INTERNAL/EXTERNAL EVENT – STRUCTURE IN SIMULATORS – CASE STUDIES WITH ARGESIM COMPARISONS

Felix Breitenecker[a], Siegfried Wassertheurer[b], Štefan Emrich[c], Nikolas Popper[d], Günther Zauner[e]

[a] [c] [e]Vienna Univ. of Technology, Austria
[b]ARCS - smart Biomedical Systems, Austria
[d] [e]"die Drahtwarenhandlung" Simulation Services, Austria

[a]Felix.Breitenecker@tuwien.ac.at, [b]siegfried.wassertheurer@arcsmed.at, [c]semrich@aurora.anum.tuwien.ac.at, [d]niki.popper@drahtwarenhandlung.at, [e]Guether.Zauner@drahtwarenhandlung.at,

ABSTRACT
Object-oriented approaches, UML – notation, DAE modelling, variable structure modelling, Modelica notation and other developments have extended the CSSL standard for simulation languages essentially. After a review of the extended CSSL structure, this contribution classifies state events in continuous system modelling and develops a concept of internal / external state events, which allows modelling of structural dynamic systems in a proper way. Additionally, this concept can be 'mapped' onto the structure of several simulation languages. The application of the concept is documented by several implementation examples.

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

1. INTRODUCTION – CLASSIC CSSL STRUCTURE
Simulation supported various developments in engineering and other areas, and simulation groups and societies were founded. One main effort of such groups was to standardise digital simulation programs and to work with a new basis: not any longer simulating the analog computer, but a self-standing structure for simulation systems. There were some unsuccessful attempts, but in 1968, the CSSL Standard 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.

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 3.

In CSSL's model frame, a system can be described in three different ways, as an interconnection of blocks, by mathematical expressions, and by conventional programming constructs as in FORTRAN or C.

Mathematical basis is for the simulation engine is the state space description

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

which is used by the ODE solvers of the simulation engine. Any kind of textual model formulation, of graphical blocks or structured mathematical description or host languages constructs must be transformed to an internal state equation of the structure given above, so that the vector of derivatives $\vec{f}(\vec{x}, \vec{u}, t, \vec{p})$ can be calculated for a certain time instant $\vec{f}_i = \vec{f}_i(\vec{x}(t_i), \vec{u}(t_i), t_i, \vec{p})$. This vector of derivates is fed into an ODE solver in order to calculate a state update $\vec{x}_{i+1} = \Phi(\vec{x}_i . \vec{f}_i, h)$, h stepsize (all controlled by the simulation engine).

Essential is CSSL's concept of SECTIONs or REGIONs, giving a certain structure to the model description. First, CSSL defines a set of operators like INTEG, which formulates parts of the state space description for the system governing ODEs. Other memory operators like DELAY for time delays, TABLE functions for generating (technical) tables, and transfer functions complete dynamic modelling parts. The dynamic model description builds up the DYNAMIC or DERIVATIVE section of the model description. Mapping the model description onto state space description, requires automatic sorting of the equations (blocks) to proper order of the calculation – an essential feature of the translator.

Sometimes together with the state space equations we also meet parameter equations, parameter dependent initial values, and calculations with the terminal values (e.g. for cost functions in an optimisation). In principle, all this calculations could be done in the dynamic model description, but then they are calculated at each evaluation of the derivative vector of the ODE solver – although they have to be calculated only once.

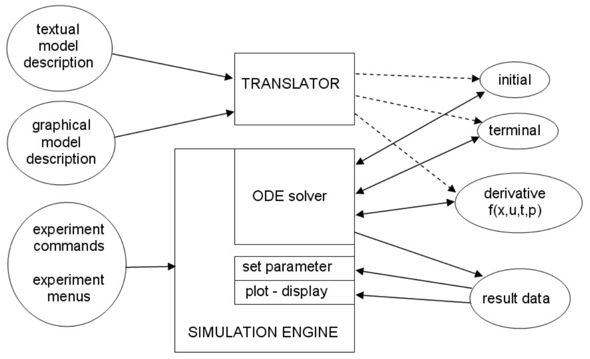


Figure 1: Basic Structure of a Simulation Language due to CSSL Standard

As example, we consider the model description for a pendulum. The well-known equations (length *l*, mass *m*, and damping coefficient *d*) and initial values and parameters are given by

$$\ddot{\varphi}(t) = -\frac{g}{l}\sin\varphi - \frac{d}{m}\dot{\varphi},$$
$$\varphi_0 = \frac{\pi}{n},\ \dot{\varphi}_0 = 0, a = \frac{g}{l}, b = \frac{d}{m} \tag{2}$$

