STRUCTURAL FEATURES IN SIMULATION SYSTEMS – EVOLUTION AND COMPARISON

Felix Breitenecker [(a)], Nikolas Popper[(b)], Günther Zauner[(a,b)]

[(a)] Vienna Univ. of Technology, Austria
[(b)] "die Drahtwarenhandlung" Simulation Services, Vienna, Austria

[(a)] Felix.Breitenecker@tuwien.ac.at, [(b)] Niki.Popper@drahtwarenhandlung.at

ABSTRACT
Object-oriented approaches, DAE modelling, variable structure modelling, Modelica notation and other developments have pushed the development of simulation languages essentially. A key point in these developments are hybrid structures and their related topics as events, event handling, state charts, structural changes, etc. This contribution compares features for modelling and simulation of extended hybrid structures in grown and new simulation systems on basis of classical features – model sorting, event description, event handling, DAE solver, index reduction, and on basis of structural features – physical modelling, Modelica modelling, state chart modelling, structural dynamic modelling, visualisation, environment. The comparison of these features is based on the ARGESIM Benchmarks for Modelling and Simulation Approaches, which investigate different model approaches and different simulation techniques by means of implementations in various simulators.

Keywords: Simulation software, CSSL standard, Modelica standard, hybrid structures, structural dynamic systems, feature comparison

1. ARGESIM BENCHMARKS ON MODELLING AND SIMULATION APPROACHES

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 Techniques*. Up to now, 20 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 and approaches solutions to these comparisons were used, mainly of the following comparisons:

- 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

.

This contribution mainly concentrates on Benchmark C5 *Constrained Pendulum*, because is is a small model and comparison results can be documented in a concentrated manner. At present, further benchmarks are in preparation, among them an extended benchmark for hybrid systems. Detailed information about definitions and solutions to these benchmarks can be found in **SNE**, www.argesim.org.

2. CLASSICAL FEATURES OF SIMULATORS

2.1 CSSL Structure for Simulators

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. This basic structure of a simulator is illustrated in Figure 1; an extended structure with service of discrete elements is given in Figure 2.

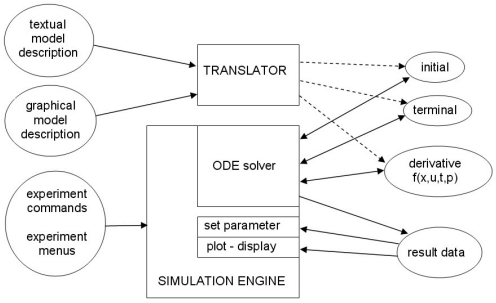


Figure 1: Basic Structure of a Simulator - CSSL

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.

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

2.2 Implicit Models – Differential-Algebraic Equations – DAE Solvers

For a long time the explicit state space description

$$\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t), \vec{u}(t), t, \vec{p}), \quad \vec{x}(t_0) = \vec{x}_0 \tag{1}$$

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):

$$F(\dot{\vec{y}}(t), \vec{y}(t), \vec{u}(t), t, \vec{p}) = \vec{0} \quad \vec{y}(t_0) = \vec{y}_0 \tag{2}$$

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 $\vec{z}(t)$:

$$\begin{aligned} &\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t), \vec{z}(t), \vec{u}(t), t, \vec{p}) = 0, \quad x(t_0) = \vec{x}_0, \\ &g(\vec{x}(t), \vec{z}(t), \vec{u}(t), t, \vec{p}) = 0 \end{aligned} \tag{3}$$

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 $g(x_{n+1}, z_{n+1}) = 0$, $\Phi_I(x_{n+1}, x_n, z_{n+1}, t_{n+1}) = 0$ 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*

2.3 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.
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:

$$\begin{aligned} \vec{x}(t) &= \vec{f}(\vec{x}(t), \vec{u}(t), \vec{p}, t), \\ h(\vec{x}(t), \vec{u}(t), \vec{p}, t) &= 0 \end{aligned} \tag{4}$$

