STRUCTURE OF SIMULATORS FOR HYBRID SYSTEMS - GENERAL DEVELOPMENT AND INTRODUCTION OF A CONCEPT OF EXTERNAL AND INTERNAL STATE EVENTS

Felix Breitenecker, Florian Judex, Nikolas Popper, Inge Troch, Julia Funovits

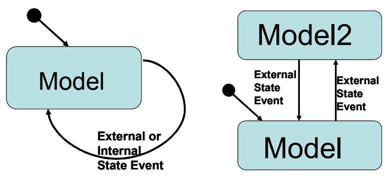
Institute for Analysis and Scientific Computing, Vienna University of Technology,
Wiedner Hauptstrasse 8-10, 1040 Vienna, Austria

Felix.Breitenecker@tuwien.ac.at (Felix Breitencker)

Abstract

New trends for the structure of simulation systems support hybrid modelling and structural dynamic systems. The paper first discusses discrete elements in the CSSL Standard, and - in more detail - classification of 'classic' state events. There, structural-dynamic systems are generated by state events, changing the dimension of the state space. The paper continues with recent developments caused by Modelica and VHDL-AMS, which introduce non-causal modelling on a high level, including implicit models and state events associated with boundary conditions. Both new standards extend the CSSL standard, with emphasis on continuous systems; but especially Modelica allows defining pure discrete model constructs based on events, state charts, and Petri nets.

The main chapters concentrate on further extensions for the CSSL frames, mainly in order to handle hybrid and structural-dynamic systems properly. There, features of two competing 'philosophies' are sketched, maximal state space versus hybrid decomposition. In order to allow a high flexibility at modelling level, state events are characterised as 'internal state events' (I-SE) or 'external state event' (E-SE). An I-SE is handled in a classic way: detecting, localising, event handling, and restart of integration. An E-SE is detected and localised, but then it terminates the simulation of the (previous) model; the event itself is managed outside the model(s), and simulation of the same or of another model is started. Both types of event can be described by state charts (se at left); implementation is the simulator's task. Finally, simulators being able to implement both state event types are reviewed: Modelica/Dymola, Mosilab, AnyLogic, ModelVision, and MATLAB/Simulink/ Stateflow.



Keywords: CSSL Standard, Hybrid Modelling, State Events, State Charts

Presenting Author's biography

Felix Breitenecker studied 'Applied Mathematics' and acts as professor for Mathematical Modelling and Simulation at Vienna University of Technology He covers a broad research area, from mathematical modelling to simulator development, from DES via numerical mathematics to symbolic computation, from biomedical and mechanical simulation to process simulation. He is active in various simulation societies: president and past president of EUROSIM since 1992, board member and president of the German Simulation Society ASIM, member of INFORMS, SCS, etc. He has published about 250 scientific publications, and he is author of two 3 books and editor of 22 books. Since 1995 he is Editor in Chief of the journal editing the journal Simulation News Europe.

1 Introduction

Since early times of simulation, attempts were made to standardise digital simulation programs by means of a self standing structure for simulation systems.

There were some unsuccessful attempts, but in 1968 the CSSL Standard (The CSSL Report commissioned by the Simulation Council Inc - *SCI*) became the milestone in the development: it unified the concepts and language structures of the available simulation programs, it defined a structure for the model, and it describes minimal features for a runtime environment.

In principle, this basic CSSL structure standard has been a standard for almost four decades, although a lot of extensions and other concepts have been developed and discussed. Also modern simulation systems, like Dimple, follow in principle an extended CSSL standard. Mainly these extensions deal with discrete model parts, and with DAE modelling (discussed in Sections 4).

An alternative standardised structure on basis of system theory is Zeigler's hybrid extension of the *DES* formalism (Discrete Event Systems). Unfortunately this approach is not familiar in the area of continuous system modelling, and prototype implementations put emphasis on the discrete world.

Since three years a new challenge is discussed in combined continuous / discrete modelling and simulation, the challenge of structural dynamic systems.

In principle, structural dynamic systems can be modelled and simulated in CSSL structures, but main problem is the fixed state space, as well in the CSSL structure as well as in hybrid DES.

In any case, state events are bridging the gap between the continuous world and the discrete world. On the one side, continuous modeling views state events as interruption of the continuous course of the system, which has to be handled properly, in order to continue continuously. On the other side, discrete system theory puts state events in the foreground, which update states and which control switching between different update algorithms (static algorithmic update, stochastic update base on event mechanisms, or also ODEs and DAEs).

Coming from side of continuous simulation, state events may be viewed in an ambivalent manner. They may cause discontinuous changes within a running algorithm updating the states (ODE solver), or they may cause a termination of the current update algorithm (ODE solver), and starting a new update algorithm (ODE solver) with same or with another model. There, the second view would allow easy modelling and simulation of structural dynamic systems.

It is worth to develop the fore-mentioned idea on basis of a structure with *internal state events* and *external state events*, discussed in chapter 5.

Interestingly, this idea of a distinction of state events is related to the development of Modelica, where *external state events* are discussed as basis for model switching based on state charts. Furthermore, since about five years simulation systems try to implement features for dynamic structures, switched by state events. First, this development used the backdoor by means of simulator coupling, now generic extensions and new systems are available (discussed in chapter 6).

2 CSSL Standard

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. The simulation engine drives the calculations in the time domain. This basic structure of a simulator - due to CSSL standard – is illustrated in Figure 1.

This basic structure does not include any features for discontinuous changes, so that very soon at least time event features were incorporated.

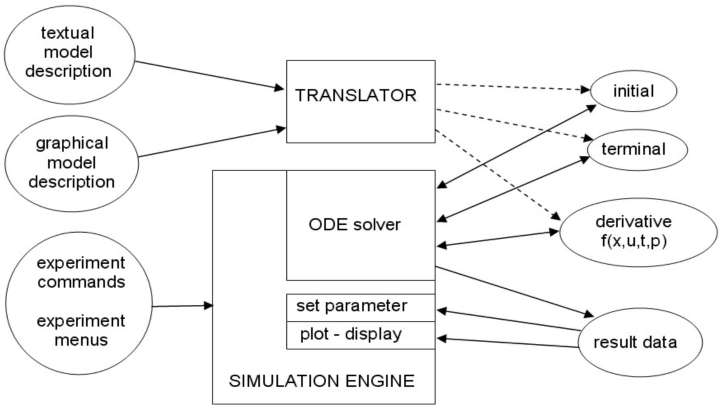


Fig.1 Basic structure of a simulation language due to CSSL standard

3 Discrete elements and events in continuous simulation

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

These discrete section model discrete events, scheduled by time-dependent inputs (time events), or scheduled by state-dependent threshold functions (state events).

