In *Modelling and Simulation 1996—Proceedings of the 1996 European Simulation Multiconference*, Budapest, pp. 219–224.

SPECIFICATION OF COMBINED CONTINUOUS-TIME / DISCRETE-EVENT MODELS

D.A. van Beek, J.E. Rooda, and M. van den Muyzenberg
Eindhoven University of Technology
Department of Mechanical Engineering
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
E-mail: vanbeek@wtb.tue.nl

ABSTRACT

Although there are many different modelling and simulation languages, only few are suited to analysis of hybrid systems. In this paper a classification of different kinds of models and modelling languages is given. It is made clear that many hybrid languages are basically either continuous-time or discrete-event languages to which discrete-event or continuous-time constructs have been added. The χ formalism has been designed with the purpose of integrating continuous-time and discrete-event concepts. In this way it is suited to specification and simulation of discrete-event, continuous-time and hybrid systems. An informal treatment of its syntax and semantics is presented, together with a case study of a barrel filling system. The barrels are brought in on a conveyor which is modelled as a discrete-event system. When a barrel is under the filling injector, the barrel is filled from a filling reservoir. This reservoir is modelled as a continuous process, but it is controlled by a discrete-event process. The pumps are modelled by hybrid processes that make use of conditional equations. In this way, the example clarifies the hybrid nature of the χ language.

INTRODUCTION

Currently, a big gap exists between languages for the specification and simulation of continuous-time and discrete-event industrial systems. Continuous systems, such as chemical reactors or distillation columns, can be modelled using a continuous language. Discrete systems, such as transportation and packing of discrete products, can be modelled using a discrete language. However, many industrial processes exhibit both continuous and discrete characteristics. Examples of such processes are batch or fed-batch processes, supervisory control systems which interact with local controllers, and discrete-event controllers that control continuous-time processes.

For the modelling of such systems, languages are needed which integrate the concepts required for continuous-time and discrete-event modelling. This will enable the analysis of combined systems instead of concentrating on smaller subsystems of either a continuous or discrete nature. The χ formalism, presented in this paper, has been designed with the objective of integrating continuous-time and discrete-event concepts. The language is equally well suited to modelling and simulation of continuous-time, discrete-time or discrete-event systems as well as to modelling and simulation of combined systems.

DISCRETE AND CONTINUOUS MODELS

In this section a classification of different kind of models is given. We distinguish continuous-time, discrete-event, discrete-time and hybrid models.

Continuous-time models are characterized by the fact that in any finite time interval an infinite number of state changes occurs. The execution of the model leads to continuously changing variables. In the time domain these models are specified by a set of equations which can either be ordinary differential equations (ODEs) or differential algebraic equations (DAEs) such as:

$$\begin{aligned} Ah' &= f \\ f &= k\sqrt{h} \end{aligned}$$

In discrete-event models the state changes only at discrete points of time; in between two adjacent discrete points of time the state remains the same. Execution of the model leads to a sequence of events. There are many different ways of specifying these models in the time domain. Some examples are: finite state machines, Petri nets, and communicating sequential processes.

In discrete-time models the state changes only at (equidistant) points of time kT. These models can be regarded as either discrete-event models or as continuous-time models that are observed at points of time kT.

In the literature, hybrid models are defined as either a combination of continuous-time and discrete-time models, or as a combination of continuous-time and discrete-event models. In this paper the latter, more general, definition is used.

MODELLING LANGUAGES

Currently, there are hundreds of continuous-time or discrete-event modelling languages. Many continuous-time languages are based on the old CSSL standard (Augustin et al. 1967). Different attempts have been made to develop languages suited to modelling combined continuous-time / discrete-event models by taking as a basis a continuous-time language. To such a language discrete-event constructs using time-events or state-events have been added. Examples of these languages

are ACSL (Mitchell and Gauthier 1976) and Dymola (Elmqvist 1994). They are suited to continuous-time modelling with discontinuity handling. An example of a such a discontinuity is the behaviour of a vessel which flows over. The languages are, however, not suited to modelling of discrete-event systems.

The language gPROMS, which is described in (Barton 1992), is based on the continuous-language Speedup (Aspen Technology 1991). Its discrete-event part is based on composition of sequential and parallel blocks, which makes it suited to modelling hybrid systems with simple discrete-event components. The most important difference between this language and the χ language is found in their discrete-event parts. The discrete-event part of χ is based on concurrent programming, offering communication, synchronization and selective waiting. Such constructs are essential for the specification of (hybrid) processes with concurrently executing discrete-event components.

