In *Proc. 1998 European Simulation Multiconference*, Manchester, 1998, pp. 252-257

Integration of the discrete and the continuous behaviour in the hybrid χ simulator

G. Fábián[1] D.A. van Beek[2] J.E. Rooda[3]
Eindhoven University of Technology, Department of Mechanical Engineering,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
E-mail:{g.fabian[1], vanbeek[2], rooda[3]}@wtb.tue.nl

KEYWORDS

Hybrid simulator, hybrid language, initial state calculation, discrete sub-phase.

ABSTRACT

In the hybrid χ simulator, the integration of the continuous and the discrete behaviour leads to delayed initial state calculation and possibly multiple discrete sub-phases within one discrete phase. In a sub-phase processes evolve independently, using a local copy of the last consistent continuous state, which simplifies reasoning about them. In the paper the algorithm implemented by the χ simulator is described and is illustrated with an example.

1 INTRODUCTION

Hybrid systems exhibit a mixed discrete/continuous behaviour. On the one hand, they behave according to a discrete model which describes how actions are executed during the discrete phases. This is specified in χ in a CSP [Hoare, 1985]-like parallel language. On the other hand, the continuous state of hybrid systems changes according to a continuous model which describes, how the values of the continuous variables are calculated from the continuous state specification. The latter one is given in χ in the form of equations. The two models are not independent; discrete actions can change the continuous state, and the values of the continuous variables can be used in discrete actions, thus altering the discrete behaviour.

In order to fully specify the behaviour of a hybrid system, the two models need to be integrated. The integration is realized by specifying the semantics of the interactions and by describing the evolution of the discrete and the continuous states relative to each other.

This paper describes the integration of the discrete and the continuous models in the χ simulator. First, the continuous model is introduced. Then, this model is integrated into the discrete model, leading to delayed initial state calculation, and division of the discrete phase into sub-phases. The kernel algorithm implemented by the χ simulator is presented with an illustrative example.

2 THE χ LANGUAGE

The χ language is a hybrid specification language that has been designed for modelling discrete-event, continuous-time and hybrid systems. The language is defined with the aid of symbols. These are replaced by ASCII equivalents when models are entered into the simulator.

A χ model consists of a fixed set of processes running concurrently. Processes can be connected by channels, which is the only means by which they can influence each other's behaviour. Channels connect pairs of processes. Discrete channels are used for point-to-point synchronous data communication. Continuous channels connect continuous variables and establish an equality relation between them. Hierarchy is achieved by grouping processes into systems, that specify a fixed connection layout of the contained processes and subsystems.

A process can have a discrete and/or continuous behaviour specification.

```
proc name(parameter declarations)=
⟦ variable declarations
; initialization | links
| DAEs | discrete-event statements
⟧
```

The continuous-time behaviour is given in the form of Differential Algebraic Equations (DAEs). The DAEs are given in a general form

$$\mathbf{g}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{h}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) \qquad (1)$$

which allows cyclic dependency among variables. If equations vary depending on the state of the system (which often indicates a discontinuity) conditional equations are used. At any time in the course of the simulation, the enabled conditional equations together with the unconditional equations form the valid set of DAEs.

The discrete-event behaviour is described in a CSP -based concurrent programming language. This allows sequential, conditional and repetitive composition of actions.

Variables used in a χ model are either continuous or discrete. Continuous variables represent the system dynamics; those that occur differentiated in the DAEs are the differential variables, while others are the algebraic ones. The values of continuous variables are calculated from the DAEs; assignments to them (denoted by the symbol ::=) determine their value only for the current point in time. The values of discrete variables on the other hand are determined by assignments; they hold their values between two subsequent assignments.