3.1 Time Events

In the graphical model description discrete controllers and the time delay could be modelled by a z-transfer block. If a discrete action is more complex, graphical descriptions have problems. SIMULINK offer for this purpose triggered submodels, which can be executed only at one time instant, controlled by a logical trigger signal. New versions of MATLAB also integrate a state machine (State Flow) for event control. Recently (2006) event control is supported in MATLAB/ Simulink by the *SimEvent Blockset*, offering also the entity concept.

In any case, the simulation engine must handle an event list, representing the time instants of discrete action and the calculations associated with the action, where in-between consecutive actions the ODE solver are to be called (see Figure 2, extended structure of a simulator due to CSSL standard).

3.2 State 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, described by a threshold function. This discrete action ('timeless' action) may simple change an input – or the structure of a system.

As example we consider the pendulum with constraints. If the pendulum is swinging, it may hit a pin positioned at angle φ_p with distance l_p from the point of suspension. After hit case the pendulum swings on with the position of the pin as the point of rotation and the shortened length $l_s = l - l_p$. and the angular velocity $d\varphi/dt$ is multiplied at position φ_p by l/ l_s , etc.

These discontinuous changes are state events, because they depend on the state φ. For state events the classical state space description is extended by the state event function $h(x)$, the zero of which determines the event:

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

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

The example involves two different events: change of parameter (length), and change of state (angular velocity). Generally, state events can be classified in four types:

- type 1: parameter change - SE-P
- type 2: one or more inputs change discontinuously - SE-I
- type 3: one or more states change discontinuously - SE-S
- type 4: the dimension of the state vector changes discontinuously - SE-D

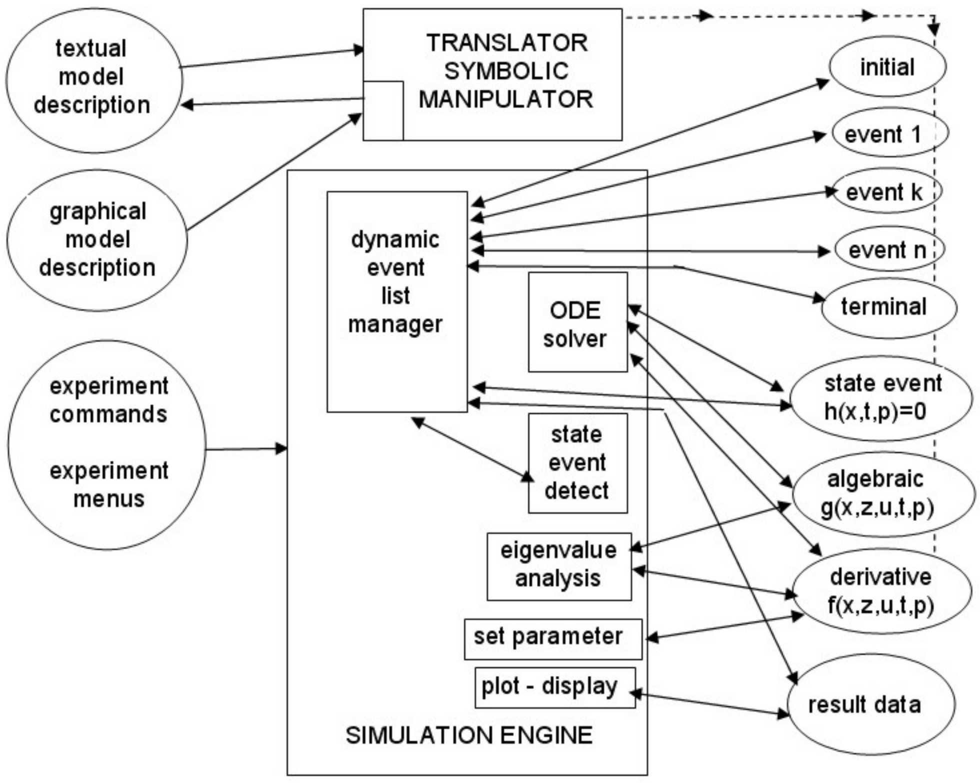


Fig. 2 Extended structure of a simulation system with discrete elements (events)

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 Type 2 (SE-I) are no real state events, they are time events – and listed here due to historic reasons.

State Events Type 3 (SE-S) are essential state events. They must be located, transformed into a time event, and modelled in discrete model parts. In principle, these types of state events cannot exist, because a state variable cannot jump; jumps in states are caused by simplified modelling approaches.

In case of the pendulum, in reality the hit at the pin is not an event changing the velocity; it is a short physical process different to the oscillation process. The whole process may be seen as sequence of different processes: oscillation of long pendulum (differential equations) – hit at pin (event or differential equation) - oscillation of short pendulum (differential equations) – leave from pin (event or differential equation) - oscillation of long pendulum (differential equations) – , etc.

State Events of Type 4 (SE-D) are essential ones and indicate a structural change in the model. In mechanical systems, they indicate a change of degrees of freedom.

Very often the threshold function switches between different algebraic constraints, so that these state events are coupled with differential-algebraic equations. In principle, these events may occur frequently, so that the system is called structural-dynamic, because the dimension of the systems changes quasi-dynamically.

Two philosophies are found in handling these structural dynamic problems: a hybrid decomposition of the process, or making use of frozen states (combined with index reduction algorithms).

3.3 Handling of State Events

In principle, the service (handling) of a state event requires four steps:

1. Detection of the event: usually by checking the change of the signum-function of $h(x)$.
2. Localisation of the event: algorithms make use of either iterative techniques, or of interpolation techniques for determining the time instant of the event with sufficient accuracy.
3. Service of the event: calculating / setting new parameters, inputs and states; switching to new equations
4. Restart of the ODE solver (in a 'maximal' state vector), or starting another model (hybrid decomposition)

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 $h(x)$ needs results from the ODE solver, and the ODE solver calls the root finder by checking the sign of h.