A structured model description in ACSL (Section 5.2) generates efficient code: only the DERIVATIVE section is translated into the derivative vector function, while INITIAL and TERMINAL section are translated into functions called evaluated separately only once (Table 1).

It is task of the translator, to recognise the static elements, and to sort them separately from the dynamic equations, so that for the simulation engine dynamic equations (derivative), initial and parameter equations (initial), and terminal equations (terminal), are provided in separate modules.

Table 1: ACSL Structured Textual Model Description

```
PROGRAM math_pendulum
! --- structured CSSL model --------------------
! --- model parameters -------------------------
  CONSTANT m=1, l=1, d=0.3 ! kg, m, N*s/m
  CONSTANT g=9.81, pi=3.141592653; dphi0=0
  CONSTANT pintel=2
INITIAL ! calculation with parameters-----------
    phi0 = pi/pintel; a = g/l; b = d/m
END ! of INITIAL -------------------------------
  DERIVATIVE ! ODE model --------------------
    phi = integ ( dphi,        phi0)
    dphi = integ (-gdl*sin(phi)-ddm*dphi, dphi0)
  END ! of DERIVATIVE --------------------------
TERMINAL ! calculations with final states ------
    phi_grad = phi*180/pi
END ! of TERMINAL ------------------------------
END ! of Program -------------------------------
```

2. DISCRETE ELEMENTS IN CONTINUOUS SIMULATION – ARGESIM BENCHMARKS

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.

In graphical model description, discrete controllers and the time delay could be modelled by a **z-transfer** blocks, delay blocks and discrete controllers.

New versions of e.g. SIMULINK (Section 5.1) offer more complex discrete model parts, as triggered submodels, which can be executed only at one time instant.

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.

2.1. State Events in Continuous Models

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.

As example, we consider the pendulum with constraints - *Constrained Pendulum*, being one of the so-called ***ARGESIM Benchmarks for Modelling and Simulation*** (published in the journal **SNE – Simulation News Europe**). 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 the point of rotation. The shortened length is $l_s = l - l_p$. and the angular velocity $\dot{\varphi}$ is changed at position φ_p from $\dot{\varphi}$ to $\dot{\varphi} \cdot l / l_s$, etc. These discontinuous changes are state events, not known in advance.

For such 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:

$$\vec{x}(t) = \vec{f}(\vec{x}(t), \vec{u}(t), \vec{p}, t),$$
$$h(\vec{x}(t), \vec{u}(t), \vec{p}, t) = 0 \tag{3}$$

The event actions are usually discontinuous changes of parameters, inputs, states and model equations or model dimension, resp. In this notation, the model for *Constrained Pendulum* is given by

$$\dot{\varphi}_1 = \varphi_2,\ \dot{\varphi}_2 = -\frac{g}{l}\sin\varphi_1 - \frac{d}{m}\varphi_2, \qquad h(\varphi_1,\varphi_2) = \varphi_1 - \varphi_p = 0 \tag{4}$$

The example involves two different events: change of length parameter, and change of state, i.e. angle velocity. 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.

