported (SC-T 'no'), but due to Modelica compatibility the Modelica state chart library can be used (SC-G – '(yes)'. Frequency analysis is directly supported in the simulation environment (FA – 'yes'), as well as interfaces to other systems (ENV – '(yes)') and pseudo-3D visualisation (VIS – 'yes').

6.8 AnyLogic

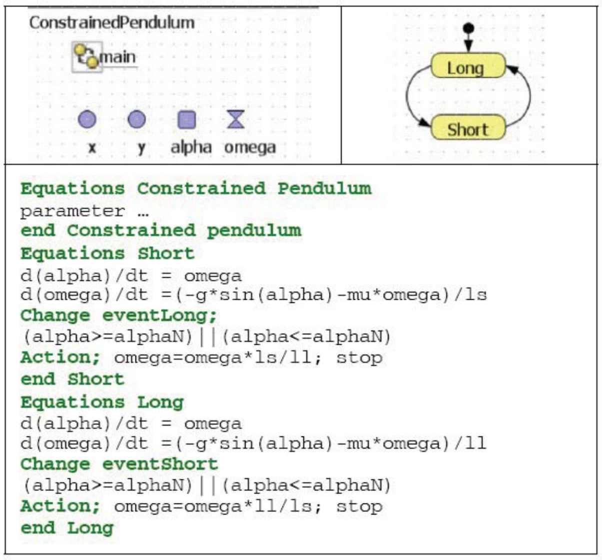
AnyLogic – already discussed in a previous section) is based on hybrid automata (SC-T and SC-G - 'yes'). Consequently, hybrid decomposition and control by external events is possible (ED, SD 'yes'). AnyLogic can deal partly with implicit systems (only nested approach, DAE '(yes)'), but does not support a-causal modelling (PM-T, PM-G - 'no') and does not support Modelica (MOD - 'no'). Furthermore, new versions of AnyLogic concentrate more on discrete modelling and modelling with System Dynamics, but without event detection (SEH '(no)'.

AnyLogic offers many other modelling paradigms, as System Dynamics, Agent-based Simulation, and DEVS. AnyLogic is Java-based and provides simulation-driven visualisation and animation of model objects (VIS 'yes') and can also generate Java web applets. From software engineering view, AnyLogic is a programming environment for Java – so ENV – '(yes)'.

In AnyLogic, various implementations for the *Constrained Pendulum* are possible. A classical implementation is given in Figure 7 following classical textual ODE modelling, whereby a state chart is used for switching (I-SE-P, I-SE-S).

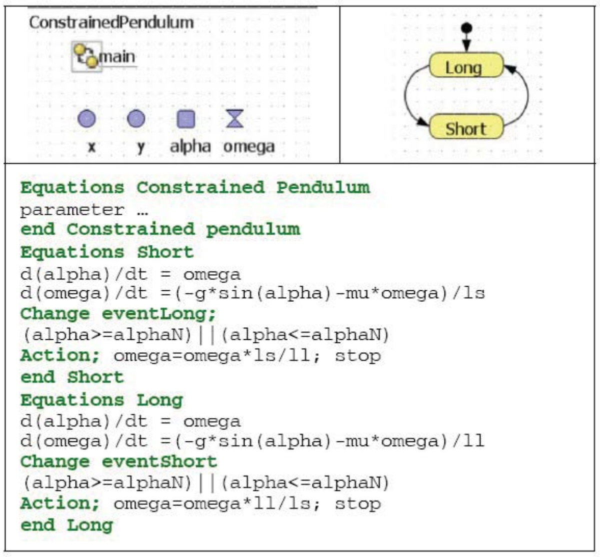
AnyLogic E-SE-P Model with State Charts. A hybrid decomposed model may make use of two different models, each defined in substate / submodel `Short` and `Long`. – both part of a state chart switching between these submodels. The events defined at the arcs stop the actual model, set new initial conditions and start the alternative model (Figure 13).