Figure 2, an extension of Figure 1, shows the now more complex structure of calls between model frame, experimental frame, simulation engine and libraries. In principle, the kernel of the simulation engine has become an event handler, managing a complex event list with feedbacks. Furthermore it has to be noted, that not only classic time domain analysis by ODE solvers is offered, but also linear analysis by means of eigenvalue algorithms. Figure 2 also shows an interesting relation to discrete simulation: an event list manager has to be implemented, which can handle also pure discrete systems without any differential equations.

In case of a structural change of the system equations (SE-D) simulators usually can manage only fixed structures of the state space ('maximal' state space with frozen states in case of loss of degrees of freedom).

In textual model description the DISCRETE construct allows to define events of any type, in graphical model descriptions calculations at discrete time instants are difficult to formulate within the continuous input/output form.

3.4 Classic implementations of *Constrained Pendulum* model

The example *Constrained Pendulum* involves state events of type 1 (SE-P: discontinuous change of pendulum length) and of type 3 (SE-S: change of angular velocity). Table 1 presents a classic ACSL model description, which works with two discrete sections `hit` and `leave`, representing the two different modes. These sections model the events, which are scheduled by the `SCHEDULE` statement in the dynamic model description. ACSL makes use of an iterative state event finder based on *regula falsi*.

In pure 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 (Figure 3). This block starts state event detection (interpolation method) depending on the input, the state event function, and outputs a trigger signal. Recent versions of SIMULINK offer an implicit use of such hit crossing facilities in all blocks with switching features.

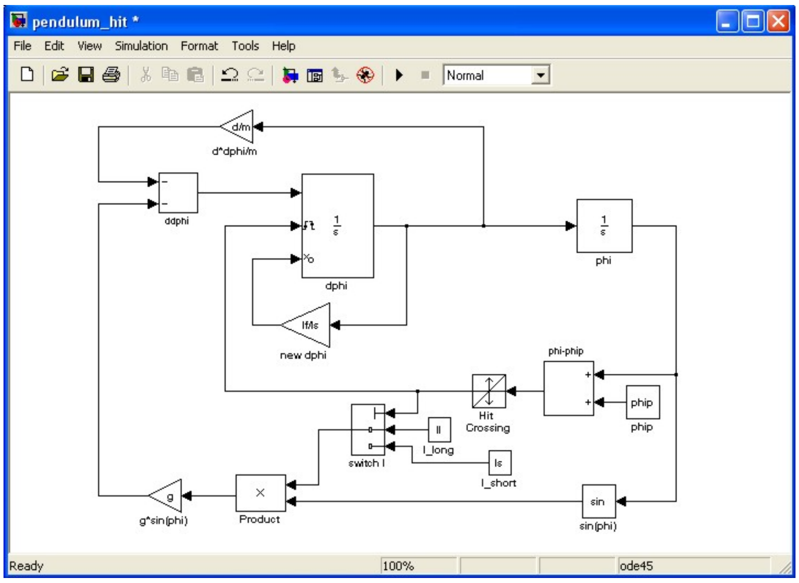


Fig.3 SIMULINK model of Constrained Pendulum example

For restarting the integration with new values for the angular velocity, a formulation going back to the times of analog computation is used: the integrator block is extended by a logical reset signal input; as this signal triggers, the integration is restarted with initial values fed into the initial value input, which is the angular velocity after the hit. In the implementation given, the new angular velocity is calculated continuously, while needed only at the hit event.

A more event-oriented implementation would make use of a triggered subsystem, which is executes only when the trigger from the hit crossing activates the event, calculating new angular velocity and providing new pendulum length..

Table 1: ACSL textual model description for Constrained Pendulum Example

```
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
```

4 From CSSL to physical object-oriented modelling and state chart modelling

In the 1990s, a lot of 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. Furthermore, UML – the Unified Modelling Language – gave new inputs for hybrid modelling.

4.1 Physical modelling in Modelica and VHDL-AMS

In physical modelling, a typical procedure for physical modelling is to cut a system into subsystems and to account for the behaviour at the interfaces. Each subsystem is modelled by balances of mass, energy and momentum and material equations. 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 the purpose of bringing together expertise in object-oriented physical modelling (port based modelling) and defining a modern uniform modelling language. The language is called Modelica. Modelica is intended for modelling within many application domains such as electrical circuits, multibody systems, drive trains, hydraulics, thermodynamical 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 automatically 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 graphical model frame, where the connections are bidirectional physical couplings, and not directed flow. An example demonstrates how drive trains are handled. The drive train consists of four inertias and three clutches, where the clutches are controlled by input signals (Figure 4).

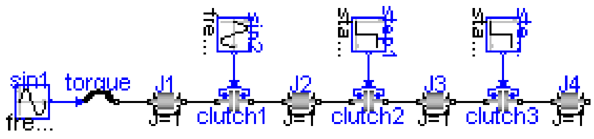


Figure 4 Graphical Modelica model for coupled clutches

The graphical model layout corresponds with a textual model representation, shown in Table 2 (abbreviated, simplified):

Table 2: Textual Modelica model for coupled clutches

```
encapsulated model CoupledClutches
  "Drive train with 3 dynamically coupled
clutches"
  parameter SI.Frequency freqHz=0.2
  ......................................
  Rotational.Inertia J1( J=1, phi(start=0),
w(start=10));
  Rotational.Torque torque;
  Rotational.Clutch clutch1(peak=1.1,
fn_max=20);
  Sources.Sine sin1(amplitude={10},
freqHz={5});
   ....................................
  Rotational.Inertia J3(J=1);
  Rotational.Clutch clutch3(peak=1.1,
fn_max=20);
   ...................................
equation
  connect(sin1.outPort, torque.inPort);
  connect(torque.flange_b, J1.flange_a);
  connect(J1.flange_b, clutch1.flange_a);
   ......................................
  connect(clutch3.flange_b, J4.flange_a);
  connect(step2.outPort, clutch3.inPort);
   .......................................
end CoupledClutches;
```

Modelica can handle very different modelling approaches, not only ODEs and DAEs, but also finite state automata, and Petri nets. By means of state automata or state charts, conditions can be described more clear and transparent (see later).

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.

Up to now – similar to VHDL-AMS – two simulation systems understand Modelica, Dymola from Dynasim, and MathModelica from MathCore Engineering. At present (2006/2007) the University of Lyngby develops and provides a Modelica simulation environment, the *Open Modelica* System, and Fraunhofer Gesellschaft develops a generic simulator, which understands Modelica models and supports variable dynamic structures.