The event actions are discontinuous changes of parameters, inputs, states and model equations or model dimension, resp. Suppose, *t** is the (unknown) time instant of the event, the changes may be given in mathematical notation, where t^{*-} indicates the value before, and t^{*+} the values after the event:

$$\begin{aligned} &p(t^{*-}) \rightarrow p(t^{*+}) \\ &\vec{u}(t^{*-}) \rightarrow \vec{u}(t^{*+}) \\ &\vec{x}(t^{*-}) \rightarrow \vec{x}(t^{*+}) \\ &n(t^{*-}) \rightarrow n(t^{*+}) \end{aligned} \tag{5}$$

In case of the *Constrained Pendulum* the events are hitting and leaving the pin, and the discontinuous changes are given by

$$\begin{aligned} &l(t^{hit^-}) \rightarrow l(t^{hit^{-+}}), \quad l(t^{hit^{-+}}) = l_s \\ &l(t^{leave^-}) \rightarrow l(t^{leave^{-+}}), l(t^{leave^{-+}}) = l \\ &\dot{\varphi}(t^{hit^-}) \rightarrow \dot{\varphi}(t^{hit^{-+}}), \\ &\quad \dot{\varphi}(t^{hit^{-+}}) = \frac{l_s}{l}\dot{\varphi}(t^{hit^-}) \\ &\dot{\varphi}(t^{leave^-}) \rightarrow \dot{\varphi}(t^{leave^{-+}}), \\ &\quad \dot{\varphi}(t^{leave^{-+}}) = \frac{l}{l_s}\dot{\varphi}(t^{leave^-}) \end{aligned} \tag{6}$$

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 step size 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.

2.2. State Event Handling

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 [ti, ti+1]
2. Localization of the event by a proper algorithm determining the time t* when the event occurs and performing the last solver step over [ti, 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*= ti+1, ti+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; Section 5.1) or iterative algorithms are used (ACSL, Section 5.2; Dymola, Section 5.3.1)

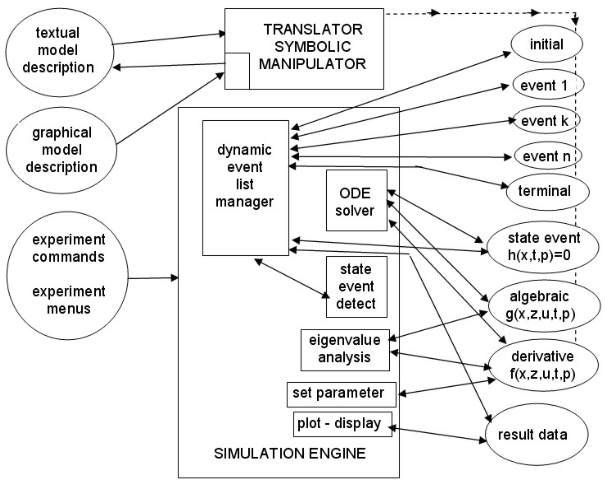


Figure 2: Extended Structure of a Simulation System due to Extensions of the CSSL Standard with Discrete Elements and with DAE Models.

Figure 2 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.

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).