AnyLogic E-SE-P Model with Parallel Models. AnyLogic works interpretatively, after each external event state equations are tracked and sorted anew for the new state space. This makes it possible, to decompose model not only in serial, but also in parallel. In *Constrained Pendulum* example, the ODE for the angle, which is not effected by the events, may be put in the main model, together with transformation to Cartesian coordinates (Figure 14), which seems to run in parallel with different velocity equations.



```
Equations Constrained Pendulum
parameter …
end Constrained pendulum
Equations Short
d(alpha)/dt = omega
d(omega)/dt =(-g*sin(alpha)-mu*omega)/ls
Change eventLong;
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*ls/ll; stop
end Short
Equations Long
d(alpha)/dt = omega
d(omega)/dt =(-g*sin(alpha)-mu*omega)/ll
Change eventShort
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*ll/ls; stop
end Long
```

Figure 18. Anylogic model for *Constrained Pendulum*, hybrid model decomposition with two Pendulum models and *External Events*



```
Equations Constrained Pendulum
parameter …
end Constrained pendulum
Equations Short
d(alpha)/dt = omega
d(omega)/dt =(-g*sin(alpha)-mu*omega)/ls
Change eventLong;
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*ls/ll; stop
end Short
Equations Long
d(alpha)/dt = omega
d(omega)/dt =(-g*sin(alpha)-mu*omega)/ll
Change eventShort
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*ll/ls; stop
end Long
```

Figure 19. Anylogic model for *Constrained Pendulum*, hybrid model decomposition with two models for angular velocity and overall model for angle

6.9 SCILAB / SCICOS

Scilab is a scientific software package for numerical computations with a powerful open computing environment for engineering and scientific applications. Scilab is open source software. Scilab is since 2003 in the responsibility of the Scilab Consortium. *Scicos* is a graphical dynamical system modeller and simulator toolbox included in Scilab. Scilab / Scicos is an open source alternative to MATLAB / Simulink, developed in France. So it has nearly the same features than MATLAB: no equation sorting– MS – 'no'!; DE, IR, PM-T, PM-G, MOD, SC-T, and SC-G – 'no'; SEH, DAE, and VIS – '(yes)', remarkably – SD, FA and ENV – 'yes'. Similarly, Scicos has classical features ED, SEH, and DAE – 'yes'.

The developers of Scicos started early with a kind of physical modelling (PM-T, PM-G – yes) - Figure 20 shows an electrical modelling palette of Scicos. They are working on extensions in two directions:

- extending the model description by full Modelica models (text/graphic) – so MOD and IR '(yes)'
- refining the `IF-THEN-ELSE` and `WHEN` clause introducing different classes of associated events, resulting 'state chart clauses' - so SC-T – 'yes'

Standalone Scicos has no features for frequency analysis, structural decomposition and extended environment (FA, SD, ENV – 'no'), but limited visualisation (VIS – '(yes)'); Scicos controlled by Scilab has all these features (VIS, FA, SD, ENV – 'yes').