As Modelica defines also graphical model elements, the user may choose between textual modelling, graphical modelling using elements from an application library. Furthermore, graphical and textual modelling may be mixed in various kinds. Figure 5 shows a graphical model for a double pendulum, consisting of two revolute joints (one with damper), and two masses modelling the rods. For joints and masses equations are predefined and sorted together during compilation.

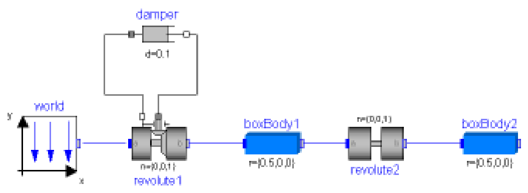


Figure 5 Graphical Modelica model for double pendulum

The model for the constrained pendulum 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 3). As instead of angular velocity, the tangential velocity is used as state variable, the second state event SE-S 'vanishes'. In principle, one could use also graphical modelling for joint and mass using elements as in Figure 5, but the change of length must be formulated textually in an `algorithm` section.

Table 3: 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:=ll; end if;
```

4.2 Modelling events by state charts in AnyLogic

In the end of the 1990s, computer science put the development forward. 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 for Real-Time (UML-RT). By means of UML-RT objects can hold the dynamic behaviour of an ODE. There exist a lot of discrete simulation libraries for discrete simulation, based on the UML notation (class diagrams, state charts, etc). They allow for convenient modelling and simulation of DEVS – Discrete Event Systems.

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, 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. Relationships between active objects set up the hybrid model.

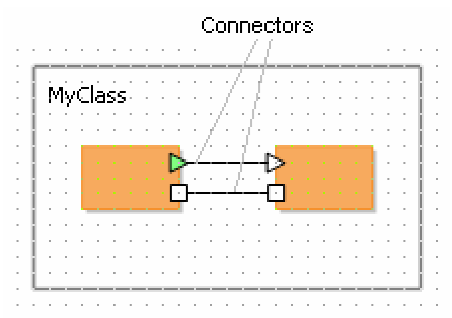
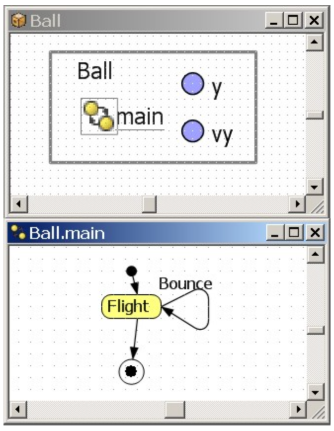


Fig.6 Active objects with connectors exchanging discrete messages (rectangle ports) and continuous signals (triangle ports)

Active objects interact with their surroundings solely through boundary objects: ports for discrete communication, and variables for continuous communication. The activities within an object are usually defined by statecharts (extended state machine). While discrete model parts are described by means of statecharts, events, timers and messages, the continuous model parts are described by means of ODEs and DAEs in CSSL-type notation and with state charts within an object.

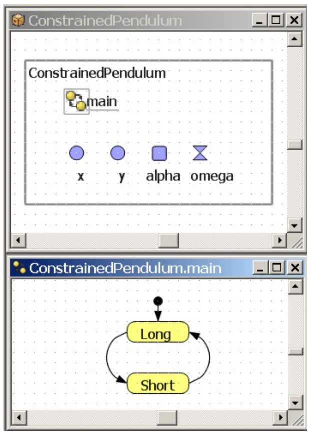
The following AnyLogic implementation of the *Bouncing Ball* example shows a simple use of statechart modelling (Figure 7). The equations are defined in the active object ball, together with the state chart ball.main. This state chart describes the interruption of the state flight (without any equations) 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;
```

Fig.7 AnyLogic model for the *Bouncing Ball*

An AnyLogic implementation for the *Constrained Pendulum* may follow the implementation for the bouncing ball. A (main) active object 'holds' the equations for the pendulum, together with a state chart switching between short and long pendulum. The state chart nodes are empty, the arcs define the events (Figure 8).



```
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
```

Fig.8 AnyLogic model for *Constrained Pendulum*, simple implementation

5 Hybrid and structural-dynamic systems

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) have 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 power-based block descriptions (Modelica, VHDL-AMS). In discrete simulation we meet very different techniques for the model frame.

Application-oriented flow diagrams, network diagrams, state diagrams, etc. allow describing complex behaviour of event-driven dynamics. Usually these descriptions are mapped to an event-based description.

On the other side, the simulator kernel is similar for discrete and continuous simulators. The model description is mapped to an event list with adequate update functions of the states within state update events. In discrete simulation the states are usually the status variables of servers and queues in the model, and state update is simple increase or decrease by increments; complex logic conditions may accompany the scheduling of events.

In continuous simulation the state space is based of various laws used in the application area, and usually defined by differential-algebraic equations. DAE solvers generate a grid for the approximation of the solutions. This grid drives an event list with state update events using complex formula depending on the chosen DAE solver and on the defined DAE. Additional time events and state events are inserted into the global event list.

Hybrid systems often come with 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 structure 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 a certain periods. The other one say 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, 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. 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*.

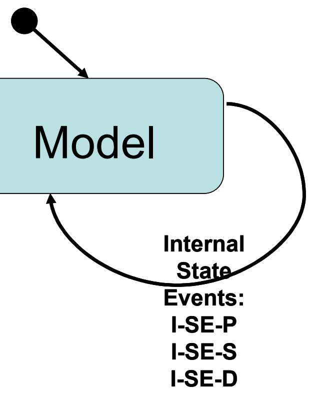


Fig.9 State chart control for *internal events* of one model

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 9).

Modelica, 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. MATLAB / Simulink generates also a maximal state space.

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 10 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 E-SE-D. As special case, this technique may be also used for serial conditional 'execution' of one model – Figure 11 (only for SE-P and SE-S).