Table 2: *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
```

The *Constrained Pendulum* example involves a state event of type 1 (**SE-P**) and type 3 (**SE-S**). A classical ACSL (Section 5.2) 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 2). Dymola (Section 5.3.1) 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.

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.

2.3. ARGESIM Benchmarks

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.

Some of these benchmarks address state events, hybrid systems, and structural dynamic systems:

- C 3 - Analysis of a Generalized Class-E Amplifier
- C 5 - Two State Model
- C 7 - Constrained Pendulum
- C 11 - SCARA Robot
- C 12 - Collision Processes in Rows of Spheres
- C 13 - Crane Crab with Embedded Control

This contribution mainly concentrates on Benchmark C5 *Constrained Pendulum*, involving state events of type **SE-P** and **SE-S**. With respect to state event types, the following list gives information about occurrence or possible model approaches in benchmarks:

- **SE-P**: C3, C5, C7, C11, C12, C13
- **SE-T**: C3, C12, C13
- **SE-S**: C3, C5, C7, C11, C12, C13
- **SD-D**: C3, C5, C7, C11, C13

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.

3. MODELLING WITH STATE CHARTS

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 *AnyLogic* (Section 5.4). The modelling language 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.

Active objects interact with their surroundings solely through boundary objects: ports for discrete communication, and variables for continuous communication (Figure 3). The activities within an object are usually defined by state charts (extended state machine).

While discrete model parts are described by means of state charts, 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.

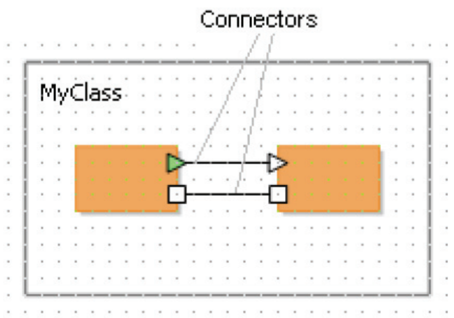
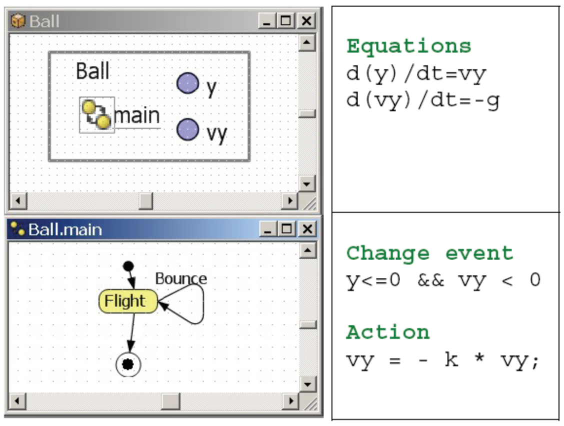


Figure3: Active Objects with Connectors - Discrete Messages (Rectangles) and Continuous Signals (Triangles)

An AnyLogic implementation of the well-known *Bouncing Ball* example shows a simple use of state chart modelling (Figure 4). The model equations are defined in the active object ball, together with the state chart ball.main. This state chart describes the interruption of the state flight (without any equations) by the event bounce (**SE-P** and **SE-S** event) defined by condition and action.



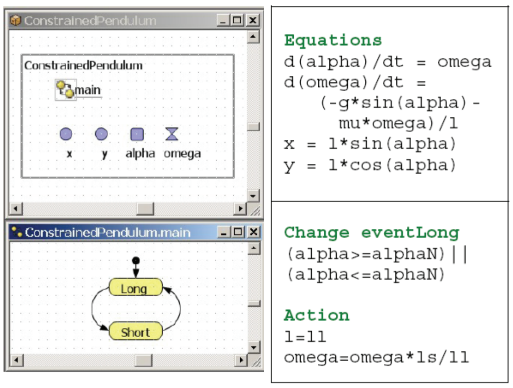
```
Equations
d(y)/dt=vy
d(vy)/dt=-g

Change event
y<=0 && vy < 0

Action
vy = - k * vy;
```

Figure 4: 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. The principle question is, whether state charts are to be seen as comfortable way to describe complex WHEN – and IF – constructs, being part of the model, or whether state charts control different models from a higher level. At present (2008) a free Modelica state chart library 'emulates' state charts by Boolean variables and IF – THEN – ELSE constructs. A further problem is the fact, that the state chart notation is not really standardised; AnyLogic makes use of the Harel state chart type.