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 (*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

$$\begin{aligned} \dot{\varphi}_1 &= \varphi_2, \quad \dot{\varphi}_2 = -\frac{g}{l}\sin\varphi_1 - \frac{d}{m}\varphi_2, \\ h(\varphi_1, \varphi_2) &= \varphi_1 - \varphi_p = 0 \end{aligned} \tag{5}$$

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

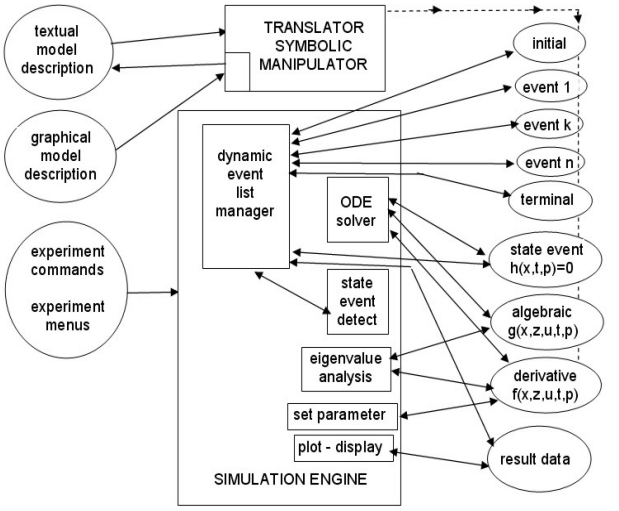


Figure 2: Extended Structure of a Simulation System due to Extensions of the CSSL Standard with Discrete Elements and with DAE Models.

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).

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 (Table 1).

Dymola defines events and their scheduling implicitly by WHEN – or IF - constructs in the dynamic model description, in case of the discussed example e.g. by

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

In case of more complex event descriptions, the WHEN – or IF – clauses are put into an ALGORITHM section similar to ACSL's DISCRETE section.

Table 1: *Constrained Pendulum*: Continuous Model with State Events (ACSL)

```
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
```

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.

2.4 Classical Features of Simulators

Event description (ED), time event handling (THE), state event handling (SEH) and DAE support (DAE) with or without index reduction (IR) became desirable structural features of simulators, supported directly or indirectly. Table 2 compares the availability of these features in the MATLAB / Simulink System, in ACSL and in Dimple, based mainly on evaluations of the ARGESIM Benchmarks.

Table 2: Comparison of Simulators' Classical Features

	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

In Table 2, 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 3).

3. STRUCTURAL FEATURES IN SIMULATORS

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.

3.1 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.

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 3). The graphical model layout corresponds with a textual model representation, shown in Table 3 (abbreviated, simplified).

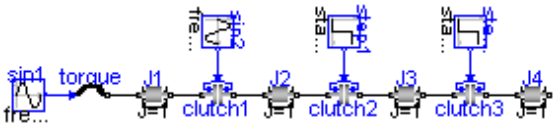


Figure 3: Graphical Modelica Model for Coupled Clutches

Table 3: Textual Modelica Model for Coupled Clutches

```
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;
```

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

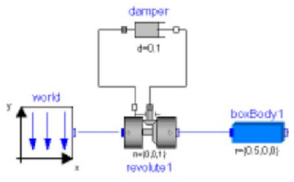
As Modelica also incorporates graphical model elements, the user may choose between textual modelling, graphical modelling, and modelling using elements from an application library. Furthermore, graphical and textual modelling may be mixed in various kinds. The minimal modelling environment is a text 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** (Table 4). As instead of angular velocity, the tangential velocity is used as state variable, the second state event **SE-S** 'vanishes'.

Table 4: Textual Modelica Model for *Constrained Pendulum*

```
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;
```

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 4).




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

Figure 4: Mixed Graphical and Textual Dymola Model for *Constrained Pendulum*

3.2 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 shows a simple use of state chart modelling (Figure 5). 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.

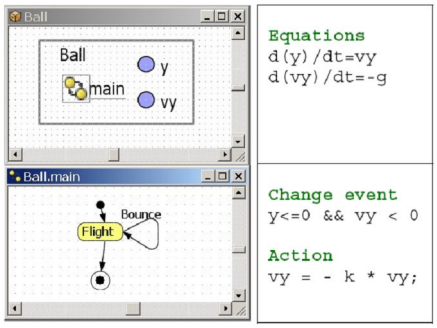


Figure 5: AnyLogic Model for the *Bouncing Ball*

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 5). 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 6). Internally, AnyLogic restarts at each hit the same pendulum model (trivial hybrid decomposition).