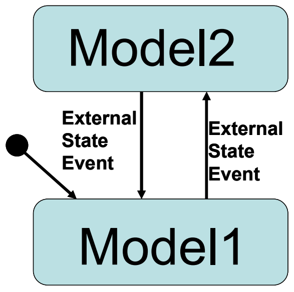


Fig.10 State chart control for *external events* for two models

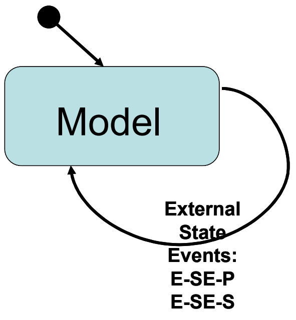


Fig.11 State chart control for *external events* for one models

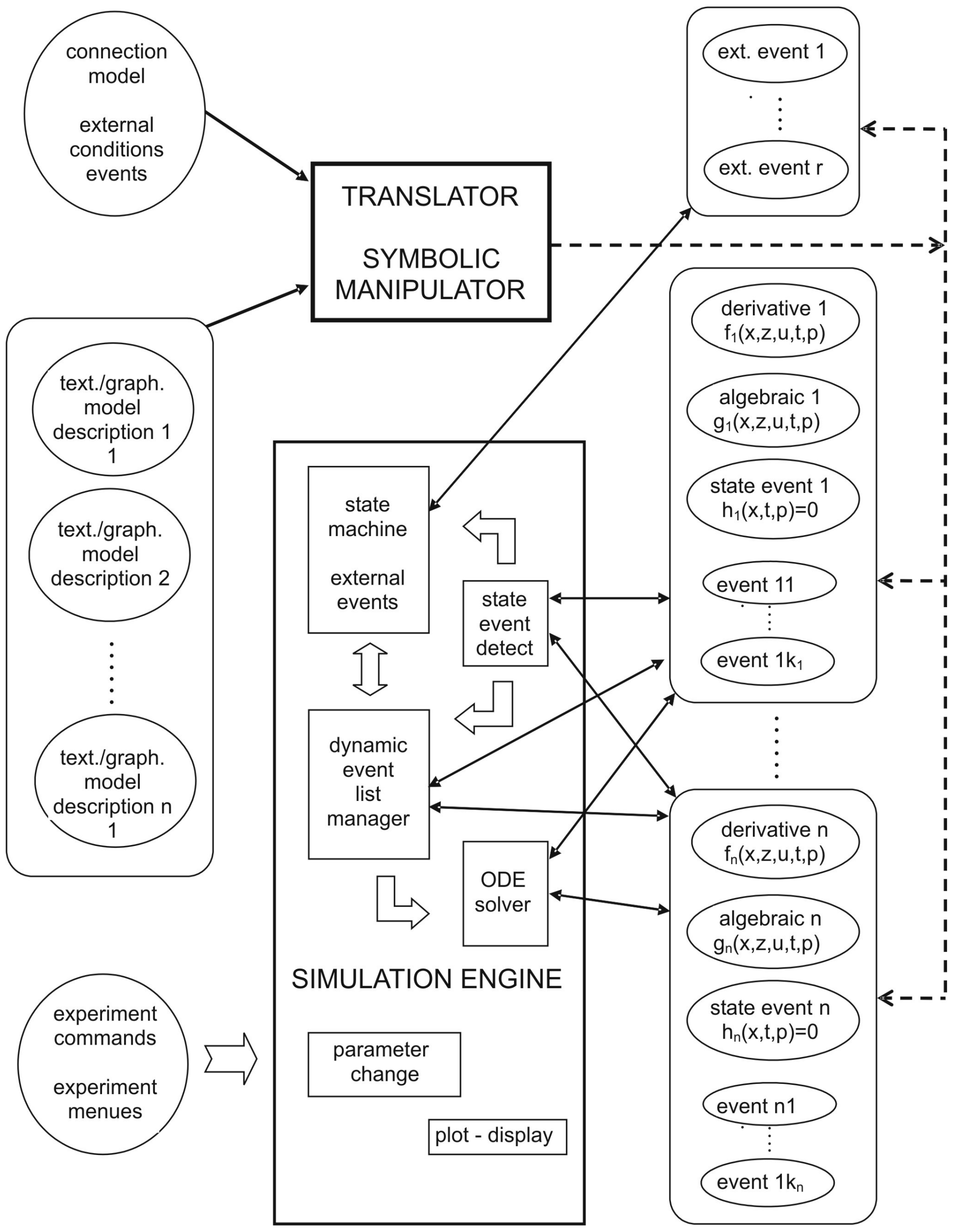

connection model
external conditions events
TRANSLATOR
SYMBOLIC MANIPULATOR
ext. event 1
ext. event r
text./graph. model description 1 1
text./graph. model description 2
text./graph. model description n 1
derivative 1 $f_1(x,z,u,t,p)$
algebraic 1 $g_1(x,z,u,t,p)$
state event 1 $h_1(x,t,p)=0$
event 11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

Fig. 12 Structure for a simulation system with *external state events* E-SE and classical internal state events I-SE for controlling different models.

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, etc. to be processed in serial or also in parallel, so that also co-simulation can be formulated based on external events.

This approach allows handling all events also outside the ODE solver frame. After an event, a totally 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 12 shows a structure for a simulator supporting this hybrid approach. Some work has to be investigated into extension of e.g. Modelica for using this external control of models. The figure summarises the outlined ideas by extending the CSSL structure by control model, external events and multiple models.

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.

5.3 Mixed approach with internal and external events

A simulator structure as proposed in Figure 12 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 13). Both approaches have advantages and disadvantages. The classical Dymola approach generates a fast simulation, because of the monolithic program. But 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 really 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

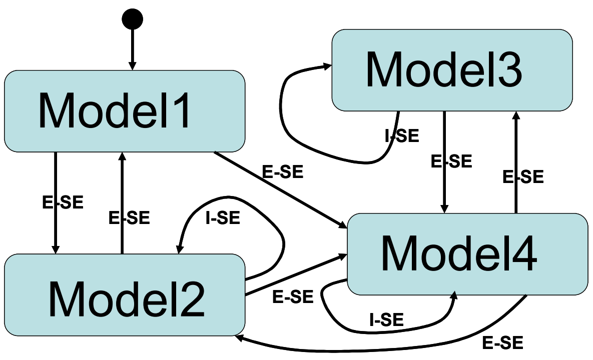


Fig.13 State chart control for different models with *internal* and *external* events

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 Simulators for hybrid and structural-dynamic systems

Up to now no simulator fulfils the structure given in Figure 12 completely. The main questions are

- whether a-causal physical modelling is supported,
- whether a-causal physical modelling is obeying the Modelica standard,
- whether external events are supported (equal to whether hybrid decomposition into independent submodels is possible),
- and whether state chart modelling or a similar construct is supported.

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.