```
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=l1
omega=omega*ls/l1
```

Figure 5: AnyLogic model for *Constrained Pendulum*, Simple Implementation

4. HYBRID AND STRUCTURAL-DYNAMIC SYSTEMS

Hybrid systems 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, the *maximal state space*, and the *hybrid decomposition*.

4.1. 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. The 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* – **I-SE**.

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; Fig. 6). VHDL-AMS and Dymola follow this approach, handling also DAE models with index > 1. Discrete model parts are only supported at event level. ACSL and MATLAB / Simulink generate also a maximal state space.

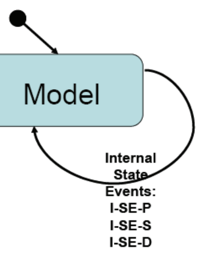


Figure 6: State Chart Control for *Internal Events* of one Model

4.2. Hybrid Decomposition for Structural-Dynamic Systems – External Events

The hybrid decomposition approach makes use of *external events* (**E-SE**), which controls 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. Control for continuous models and for discrete actions can by modelled by state charts. Figure 7 (left) shows the hybrid coupling of two models, which may be extended to an arbitrary number of models, with possible events **E-SE-P**, **E-SE-S**, and **ESE-D**.

As special case, this technique may be also used for serial conditional 'execution' of one model – Figure 7 (only for SE-P and SE-S).

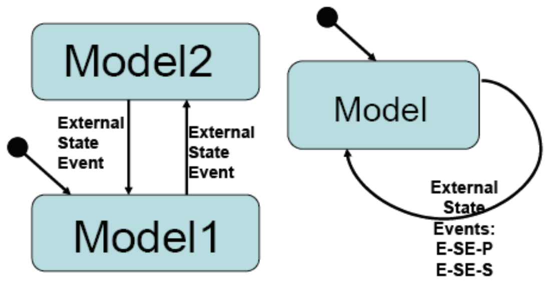


Figure 7: State Chart Control for *External Events* for two Models (left) and for one Model (right).

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. The approach allows handling all events also outside the ODE solver frame. After an event, a very new model can be started. This procedure may make sense especially in case of events of type **SE-D** and **SE-S**. As consequence, consecutive models of different state spaces may be used.

Figure 8 shows a structure for a simulator supporting structural dynamic modelling and simulation. The figure summarises the outlined ideas by extending the CSSL structure by control model, external events and multiple models. The main extension is that the translator generates not only one DAE model; he generates several DAE models from the (sub)model descriptions, and external events from the connection model, controlling the model execution sequence in the highest level of the dynamic event list.

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).

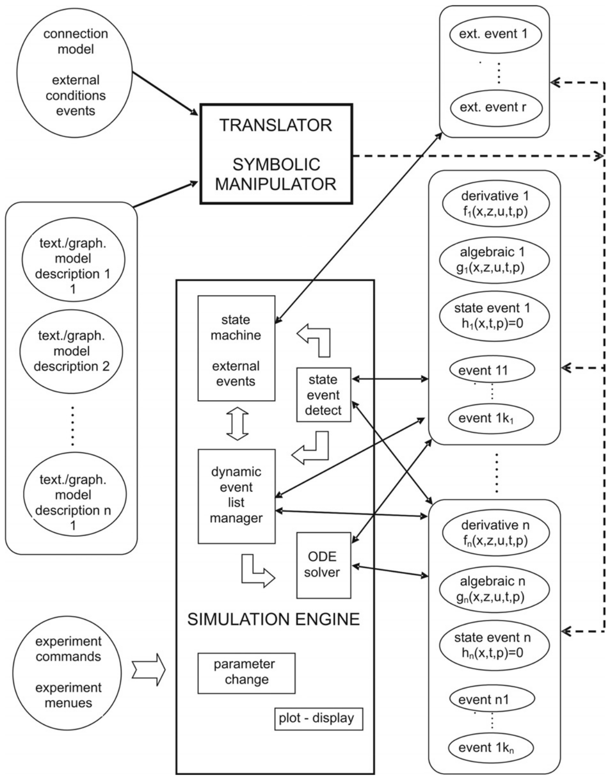

connection model
external conditions events
TRANSLATOR
SYMBOLIC MANIPULATOR
text./graph. model description 1
text./graph. model description 2
text./graph. model description n
state machine
external events
state event detect
dynamic event list manager
ODE solver
SIMULATION ENGINE
parameter change
plot - display
experiment commands
experiment menues
ext. event 1
ext. event r
derivative 1 f1(x,z,u,t,p)
algebraic 1 g1(x,z,u,t,p)
state event 1 h1(x,t,p)=0
event 11
event 1k1
derivative n fn(x,z,u,t,p)
algebraic n gn(x,z,u,t,p)
state event n hn(x,t,p)=0
event n1
event 1kn

Figure 8: Structure for a Simulation System with *External State Events* E-SE and Classical Internal State Events I-SE for Controlling Different Models.

4.3. Mixed Approach with Internal and External Events

A simulator structure as proposed in Figure 8 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 9). 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. 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

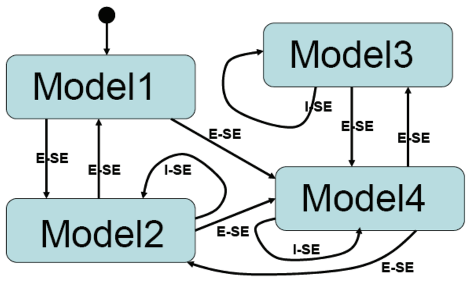


Figure 9: State Chart Control for Different Models with *Internal* and *External Events*.

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.

5. STRUCTURAL FEATURES IN SIMULATORS

Structural dynamic system are up to now – 2008 – a challenge for simulators. In principle, model-compiling simulators must 'emulate' the dynamic structure in a maximal state space by switching between 'active' states, while interpreting simulators can switch between different models by means of a control model handling the structural changes.

But there exist also mixed strategies. In the following, some simulators are discussed with respect to their features for structural dynamic systems. Mainly using the benchmark Constrained Pendulum, in detail features for state chart modelling (as convenient tool for control models) and features for hybrid decomposition are investigated:

- Support of state chart modelling or of a similar construct, by means of textual or graphical constructs.
- Decomposition of structural dynamic systems with dynamic features– features for external events.

5.1. MATLAB / Simulink / Stateflow

The mainly interpretative systems MATLAB / Simulink offer different approaches. First, MATLAB itself allows any kind of static and dynamic decomposition, but MATLAB is not a simulator, because the model equations have to be provided in a sorted manner.

Second, MATLAB allows hybrid decomposition at MATLAB level with Simulink models. There, from MATLAB level, 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 constructs are available. Table 3 – MATLAB control model, and Figure 10 – graphical Simulink model, show a hybrid decomposition of this type for the *Constrained Pendulum.*

Table 3: MATLAB Control Model for Constrained Pendulum with External Events Switching between Long 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
```