Simulation of a χ model can be described as an alternating sequence of continuous and discrete phases. In the continuous phase the simulation time advances and the valid set of DAEs is solved. A discrete phase is launched when the discrete body of some processes becomes active and it lasts until all discrete process bodies are inactive. In the sequel, a process is said to be active or inactive/blocked when its discrete body is active or inactive. To activate or deactivate processes the following kinds of event statements are used:

1. *time-out statement* or so-called *delta statement*: a process executing a statement $\triangle s$, where s is an expression of type real, becomes blocked for s time units. The time-out of a blocking $\triangle$ statement is the time point when the process becomes active again.
2. *state-event specification* or so-called *nabla statement*: a process executing a statement ∇c, where c is a boolean expression (so-called state-event condition) involving at least one continuous variable becomes blocked until c becomes true.
3. discrete *communication* and *synchronization*: suppose process P_1 is connected to process P_2 with channel a. P_1 initiates a communication by sending the value of expression e along channel a in statement $a\,!\,e$. Execution of this statement blocks P_1 until P_2 executes a receive action $a\,?\,y$, where y is a variable in P_2. Communication can be initiated by a receive action as well. Synchronization is a special kind of communication, where no data is exchanged.

The *selective waiting statement* ($[GW]$) is used to allow a process to specify a number of events to wait for. It is denoted by

$$[\ b_1;\ E_1 \longrightarrow S_1\ []\ \ldots\ []\ bn;\ E_n \longrightarrow S_n\ ]$$

The boolean expression b_i ($1 \le i \le n$) denotes a guard, which is open if b_i evaluates to true, and is closed otherwise. An event statement E_i is enabled, if its guard is open and the event can actually take place. (A time-out statement $\triangle s$, can actually take place when s seconds have passed). If there are open guards, execution of the process is blocked until one of the event statements is enabled. If more statements are enabled, priority rules decide which one is selected. Subsequently, the statement S_i associated with the selected event statement is executed. Repetition of a statement $[S]$ is denoted by $*[S]$.

3 CONTINUOUS MODEL

The problem solved by hybrid simulators in continuous phases can be formulated as

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0} \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the vector of differential variables, $\mathbf{y} \in \mathbb{R}^m$ is the vector of algebraic variables, $t \in \mathbb{R}$ is the independent variable and $\mathbf{f} \in \mathbb{R}^{2n+m+1} \to \mathbb{R}^{n+m}$ is the set of DAEs [Barton and Pantelides, 1994].

Let I_k denote the time interval of the k^{th} continuous phase ($k \ge 1$) and t_{k-1} and t_k be its boundaries: $I_k = [t_{k-1}, t_k]$. In this domain the equations have the form $\mathbf{f}^{(k)} \in \mathbb{R}^{2n+m+1} \to \mathbb{R}^{n+m}$. Equation (2) in the k^{th} continuous phase takes the form

$$\mathbf{f}^{(k)}(\dot{\mathbf{x}}^{(k)}, \mathbf{x}^{(k)}, \mathbf{y}^{(k)}, t) = \mathbf{0} \quad t \in [t_{k-1}, t_k] \tag{3}$$

The solution to (3) $(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$ is the value of the continuous variables over I_k.

At time point t_k ($k \ge 1$) a discrete phase takes place, and the discrete body of the processes are executed. If variables (discrete or continuous) that occur in the equations are assigned, the equations may become inconsistent and a consistent initial state may need to be re-established. In fact, within one discrete phase a series of initial state calculations can take place. In order to incorporate this into the above model, the discrete phase at time t_k is divided into $j_k \in \mathbb{N}$ sub-phases. Each sub-phase is enclosed between two initial state calculations as shown in Figure 1 .

3.1 Initial state calculation

In the k^{th} discrete phase, j_k initial state calculations take place. At the end of the i^{th} sub-phase ($i \in [1, j_k]$) the following equation is solved.

$$\mathbf{f}^{(k)(i)}(\dot{\mathbf{x}}^{(k)(i)}, \mathbf{x}^{(k)(i)}, \mathbf{y}^{(k)(i)}, t_k) = \mathbf{0} \tag{4}$$