6.1 MATLAB / Simulink

The mainly interpretative systems MATLAB / Simulink offers different approaches. First, it allows hybrid decomposition at MATLAB level. 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). For control, in MATLAB only if-then-else constructs are available (Table 4 and Figure 14) .

Table 4 MATLAB control in *Constrained Pendulum* example for *external events* witching between ling and short pendulum

```
if ((phi_p-phi0)*phi_p<0 |
                  (phi0==phi_p & phi_p*v>0))
    dphi0=v/ls;
    sim('pendulum_short',[t(length(t)),10]);
    v=dphi(length(dphi))*ls;
else
    dphi0=v/l;
    sim('pendulum_long',[t(length(t)),10]);
    v=dphi(length(dphi))*l;
end
```

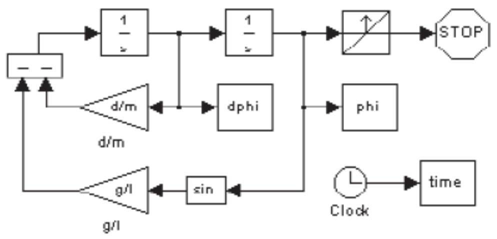


Fig.14 Simulink model for *Constrained Pendulum* with *external event* detected by hit-crossing block

At Simulink level, different submodels may be controlled by Stateflow, Simulink's state chart modelling tool (Figure 15). But the system generates in any case a maximal state space. In both cases, a-causal modelling is not supported.

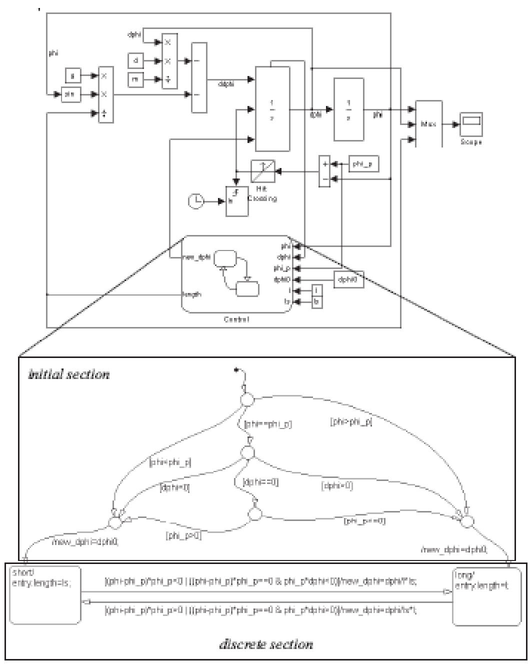


Fig.15 Simulink model for *Constrained Pendulum* with *external event* detected by hit-crossing block and controlled by Stateflow

6.2 Dymola / Modelica

Modelica and Dymola have already been discussed in Section 4, together with examples also for the *Constrained Pendulum* example. Modelica clearly offers a-causal modeling, and so Dymola does.

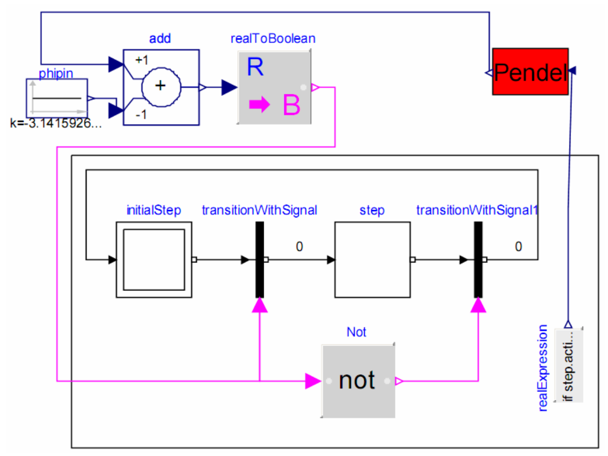


Fig.16 Graphical Dymola model for *Constrained Pendulum* with *internal evenst* managed by elements of Dymola's state chart library

But the Modelica definition says nothing about structural-dynamic systems, and Dymola builds up a maximal state space. Up to now, Modelica does not directly define state charts, and in Dymola a state chart library is available, but working only with internal events within the maximal state space. Figure 16 shows a Constrained Pendulum implementation with Dymola's state chart library.

6.3 Mosilab / Dymola

At present Fraunhofer Gesellschaft Dresden develops a generic simulator *Mosilab*, which defines an extension to Modelica: multiple models controlled by state automata and coupled serially. This simulator meets most of the challenges for the hybrid decomposition approach: 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). Also parameter events may be handled in both manners.

As first example, a model is presented, which describes the simplified dynamics of a landing device, which is falling and slowing down alternatively. The state chart in Figure 17 is translated into extended Modelica (textual) model description given in Table 5.

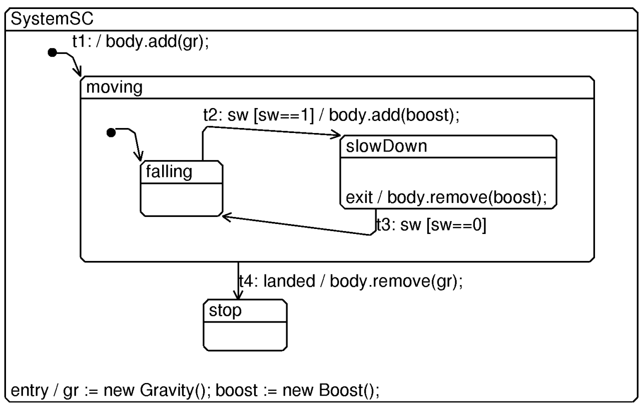

SystemSC
t1: / body.add(gr);
moving
t2: sw [sw==1] / body.add(boost);
slowDown
falling
exit / body.remove(boost);
t3: sw [sw==0]
t4: landed / body.remove(gr);
stop
entry / gr := new Gravity(); boost := new Boost();

Fig. 17 State chart for dynamics of landing device, Modelica Extension in Mosilab

The dynamic models for the different phases may be modelled textually in Modelica standard or using elements from a graphical Modelica library. Mosilab translates each model separately, and generates a main simulation program from the state chart, 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 it does not perform index reduction, so that a-causal modelling is supported only at a lower level.