MATLAB is a very powerful environment with various modules. Simulink is MATLAB's simulation module for block-oriented dynamic models (directed signal graphs), which can be combined with Stateflow, MATLAB's module for event-driven state changes described by state chart.

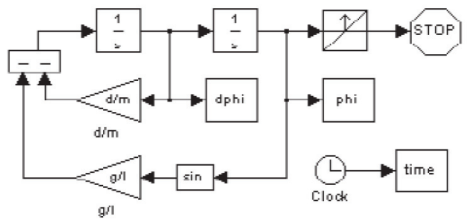


Figure 10: Simulink Model for *Constrained Pendulum* with *External Event* detected by Hit-Crossing Block.

At Simulink level, Stateflow, Simulink's state chart modelling tool, may control different submodels. These submodels may be dynamic models based on ODEs (DAEs), or static models describing discrete actions (events). Consequently, Stateflow can be used for implementation of the *Constrained Pendulum*, where the state charts control length and change of velocities in case of hit by triggering the static changes (Figure 11). A solely Simulink implementation would make use of a triggered submodels describing the events by AND – and OR – blocks, or by a MATLAB function.

Alternatively, for *Constrained Pendulum* Stateflow could control two different submodels representing long and short pendulum enabled and disabled by the state chart control. Internally Simulink generates a state space with 'double' dimension, because Simulink can only work with a maximal state space and does not allow hybrid decomposition.

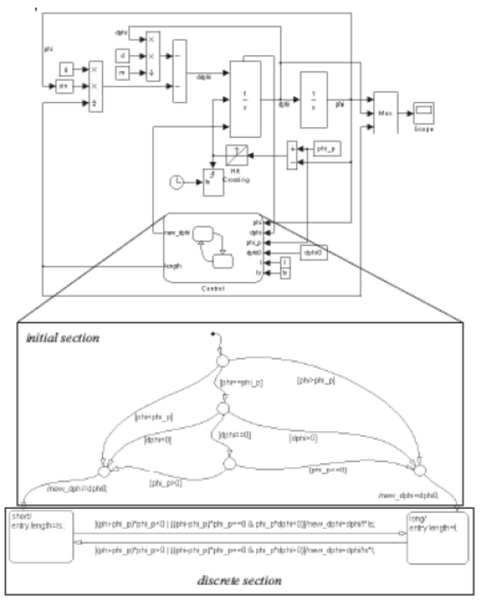


Figure11: Simulink Model for *Constrained Pendulum* with *External Event* (Hit-Crossing Block, Stateflow)

5.2. ACSL

ACSL – Advanced Continuous Simulation Language – has been developed since more than 25 years. ACSL was strongly influenced by the CSSL standard. ACSL' software structure is a direct mapping of the structure in Figure 2. Implementations of the *Constrained Pendulum* have been shown in the previous sections Table 1, Table 2), as example for modelling due to CSSL standard.