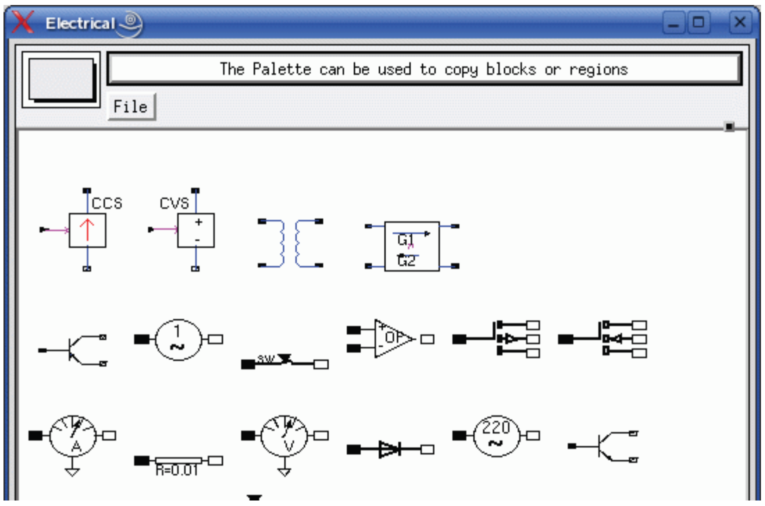
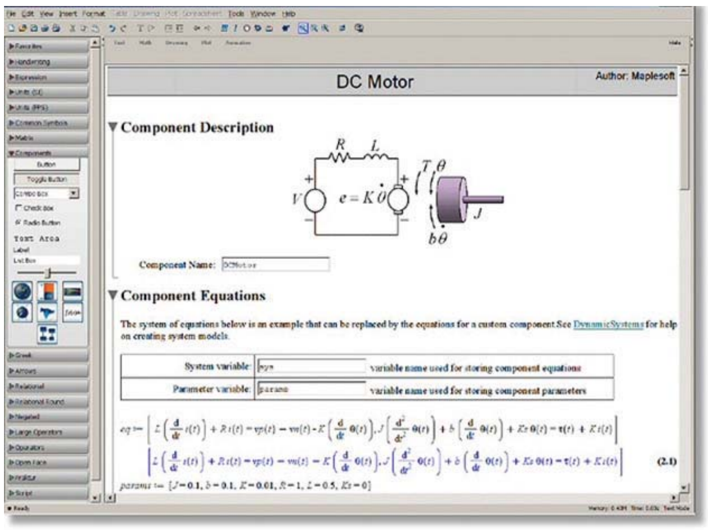


Figure 20. Scicos *Physical Modelling Palette* for electrical applications

6.10 Maple

Maple – developed by Maplesoft, Canada, has developed a toolbox *MapleSim*, which will understand Modelica models (PM-T, PM-G, and MOD – 'yes'). MapleSim comes with a big library of physical components, basically modelled in Maple code (Figure 21). Maple also acts as environment and provides sophisticated DAE solvers, as well as symbolic algorithms for index reduction (DAE, IR, ENV, VIS, and FA – 'yes').



DC Motor

Author: Maplesoft

Component Description

Component Name: DCMotor

Component Equations

The system of equations below is an example that can be replaced by the equations for a custom component. See DynamicSystems for help on creating system models.

System variable:	sys variable name used for storing component equations
Parameter variable:	params variable name used for storing component parameters

$$eq := \left[L\left(\frac{d}{dt} i(t)\right) + R\,i(t) = vp(t) - vn(t) - K\left(\frac{d}{dt}\theta(t)\right), J\left(\frac{d^2}{dt^2}\theta(t)\right) + b\left(\frac{d}{dt}\theta(t)\right) + Ks\,\theta(t) = \tau(t) + K\,i(t) \right]$$

$$\left[L\left(\frac{d}{dt} i(t)\right) + R\,i(t) = vp(t) - vn(t) - K\left(\frac{d}{dt}\theta(t)\right), J\left(\frac{d^2}{dt^2}\theta(t)\right) + b\left(\frac{d}{dt}\theta(t)\right) + Ks\,\theta(t) = \tau(t) + K\,i(t) \right] \quad (2.1)$$

$params := [J = 0.1, b = 0.1, K = 0.01, R = 1, L = 0.5, Ks = 0]$

Figure 21. Physical modelling definition in *MapleSim* library.

In development are constructs for events and event handling (ED–'yes)', SEH–'(no)'); textual state chart modelling has not been discussed yet, graphical state chat notation may com from the experimental Modelica state chart library (SC-T – 'no', SC-G – '(yes)').