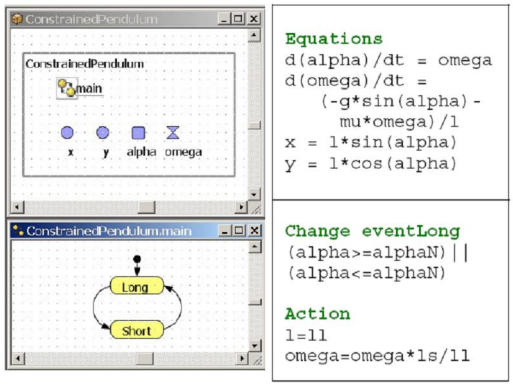


Figure 6: AnyLogic model for *Constrained Pendulum*, Simple Implementation

3.3 Structural Dynamic Systems

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.

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.

The hybrid decomposition approach makes use of state events, 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 these 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. 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.

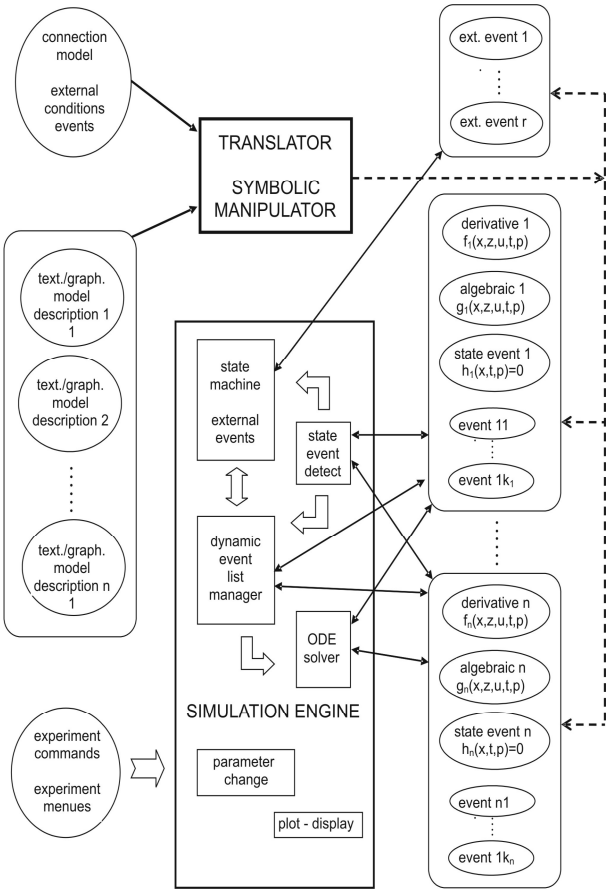


Figure 7: Simulator Structure for *Structural-Dynamic Systems.*

Figure 7 shows a structure for a simulator supporting structural dynamic modelling and simulation. The figure summarises the outlined ideas by extending the CSSL structure by 'connection' models, model-changing state 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. 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.

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. 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
- Efficient implementation of models /control

For instance, for parameter state events (**SE-P**) an implementation within a model may be sufficient, for an event of **SE-S** type implementation with a model change may be advantageous because of easier state re-initialisation, and for a structural model change (**SE-D**) an implementation with hybrid decomposition may be preferred, 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.

3.4 Classification of Structural Features

While the *Classical 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).

Appendix A summarises the availability of these *Structural Features* in some frequently used simulators, and especially in simulators understanding the MODELICA modelling notation (Section 3.1), together with the classical features. Basis for the classification are solutions to specific ARGESIM Benchmarks: Table 5 documents the evaluation of the benchmarks wrt to classical – 'C' -and structural – 'S' - features.

Table 5 also list features for further evaluation:

- 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)

4. IMPLEMENTATION EXAMPLES

As example, structural dynamic implementations of the *Constrained Pendulum* within the experimental simulator *Mosilab* are shown. 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.