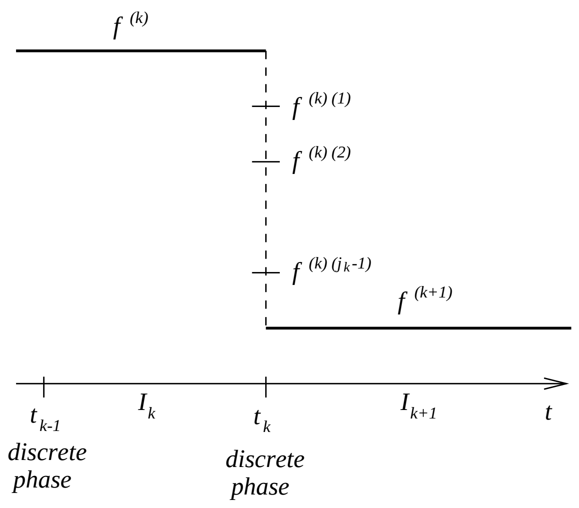

Figure 1: State calculations in χ

The initial state calculation starts with the identification of the valid set of DAEs i.e., $\mathbf{f}^{(k)(i)}$. Then these equations are solved for the algebraic variables and the time derivatives of the differential variables taking the differential variables as known. It implies that (4) is solved for $\dot{\mathbf{x}}^{(k)(i)}$ and $\mathbf{y}^{(k)(i)}$. At the end of the last sub-phase $\mathbf{f}^{(k)(j_k)} \equiv \mathbf{f}^{(k+1)}$ is calculated.

3.2 Integration of the continuous and the discrete model

In the χ language, the discrete and continuous models are integrated by specifying the evolution of the continuous state relative to the discrete one. The beginning and the end of the continuous phase have already been specified. What is left, is to define the moment when the initial state is calculated.

A possible choice (made by several other hybrid language developers, e.g. [Barton, 1992]) could be to re-calculate the state immediately after it becomes inconsistent. In the χ language, however, immediate state re-calculation is not adopted for the following reasons.

First, it would require the modeller to specify by means of special language elements that a discontinuity occurs that requires the state to be recalculated. If such language constructs are not used, state consistency should be checked after each assignment statement which is very inefficient.

Second, immediate state re-calculation may lead to undesirable intermediate states if related discontinuities occur in different processes at the same time. Consider, for example, a piping system where two valves are used to select an alternative routing. The two valves must be either both on or both off. This is controlled by a control unit as depicted in Figure 2. An intermediate state, where one valve is on and the other one is off may not be desirable to be established.

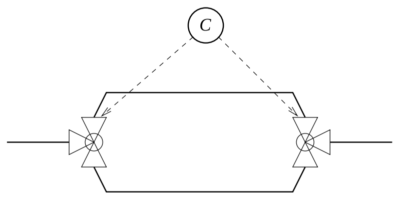

Figure 2: Pipe system with alternative routes

The third problem with immediate state re-calculation arises because processes are executed concurrently. While one process may trigger an initial state calculation, other processes can refer to their connected variables that are subject to change as a result of the initial state calculation. This has to be prevented, or else a scheduling dependency is introduced in the outcome of the simulation.

To prevent the above mentioned problems, the initial state is calculated when all processes are blocked, waiting for events to happen. As a result of the initial state calculation, a state-event condition which was false in the old state may become true in the new state in any of the processes. In this case a new discrete sub-phase resumes at the very same time point, followed again by a new initial state calculation.

Between two initial state calculations, the state may become inconsistent. This, however does not lead to modelling problems. Since all variables of the processes are local, the processes work with an independent local copy of the last consistent state. In this way, the behaviour of the processes depends only on the local variables (and data that is sent explicitly via discrete channels), which makes reasoning about them much easier.

3.3 Phase length

A discrete phase begins, when one or more processes become active. The end of a discrete phase (and the number of the sub-phases) is determined implicitly: a discrete phase ends when a consistent initial state is established in which all processes are blocked.

The length of the continuous-phase is limited by the smallest time-out. The continuous phase ends before that if *a)* the valid set of the DAEs is changed or *b)* a state-event condition becomes true in one of the processes.