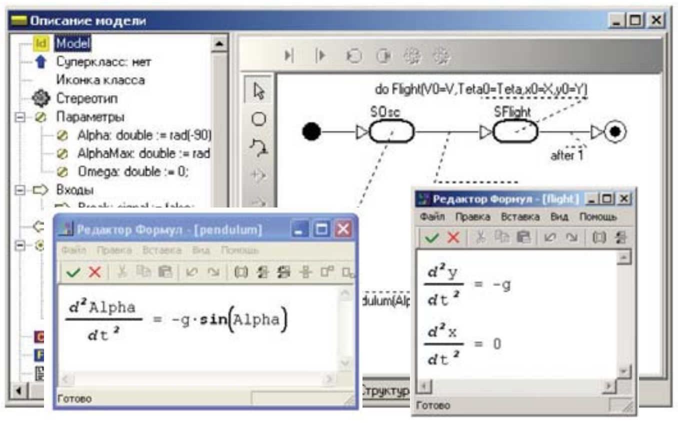
6.11 Model Vision Studium MVS

Model Vision Studium (MVS) – is an integrated graphical environment for modelling and simulation of complex dynamical systems. Development of MVS started in the 1990ies at Technical University of St. Petersburg; for end of 2008, an English version is announced.

Basis of MVS are hybrid state charts (SC-T, SC-G - 'yes'), allowing any parallel, serial, and conditional combination of continuous models, described by DAEs, and controlled and interrupted by state events (ED, SHE - 'yes'). State models itself are objects to be instantiated in various kinds, so that structural dynamic systems of any kind can be modelled (SD - 'yes'). Textual physical and DAE modelling is supported by an editor capable of editing mathematical formula (DAE and PM-T 'yes', PM-G no), but no Modelica compatibility (MOD – 'no').

For MVS, a subset of *UML Real Time* was chosen and extended to state chart activities (Java – based). Other modules (simulation kernel, environment) are linked modules (e.g. C-modules), e.g. Java-base simulation driven visualisation (VIS - 'yes'). In principle, MVS and AnyLogic have been developed in parallel. The continuous elements in AnyLogic have been taken from MVS, because AnyLogic started as pure discrete simulator.

State charts are similar to AnyLogic, consisting of different implicit state space descriptions – and also defining complex experiments (calling different models; ENV – 'yes), but without frequency analysis (FA – 'no'). As example, two states pendulum and flight, and a state chart handling the external event of type E-SE-D (Figure 22) describe a breaking pendulum.



Alpha: double := rad(-90)
AlphaMax: double := rad
Omega: double := 0;

do Flight(V0=V,Teta0=Teta,x0=X,y0=Y)

$$\frac{d^2 Alpha}{dt^2} = -g \cdot \sin(Alpha)$$

$$\frac{d^2 y}{dt^2} = -g$$

$$\frac{d^2 x}{dt^2} = 0$$

Figure 22. MVS model for *Breaking Pendulum* - hybrid model decomposition into pendulum and flight model.

7 Comparison of Structural Features availability in selected simulators

While the *Extended Features* address the CSSL-standard and its extensions, *Structural Features* characterise features for physical modelling and for structural dynamic systems, the main development from the year 2000 on. The *Structural Features* may be classified as follows:

- Support of a-causal physical modelling at textual (PM-T) or graphical level (PM-G),
- Modelica standard (MOD) for physical modelling,
- Decomposition of structural dynamic systems with dynamic features (SD)
- Support of state chart modelling or of a similar construct, by means of textual (SC-T) or graphical (SC-G) constructs.

Simulators with a-causal modelling may support hybrid decomposition or not, and state chart modelling may be available or not. Simulators with features for state chart modelling may support hybrid decomposition or not, and a-causal modelling may be offered or not. In general, interpreter-oriented simulators offer more structural flexibility, but modern software structures would allow also flexibility with precompiled models. In addition, of interest are also structural features as:

- simulation-driven visualisation (visualisation objects defined with model objects; VIS),
- frequency domain analysis and linearization for steady state analysis (FA), and
- extended environment for complex experiments and data processing (ENV).