Table 5: Evaluation of Classical and Structural Features in specific Benchmarks

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

Mosilab is a generic Modelica simulator, so all classical 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, so that the software model of Mosilab follows the structure in Figure 7.

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)' – indicating that the feature is not available at present (2008), but is scheduled for the future. 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 (except models with contact problems).

Mosilab allows very different approaches for modelling and simulation tasks, to be discussed with the *Constrained Pendulum* example.

Table 6: Mosilab Model for *Constrained Pendulum* – State Chart Switching between Different Models

```
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
end transition;
transition Short -> Long event lengthen
 action
end transition; end ChangePendulum;
```

In a *Mosilab Standard Modelica Model* the *Constrained Pendulum* is defined in the MOSILAB equation layer as implicit law or with graphical blocks as in Dymola (Table 4, Figure 4). In an *Mosilab Model with State Charts*, state charts may be used instead of IF - or WHEN - clauses, with much higher flexibility and readability in case of complex conditions. In a *Structural Dynamic Mosilab Model* state charts may switch

externally between two different pendulum models, controlled externally by a state chart.

Clearly, in case of this simple model, different models would not be necessary. Here, the system is decomposed into two different models, `Short` pendulum model, and `Long` pendulum model (Table 6), switched by external state charts.

REFERENCES

Breitenecker F., Troch I., 2004. Simulation Software – Development and Trends. In: Unbehauen H., Troch I., Breitenecker F., eds. *Modelling and Simulation of Dynamic Systems / Control Systems, Robotics, and Automation.* Oxford: Eolss Publishers, .

Cellier, F.E., *1991. Continuous System Modeling.* New York, Springer,

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

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

Nytsch-Geusen C, Schwarz P, 2005. MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics. *Proc. 4th Intern. Modelica Conference,* 527-535. March 2005, Hamburg.

Strauss J. C. 1967. The SCi continuous system simulation language (CSSL). *Simulation* 9: 281-303.

APPENDIX
Appendix A

Availability of Structural and Classical Features in Simulators and Simulation Systems

	MS - Model Sorting	ED -Event Description	THE – Time Event Handling	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 Dynamic Systems	FA – Frequency Analysis	ENV – Extended Environment
MATLAB	no	no	no	(yes)	(yes)	no	no	no	(yes)	no	no	no	yes	yes	yes
Simulink	yes	(yes)	(yes)	(yes)	(yes)	no	no	(no)	(yes)	no	no	no	no	yes	(yes)
MATLAB/Simulink	yes	yes	yes	yes	(yes)	no	no	(no)	(yes)	no	no	no	yes	yes	yes
Simulink/ Stateflow	yes	yes	yes	yes	(yes)	no	no	(no)	(yes)	no	(yes)	yes	no	yes	(yes)
ACSL	yes	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)	(yes)	no	(no)	(yes)
MathModelica	yes	yes	yes	yes	yes	yes	yes	yes	(yes)	yes	(no)	(yes)	no	(no)	(no)
MathModelica / Mathematica	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	(no)	(yes)	yes	yes	yes
Mosilab	yes	yes	yes	yes	yes	(no)	yes	yes	(no)	(yes)	yes	yes	yes	no	(yes)
Open Modelica	yes	yes	yes	yes	yes	yes	yes	(no)	(no)	yes	(no)	(yes)	no	no	no
SimulationX	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	(no)	(yes)	no	yes	(yes)
AnyLogic	yes	yes	yes	(yes)	(yes)	no	no	no	yes	no	yes	yes	yes	no	no
Model Vision	yes	yes	yes	yes	yes	yes	yes	no	yes	no	yes	yes	yes	yes	no
Scilab	no	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	(yes)	no	no	no
Scilab/ Scicos	yes	yes	yes	yes	yes	(yes)	yes	yes	(yes)	(yes)	yes	(yes)	yes	yes	yes
(MapleSim)	yes	(yes)	(no)	(yes)	yes	yes	yes	yes	yes	yes	no	no	(yes)	(yes)	yes