A very interesting additional ACSL module is an extended environment called ACSLMath. ACSLMath was intended to have same features as MATLAB; available is only a subset, but powerful enough for an extended environment, which can be used for hybrid decomposition of a structural dynamic model in almost the same way than MATLAB does (the MATLAB model in Table 3 can be used in ACSLMath, only by replacing the `sim` calls by `run` calls).

5.3. MODELICA - Simulators

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.

These ideas stimulated the development of the simulator *Dymola*, whose modelling frame has been extended to a general standardised modelling language 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* serves as a standard format so that models arising in different domains can be exchanged between tools and users.

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

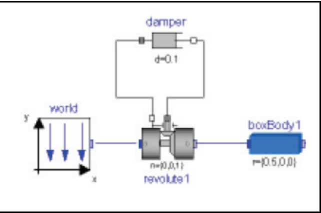
As Modelica also incorporates graphical model elements, the user may choose between textual modelling, graphical modelling, and modelling using elements from an application library. Furthermore, graphical and textual modelling may be mixed in various kinds. The minimal modelling environment is a text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

But *Modelica* allows also 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 12).



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

Figure 12: Mixed Graphical and Textual Modelica Model for *Constrained Pendulum*

5.3.1. Dymola

Dymola was the first Modelica simulator. Dymola, introduced by F. E. Cellier as a-causal modelling language, and developed to a simulator by H. Elmquist, can be called the mother of Modelica. Dymola clearly can understand the Modelica models given in Table 4 and Figure 12. Dymola offers also a Modelica – compatible state chart library, which allows to model complex conditions (internally translated into IF – THEN – ELSE or WHEN constructs). Figure 14 shows an implementation of the *Constrained Pendulum* using this library.

5.3.2. Mosilab

Since 2004, Fraunhofer Gesellschaft Dresden develops a generic simulator *Mosilab*, which also initiates an extension to Modelica: multiple models controlled by state automata, coupled in serial and in parallel. Furthermore, Mosilab puts emphasis on co-simulation and simulator coupling, whereby for interfacing the same constructs are used than for hybrid decomposition. Mosilab is a generic Modelica simulator.

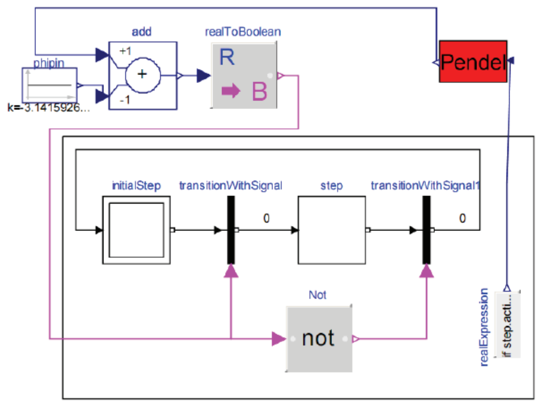


Figure 13: Graphical Dymola Model for *Constrained Pendulum* with *Internal Events* Managed by Elements of Dymola's State Chart Library

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. 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 8. The constructs for the state charts are modifications of state chart modelling in AnyLogic.

Mosilab allows very different approaches for modelling and simulation tasks, to be discussed with the *Constrained Pendulum* example. Three different modelling approaches reflect the distinction between internal and external events as discussed before.

Mosilab Standard Modelica Model. In a standard Modelica approach, the *Constrained Pendulum* is defined in the MOSILAB equation layer as implicit law; the state event, which appears every time when the rope of the pendulum hits or 'leaves' the pin, is modelled in an `algorithm section` with if (or when) – conditions (Table 7).