In practice, the length of the continuous phase that can be simulated is lower bounded by the minimum stepsize of the DAE solver. Therefore, if at the beginning of a new continuous phase the minimum of the pending time-outs is found to be closer to the current time point than the minimum stepsize, the next discrete phase is executed at the current time point (without changing the current time). This situation often occurs in big models, where events of different processes nearly coincide. Note, that in this case the simple solution of advancing the time is not feasible, because this may lead to numerical inconsistency.

4 THE χ SIMULATOR

This section treats the algorithm implemented by the χ simulator. Here only the hybrid behaviour is detailed. Process scheduling within one sub-phase is described in [Naumoski and Alberts, 1998].

The algorithm is as follows.

```
$\tau := 0$;
*Initialize;*
*InitialStateCalc;*
*CheckOnActive;* ①
**while** *(actives $\neq \emptyset$* and $\tau \leq$ *endtime)* {
    **while** *actives* $\neq \emptyset$ {
        *Execute;* ②
        *InitialStateCalc;* ③
        *CheckOnNabla;* ④
    }
    *nextpoint* := *min*(*MinDelta*, *endtime*); ⑤
    **if** *(nextpoint* $<$ $\tau$+*solverminstep)* {
        *CheckOnDelta(nextpoint);*
        **if** (*actives* $= \emptyset$) { $\tau$ := *endtime*; }
    }
    **else if** *Hybrid*{*Solve*(*nextpoint*); ⑥ }
    **else** {
        $\tau$ := *nextpoint*;
        *CheckOnDelta(nextpoint);*
    }
}
```

The algorithm uses two sets: *actives* which contains the active processes, and *inactives* which contains the inactive processes. The current simulation time is denoted by τ. The following operations are used by the algorithm.

Initialize – executes the initialization block of each process:
foreach *p in inactives* { *ExecuteInit(p);* }

InitialStateCalc calculates the initial state.

CheckOnActive determines the active processes and moves them from *inactives* to *actives.*

CheckOnDelta(t) moves the processes that are blocked at a delta statement with a time-out equal to t from *inactives* to *actives.*

CheckOnNabla checks the conditions of blocking nabla statements. If any is found to be true, the process it belongs to is moved from *inactives* to *actives.*

Execute executes the non-blocking statements of the processes in *actives.* During this operation processes can be moved from *actives* to *inactives* and vice-versa. When *Execute* is finished, all processes are blocked waiting for an event.

MinDelta returns the minimum of the time-outs.

endtime simulation endtime.

solverminstep minimum stepsize of the DAE solver.

Hybrid returns true if the model is hybrid i.e., not pure discrete.

Solve(t) solves the DAEs until t. It monitors the blocking nabla conditions and updates τ. If any found to be true, solving stops and the process it belongs to is moved into *actives.* It also monitors the conditions of the active conditional equations. When any becomes false, solving stops and {*InitialStateCalc*; *CheckOnNabla*;} is executed.

In the next section, operation of the algorithm is explained by referring to the breakpoints ① ... ⑥.

5 EXAMPLE

To illustrate how the χ simulator works, consider the following control system.

```
proc $Tank(h\_ : \multimap [\mathrm{m}]$
            $, c : ?\,\mathrm{int},\ s : ?\,\mathrm{real}, k, A : \mathrm{real}) =$
[[ $V :: [\mathrm{m}^3],\ h :: [\mathrm{m}],\ Q_o :: [\mathrm{m}^3/\mathrm{s}],\ V_{add} : \mathrm{real},\ n : \mathrm{int}$
; $V ::= 10; n := 0$
| $h\_ \multimap h$
| $V' = -n \cdot Q_o,\ V = A \cdot h,\ Q_o = k \cdot \sqrt{h}$
| *[ $c\,?\,n$          $\longrightarrow$ skip
  [] $s\,?\,V_{add}$    $\longrightarrow V ::= V + V_{add}$
  ]
]]

proc $Supply(s : !\,\mathrm{real})=$
[[ *[ ① $\Delta 10$; $s\,!\,30$ ]   ]]
```

```
proc Control(h_ : ⊸ [m]
                , c : !real, h_max , h_min : real) =
⟦ h :: [m]
| h_ ⊸ h
| *[ ① ∇h > h_max ; c!1; ② ∇h < h_min  ; c!0 ]
⟧
```