Table 5 Textual state chart notation for dynamics of landing device, Modelica Extension in Mosilab

```
model System
   statechart
   state SystemSC extends State;
                                // State is basic class
      state Moving extends State;
                 // Definition State "Moving"
         state SlowDown extends State;
         ...
         end SlowDown;
         State falling, State start(isInitial=true);
                                     // Intro of States
         SlowDown slowDown;                         //
         ...
         transition t2 : falling -> slowDown
                                 // state transition t2
         event sw guard sw==1 action body.add(boost)
         end transition;
         transition t3 : slowDown -> falling
                                 // state transition t3
            event sw guard sw==0
         end transition;
      end Moving;
      State stop, start(isInitial=true);
      Moving moving;
      entry action  // executed, if state
                                    SystemSC activ
         gr := new Gravity();
         boost := new Boost(empty=false);
      end entry;
      ...
   end SystemSC;
end System;
```

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

In a standard Modelica approach, the *Constrained Pendulum* is defined in the MOSILAB equation layer as implicit law (it is non necessary to transform to an explicit state space); 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 (Table 6).

Table 6 Mosilab model for *Constrained Pendulum* – standard Modelica approach with *internal events* (I-SE-P)

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

MOSILAB state chart approaches model discrete elements by state charts, which may be used instead of if- or when- clauses, with much higher flexibility and readability in case of complex conditions. There, Boolean variables define the status of the system and are managed by the state chart.

The state charts initialise the system (`Initial` state) and manage switching between long and short pendulum, by changing the length appropriately (Table 7).

Table 7 Mosilab model for *Constrained Pendulum* – state chart model with *internal events* (I-SE-P)

```
event Boolean lengthen(start=false),
shorten(start = false);
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;
statechart
state LengthSwitch extends State;
State Short,Long,Initial(isInitial=true);
transition Initial -> Long end transition;
transition Long -> Short event shorten action
length := ls;
end transition;
transition Short -> Long event lengthen action
length := l1;
end transition; end LengthSwitch;
```

From the modelling point of view, this description is equivalent to the description with if-clauses. The Mosilab translator clearly generates there an implementation with different internal equations. Mosilab's simulator performs simulation by handling the state event within the integration over the simulation horizon.

Mosilab's state chart construct is not only a good alternative to if- or when - clauses within one model, it offers also the possibility to switch between structural different models. This very powerful feature allows any kind of hybrid composition of models with different state spaces and also of different type (from ODEs to PDEs, etc.) see Table 8.

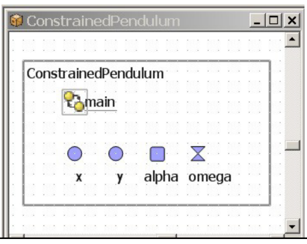
Table 8 Mosilab model for *Constrained Pendulum* – state chart switching between different pendulum models by *external events* (E-SE-P)

```
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 startState -> Long action
L:=new Long(); K:=new Short(); add(L);
end transition;
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 ChangePendulum;
```

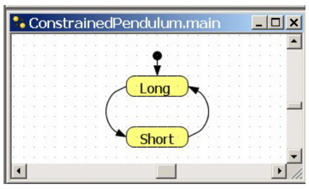
In case of the constrained pendulum, the system is decomposed the system into two different models, `Short` pendulum model, and `Long` pendulum model, controlled by a state chart. The model description (Table 6) defines now first the two pendulum models, and then the event as before. The state chart creates first instances of both pendulum models during the initial state (`new`). The transitions organise the switching between the pendulums (`remove`, `add`). The `connect` statements are used for mapping local states to global state variables.

6.4 AnyLogic

AnyLogic, already discussed in Section 4, is based on hybrid automata. Consequently hybrid decomposition and control by external events is possible. AnyLogic can deal partly with implicit systems, but does not support a-causal modelling. Furthermore, new versions of AnyLogic concentrate more on discrete modelling and modelling with System dynamics, whereby state event detection has been sorted out. For the *Constrained Pendulum* example, a hybrid decomposed model may make use of a model 'similar' to that one in Figure 7, but now two sets of the state equations are found in the substates `Short` and `Long`. The events defined at the arcs stop the actual model, set new initial conditions and start the alternative model (Figure 18).



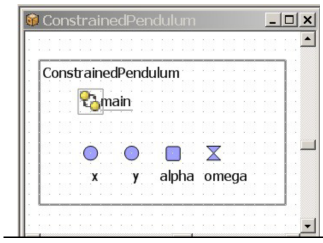
```
Equations
x = l*sin(alpha)
y = l*cos(alpha)
```



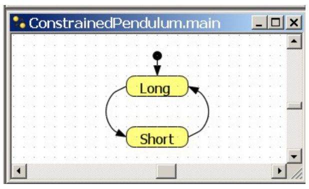
```
Equations
d(alpha)/dt = omega
d(omega)/dt =
    (-g*sin(alpha)-
      mu*omega)/ls

Change eventLong
(alpha>=alphaN)||
(alpha<=alphaN)
Action
omega=omega*ls/ll
stop

Change eventShort
(alpha>=alphaN)||
(alpha<=alphaN)
Action
omega=omega*ll/ls
stop

Equations
d(alpha)/dt = omega
d(omega)/dt =
    (-g*sin(alpha)-
      mu*omega)/ll
```

Fig.18 AnyLogic model for *Constrained Pendulum*, hybrid model decomposition with two pendulum models and *external event*

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. For instance, in *Constrained Pendulum* example, the ODE for the angle, which is not effected by the events, may be put in the main model (Figure 19).



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



```
Equations
d(omega)/dt =
    (-g*sin(alpha)-
      mu*omega)/ls

Change eventLong
(alpha>=alphaN)||
(alpha<=alphaN)
Action
omega=omega*ls/ll
stop

Change eventShort
(alpha>=alphaN)||
(alpha<=alphaN)
Action
omega=omega*ll/ls
stop

Equations
d(omega)/dt =
    (-g*sin(alpha)-
      mu*omega)/ll
```

Fig.19 AnyLogic model for *Constrained Pendulum*, hybrid model decomposition with two models for angular velocity and overall model for angle, controlled by *external event*

6.5 Model Vision Studium

Model Vision Studium (for short MVS) – is an integrated graphical environment for fast and safe designing of interactive models of complex dynamical systems and experimenting on them.