The evaluation of the feature availability is mainly based on solutions of the ARGESIM Benchmarks. In section 3 these benchmarks have been introduced, citing the 'continuous benchmarks, which gave information for this evaluation. Table 2 indicates, which features (E – classical feature, S – structural feature) occurred in which benchmarks. The table list also features which may be also of interest for further investigations:

- optimisation and identification (OPT/ID)
- VHDL-AMS standard (V-AMS)
- System Dynamics modelling (SYS-D)
- Real-time Simulation (RT)
- Co-Simulation (COS)
- Spatial Dynamics (SPAT)
- Parallel Simulation (PAR)

Feature	Type	Benchmarks
MS	C	C1, C3, C5, C7, C9, C11, C13
ED	C	C3, C5, C7, C9, C11, C12, C13, C18, (C20)
TEH	C	C3, C9, C12, C13, C18
SEH	C	C5, C7, C11, C12, C13, (C20)
DAE	C	C7, C11, C13, (C20)
IR	C	C11, C13, (C20)
PM-T	S	C1, C3, C5, C7, C9, C11, C13, C15, C19, (C20)
PM-G	S	C1, C3, C5, C7, C9, C11, C13, C15, C19, (C20)
VIS	S	C1, C3, C7, C9, C11, C12, C17
MOD	S	C1, C3, C5, C7, C9, C11, C13, C15, (C20)
SC-T	S	C3, C5, C7, C9, C11, C12, C13, (C20)
SC-G	S	C3, C5, C7, C9, C11, C12, C13, (C20)
SD	S	C5, C7, C9, C11, C13, C15, (C20)
FA	S	C1, C3, C11, C13
ENV	S	C1, C3, C5, C7, C9, C11, C12, C13, C15, C17, C18, C19, (C20)
OPT/ID	(S)	C7, C15, C17, C18, (C20)
V-AMS	(S)	C3, C5, C7, C9, C13, (C20)
SYS-D	(C)	C1, C7, C15, C17
RT	(C)	C3, C9, C13, C18
COS	(C)	C9, C11, C18
SPAT	(C)	C17, C19
PAR	(S)	CP-1, C19

Table 2. Register of benchmarks and associated investigated features
– first 15 lines indicate evaluated features,
- last 7 lines indicate features for planned evaluation.

Main results of the availability evaluation for extended features (sometimes also referred as classical feature) are summarised in Table 3. The availability of features - is indicated by ‘yes’ and ‘no’; a ‘yes’ in parenthesis ‘(yes)’ means, that the feature is complex to use, a ‘no’ in parenthesis ‘(no)’ means, that the feature is either foreseen or there is a way around. The evaluation will be refined and completed by further benchmark solutions – mainly *C20 – Hybrid Approaches and Hybrid Modelling* – and by further features as given in Table 2.

8 References

As a really adequate reference list, with details on structures, features, and detailed developments and background would cover again 10 pages, alternatively the list is restricted to only few main sources. For information modelling approaches, it is referred to the journal SNE – Simulation News, where regularly benchmarks, also for Modelica modelling, are published (sne.argesim.org, ww.argesim.org). For simulator information, see webpages of distributors / developers.

[1] F. Breitenecker F., I. Troch. *Simulation Software – Development and Trends*. In *Modelling and Simulation of Dynamic Systems / Control Systems, Robotics, and Automation.* H. Unbehauen, I. Troch, and F. Breitenecker (Eds.). Encyclopedia of Life Support Systems (EOLSS), Oxford ,UK, www.eolss.net, 2004.

[2] Cellier, F.E. (1991). *Continuous System Modeling*. Springer, New York.

[3] Cellier, F.E., and E. Kofman. 2006. *Continuous System Simulation*. Springer, New York.

[4] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica.* Wiley IEEE Press, 2005.

[5] Fritzson, P., F.E.Cellier, C. Nytsch-Geusen, D. Broman, M. Cebulla, Eds. 2007. *EOOLT'2007 - Proc. 1st Intl. Workshop on Equation-based Object-oriented Languages and Tools*. TU Berlin Forschungsberichte, Vol. 2007-11.