Process *Tank* represents a tank filled with some substance. Variables V and h denote the volume and the height of the substance respectively. The tank has an outlet with a valve. The outgoing flow is denoted by Q_o. The substance is entered by process *Supply* in discrete doses, represented by a discrete communication over channel s. Process *Control* controls the level of the substance in the tank. If it exceeds a maximum level h_{max} the outlet is opened, if it drops under a minimum level h_{min} the outlet is closed.

The system is initialized with the following parameters:

```
syst S =
⟦ h : ⊸ [m], c : !? int, s : !? real
| Tank(h, c, s, 1, 2.5) ‖ Supply(s)
‖ Control(h, c, 15, 4)
⟧
```

The model is executed with *endtime* = 100. The first part of the trace is shown in Table 1. The values of the variables are printed in the order of $Tank.V, Tank.V'\ \ Tank.h, Tank.Q_o, Control.h$ $Tank.n$.

break-point	variables	*actives*	τ
①	$10, 0\ \ 4, 2, 4\ \ 0$	$\emptyset$	0
⑤	$nextpoint = 10$	$\emptyset$	0
⑥	$10, 0\ \ 4, 2, 4\ \ 0$	*Supply* ①	10
②	$40, 0\ \ 4, 2, 4\ \ 0$	$\emptyset$	10
③	$40, 0\ \ 16, 4, 16\ \ 0$	$\emptyset$	10
④	$40, 0\ \ 16, 4, 16\ \ 0$	*Control* ①	10
②	$40, 0\ \ 16, 4, 16\ \ 1$	$\emptyset$	10
③	$40, -4\ \ 16, 4, 16\ \ 1$	$\emptyset$	10
④	$40, -4\ \ 16, 4, 16\ \ 1$	$\emptyset$	10
⑤	$nextpoint = 20$	$\emptyset$	10
⑥	$10, -4\ \ 4, 4, 4\ \ 1$	*Control* ②	17.5

Table 1: Example trace

At time point 10 there is a discrete event with two sub-phases. The supplier adds 30 m^3 substance (fourth row). The increase in the height of the tank is detected by the control process after the new initial state has been calculated (sixth row).

6 CONCLUSIONS

In the χ language initial state calculation takes place when all processes are blocked. Therefore, no additional language constructs are needed to indicate when initial state calculation is required, which keeps the language small and easy to use. Furthermore, continuous variables that are being changed - either by means of assignment statements, or as a result of the initial state calculation - cannot be used in other processes because: *a)* variables are local to processes and *b)* when the initial state calculation actually changes the variables, all processes are blocked.

References

[Barton, 1992] Barton, P. (1992). *The Modelling and Simulation of Combined Discrete/Continuous Processes.* PhD thesis, University of London.

[Barton and Pantelides, 1994] Barton, P. and Pantelides, C. (1994). Modeling of combined discrete/continuous processes. *AIChE*, 40(6):966–979.

[Hoare, 1985] Hoare, C. (1985). *Communicating Sequential Processes.* Prentice-Hall, Englewood-Cliffs.

[Naumoski and Alberts, 1998] Naumoski, G. and Alberts, W. (1998). *A Discrete-Event Simulator for Systems Engineering.* PhD thesis, Eindhoven University of Technology. The Netherlands.

Georgina Fábián has received her M.Sc. degree in Computer and Information Science in 1994 from Eötvös Loránd University of Science, Budapest, Hungary. In the period 1994-1996 she completed the post-masters programme Software Technology at Eindhoven University of Technology, The Netherlands. Currently she is working towards her Ph.D. degree at the Faculty of Mechanical Engineering, Eindhoven University of Technology, The Netherlands. Her thesis concerns hybrid language and simulator development. Her research interest includes hybrid languages, distributed systems and real-time embedded systems.