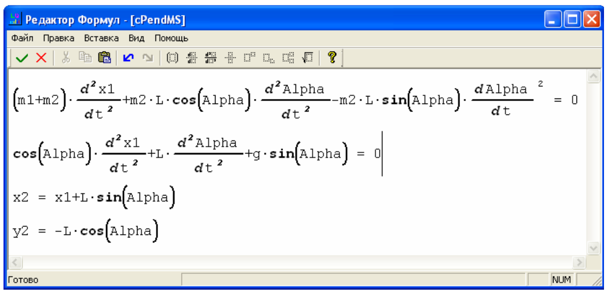
Development of MVS started in the 1990ies at Technical University of St. Petersburg, for end of 2007 an English version is to be released. Basis of MVS are hybrid statecharts, allowing any parallel, serial, and conditional combination of continuous models, described by DAEs. State models itself are objects to be instantiated in various kinds, so that structural-dynamic systems of any kind can be modeled. DAE modeling is supported by an editor capable of editing mathematical formula.

For MVS, a subset of *UML Real Time* was chosen and extended to incorporate continuous behavior.

The modeling language implemented in the tool supports two types of UML diagrams: collaboration diagrams and state chart (state machine) diagrams with some changes. In collaboration diagrams, unidirectional continuous connections between objects (capsules in UML-RT) and the corresponding interface elements – input and output variables – have been added. UML state charts are made hybrid: a system of algebraic-differential equations over variables (interface or object's internal ones) can be associated with each simple or composite state. To make such an UML-based model fully executable Java as a reasonably high-level language for defining data types and data transformation has been chosen.

In principle, MVS and AnyLogic have been developed in parallel. The continuous elements in AnyLogic have been taken from MVS, because AnyLocis started as pure discrete simulator. Since 2007, some advanced hybrid features of MVS, e.g. state event location by iterative algorithms, cannot be supported any more longer in AnyLogic, so that MVS has more features for hybrid modeling and external events than AnyLogic.

MVS supports a-causal modeling on textual level (Figure 20), using a formula editor for DAE systems. Connections between submodels are at present unidirectional, allowing only predefined input-ouptput relations. The model description is mathematically oriented, and does not follow Modelica standard.



$$(m1+m2)\cdot\frac{d^2 x1}{dt^2}+m2\cdot L\cdot\cos(Alpha)\cdot\frac{d^2 Alpha}{dt^2}-m2\cdot L\cdot\sin(Alpha)\cdot\frac{d Alpha}{dt}^2 = 0$$

$$\cos(Alpha)\cdot\frac{d^2 x1}{dt^2}+L\cdot\frac{d^2 Alpha}{dt^2}+g\cdot\sin(Alpha) = 0$$

$$x2 = x1+L\cdot\sin(Alpha)$$

$$y2 = -L\cdot\cos(Alpha)$$

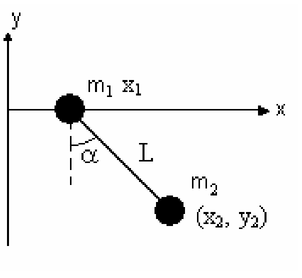
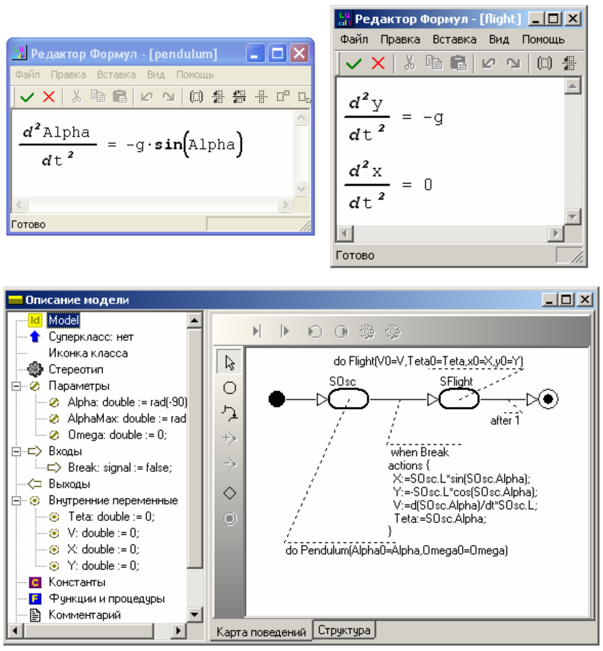


Fig.20 MVS model for pendulum with free suspension – model definition with physical laws (DAEs)

State charts are similar to AnyLogic, each 'state' may consists of different implicit state space descriptions. State charts may also be used to define complex experiments, calling one ore more models with different parameters in a loop.

As example, the a breaking pendulum is described by two states pendulum and flight, with different state spaces, and a state chart handling the external event of type E-SE-D (Figure 21).



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

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

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

```
Alpha: double := rad(-90)
AlphaMax: double := rad
Omega: double := 0;
Break: signal := false;
Teta: double := 0;
V: double := 0;
X: double := 0;
Y: double := 0;
```

```
do Flight(V0=V,Teta0=Teta,x0=X,y0=Y)
when Break
actions {
 X:=SOsc.L*sin(SOsc.Alpha);
 Y:=-SOsc.L*cos(SOsc.Alpha);
 V:=d(SOsc.Alpha)/dt*SOsc.L;
 Teta:=SOsc.Alpha;
}
do Pendulum(Alpha0=Alpha,Omega0=Omega)
```

Fig.21 MVS model for breaking pendulum - hybrid model decomposition into pendulum and flight model, controlled by an *external event* of type E-SE-D

6.6 SCILAB / SCICOS

SCILAB/SCICOS is an open source alternative to MATLAB / Simulink. The developers of this system discuss extensions in two directions:

- Extending the model description by Modelica models (textually and graphically), and
- refining the if-then-else – and when – clause by introducing different classes of associated events, resulting in clauses being as capable as state charts.

With these extensions, would fulfil all requirements, from hybrid decomposition to Modelica standard.

7 References

[1] Strauss, J. C. 'The SCi continuous system simulation language (CSSL)', *Simulation* 9, 281-303. San Diego: SCS Publishing, 1967.

[2] P. Fritzson: *Principles of Object-Oriented Modeling and Simulation with Modelica,* Wiley IEEE Press, ISBN 0-471-471631, 2005.

[3] C. Nytsch-Geusen, and P. Schwarz, 'MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics', In *Proc. 4th Intern. Modelica Conference TU Hamburg-Harburg*, pp 527 – 535, 2005;.

[4] F. Breitenecker, and I. Husinsky I. (Eds.), *SNE – Simulation News Europe*, Vienna, 1992 – now.