[6] C. Nytsch-Geusen P. Schwarz. *MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics*. In *Proc. 4th Intern. Modelica Conference,* G. Schmitz (Ed.), Modelica Association - www.modelica.org, 2005, 527 – 535.

[7] N. Popper, F. Breitenecker. Extended and Structural Features of Simulators – A Comparative Study. SNE 18/3-4, 2008, 27-39.

[8] J. C. Strauss. *The SCi continuous system simulation language (CSSL).* Simulation9, SCS Publ. 1967, 281-303.

	MS - Model Sorting	**ED -Event Description**	**SEH -State Event Handling**	**DAE - DAE Solver**	**IR - Index Reduction**	**PM-T - Physical Modelling -Text**	**PM-G - Physical Modelling -Graphics**	**VIS – 'Onlie' - Visualisation**	**MOD – Modelica Modelling**	**SC-T – State Chart – Modelling - Text**	**SC-G – State Chart Modelling - Graphics**	**SD – Structural Dy-namic Systems**	**FA – Frequency Analysis**	**ENV – Extended En-vironment**
MATLAB	no	no	(yes)	(yes)	no	no	no	(yes)	no	no	no	yes	yes	yes
Simulink	yes	(yes)	(yes)	(yes)	no	no	(no)	(yes)	no	no	no	no	yes	(yes)
MATLAB / Simulink	yes	yes	yes	(yes)	no	no	(no)	(yes)	no	no	no	yes	yes	yes
Simulink / Stateflow	yes	yes	yes	yes	no	no	(no)	(yes)	no	(yes)	yes	no	yes	(yes)
Simulink / Simscape	yes	yes	yes	yes	(yes)	yes	yes	(yes)	(no)	no	no	no	yes	(yes)
Simulink / Simscape/ Stateflow	yes	yes	yes	yes	(yes)	yes	yes	(yes)	(no)	(yes)	yes	no	yes	(yes)
ML/Simulin k Simscape Stateflow	yes	yes	yes	yes	(yes)	yes	yes	(yes)	(no)	(yes)	yes	yes	yes	yes
ACSL	yes	yes	yes	yes	no	no	(no)	(yes)	no	no	no	no	yes	yes
Dymola	yes	yes	yes	yes	yes	yes	yes	yes	yes	(yes)	(yes)	no	(no)	(yes)
Math Modelica	yes	yes	yes	yes	yes	yes	yes	(yes)	yes	(no)	(yes)	no	(no)	(no)
MathMode-lica/ Mathe-matica	yes	yes	yes	yes	yes	yes	yes	yes	yes	(no)	(yes)	yes	yes	yes
Mosilab	yes	yes	yes	yes	(no)	yes	yes	(no)	(yes)	yes	yes	yes	no	(yes)
Open Modelica	yes	yes	yes	yes	yes	yes	(no)	(no)	yes	(no)	(yes)	no	no	no
SimulationX	yes	yes	yes	yes	yes	yes	yes	yes	yes	(no)	(yes)	no	yes	(yes)
AnyLogic	yes	yes	(yes)	(yes)	no	no	no	yes	no	yes	yes	yes	no	no
Model Vision	yes	yes	yes	yes	yes	yes	no	yes	no	yes	yes	yes	yes	no
Scilab	no	no	(yes)	(yes)	no	no	no	(yes)	no	no	no	yes	yes	yes
Scicos	yes	(yes)	yes	yes	(yes)	yes	yes	(yes)	(yes)	yes	(yes)	no	no	no
Scilab / Scicos	yes	yes	yes	yes	(yes)	yes	yes	(yes)	(yes)	yes	(yes)	yes	yes	yes
MapleSim	yes	(yes)	(yes)	yes	yes	yes	yes	yes	yes	no	no	(yes)	(yes)	yes

Table3. Availability of *Classical and Structural Features* in selected simulators;
yes / no – available / not available
(yes) / (no) – available, but difficult to use / yet not available, but foreseen or way around.