Table 5: Mosilab Model for Constrained Pendulum – Standard Modelica Approach - *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 I-SE-P Model with State Charts. MOSILAB's state chart approach models 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.

Table 6: 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;
….;
```

Table 6 shows a Mosilab implementation of the *Constrained Pendulum*: state charts initialise the system and manage switching between long and short pendulum, by changing the length appropriately.

Mosilab E-SE-P Model. 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 of different type (from ODEs to PDEs, etc.). Table 7 shows a Mosilab implementation of the *Constrained Pendulum* making use of two different pendulum models, controlled externally by a state chart.

Here, the system is decomposed into two different models, `Short` pendulum model, and `Long` pendulum model, controlled by a state chart. The model description (Table 7) 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 to global state.

Table 7: Mosilab Model for *Constrained Pendulum* – State Chart Switching between Different Pendulums Models by *External Events* (E-SE-P)

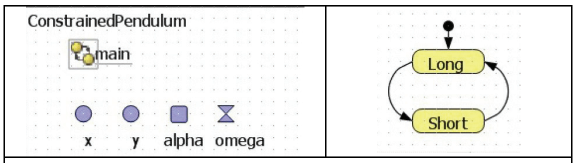
```
model Long
equation
 mass*vdot/l1 + mass*g*sin(phi)+damping*v = 0;
end Long;
model Short
equation
 mass*vdot/ls + mass*g*sin(phi)+damping*v = 0;
end Short;
event discrete Boolean lengthen(start=true),
equation
 lengthen =
 (phi>phipin);shorten=(phi<=phipin);
statechart
state ChangePendulum extends 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;
```

5.4. AnyLogic – Hybrid State Chart Simulator

AnyLogic – already discussed in section 3) is based on hybrid automata. Consequently, hybrid decomposition and control by external events is possible. In AnyLogic, various implementations for the *Constrained Pendulum* are possible. A classical implementation is given in Figure 5, following classical textual ODE modelling, whereby instead of IF – THEN clauses a state chart is used for switching (**I-SE-P**, **I-SE-S**).

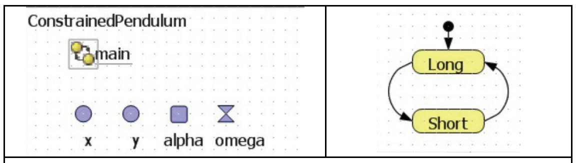
AnyLogic E-SE-P Model with State Charts. A hybrid decomposed model makes use of two different models, defined in substate / submodel `Short` and `Long`. – part of a state chart switching between these submodels. The events defined at the arcs stop the actual model, set new initials and start the alternative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

AnyLogic E-SE-P Model with Parallel Models. AnyLogic works interpretatively, after each external event state equations are tracked and sorted anew for the new state space. This makes it possible, to decompose model not only in serial, but also in parallel (Figure 14).



```
Equations Constrained Pendulum
parameter …
end Constrained pendulum
Equations Short
d(alpha)/dt = omega
d(omega)/dt =(-g*sin(alpha)-mu*omega)/ls
Change eventLong;
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*ls/l1; stop
end Short
Equations Long
d(alpha)/dt = omega
d(omega)/dt =(-g*sin(alpha)-mu*omega)/l1
Change eventShort
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*l1/ls; stop
end Long
```

Figure 13: AnyLogic Model for *Constrained Pendulum*, Hybrid Model Decomposition with two Pendulum Models and *External Events*



```
Equations Constrained Pendulum
d(alpha)/dt = omega
x = l*sin(alpha); y = l*cos(alpha)
end Constrained pendulum
Equations Short
d(omega)/dt =(-g*sin(alpha)-mu*omega)/ls
Change eventLong
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*ls/l1; stop
end Short
Equations Long
d(omega)/dt =(-g*sin(alpha)-mu*omega)/l1
Change eventShort
(alpha>=alphaN)||(alpha<=alphaN)
Action; omega=omega*l1/ls; stop
end Long
```

Figure 14: AnyLogic Model for *Constrained Pendulum*, Hybrid Model Decomposition with Two Models for Angular Velocity and Overall Model for Angle

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, .

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.