Examples of discrete-event languages are languages based on Petri nets (see (David and Alla 1994) for a survey paper) and many others (see (Hlupic 1995) for a comparative paper). Some of these languages, such as SIMAN (Pegden et al. 1995), have been extended with assignment based constructs for representing equations. Using such languages, the user is forced to concentrate a great deal on equation *solving* instead of simply *specifying* the equations describing the continuous-time systems. In the case of systems containing algebraic loops, it is the responsibility of the user to break the loops. The user must also specify the 'equations' in the right order. Therefore, these languages are only suited to the modelling of hybrid systems with simple equations, such as the ones specified in the example in this paper.

The main purpose of the modelling languages described above is the time-dependent analysis of dynamic systems by means of simulation. Hybrid systems can also be specified by means of finite automata (e.g. see (Alur et al. 1995)), where each discrete state can be associated with a set of equations. The main purpose of many such formalisms is algorithmic verification of hybrid systems. Research in this direction is also taking place using the χ language (Fey 1996), but the main purpose of this language is the design and analysis of industrial systems using specification and simulation.

THE χ LANGUAGE

The language is based on a small number of orthogonal language constructs which makes it easy to use and to learn. Where possible, the continuous-time and discrete-event parts of the language are based on similar concepts. Parametrized processes and systems provide modularity, parallelism with communication and synchronization, and hierarchical modelling. The language is based on mathematical concepts with well defined semantics. Its symbolic notation makes the specifications easy to read and to develop. Compare the symbolic notation of

$$h' = k\sqrt{p_i - p_o} \mid \Delta t - \tau$$

with its ASCII equivalent for simulation purposes

```
h' = k * sqrt(p_i - p_o) | delta t - time.
```

In this paper, only the language constructs used in the example are treated. The syntax and operational semantics of the language constructs are explained in an informal way.

The model of a system consists of a number of process (or system) instantiations, and channels connecting these processes. Systems are parametrized:

syst *name*(*parameter declarations*) =
⟦ *channel declarations*
| *process or system instantiations*
⟧

The parameter declaration of processes is identical to that of systems. A process may consist of a continuous-time part only (links and DAEs), a discrete-event part only, or a combination of both.

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

The continuous-time and discrete-event parts of the language are based on similar concepts. Processes have local variables only; all interactions between processes take place by means of channels. A channel connects two processes or systems. A (discrete) synchronization channel is symmetrical; all other channels are used for output in one process and input in the other. Channels are declared in systems and are either discrete (e.g. p : int, s : void) or continuous (e.g. q : [m/s]). Channels are also declared as process or system parameters in which case the usage of the channel is declared as either output (e.g. p : !int or q : ↑[m/s]), input (e.g. p : ?int or q : ↓[m/s]), or synchronization (s : ~void). A continuous channel is represented graphically by a line ending in a small circle, a discrete communication channel by an arrow, a discrete synchronization channel by a line; processes and systems are represented by circles.

Data types and variables

All data types and variables are declared as either continuous or discrete. This is an important distinction with other hybrid modelling languages in which the type of a variable is often implicitly inferred from its use.

The value of a discrete variable is determined by assignments. Between two subsequent assignments the variable retains its value. The value of a continuous variable, on the other hand, is determined by equations. An assignment to a continuous variable (initialization) determines its value for the current point of time only.

Some discrete data types are predefined like bool (boolean), int (integer) and real. A real discrete data type can be postfixed with a unit of measurement. For example, the declaration v : real[m/s] defines a discrete variable (or

parameter) v of type real with unit m/s (a velocity). Since all continuous variables are assumed to be of type real, they are defined by specifying their units only. For example, the declaration v : [m/s] defines a continuous variable or channel v. Continuous channels are specified likewise in parameter declarations, e.g. $v : \uparrow$ [m/s] defines a continuous channel v which may be linked to a continuous variable of type [m/s].

The continuous-time part of χ

The continuous-time part of χ (see (Arends et al. 1994) for a preliminary description) is based on Differential Algebraic Equations (DAEs). An informal explanation of the constructs follows below.

DAEs are separated by commas:

$$DAE_1, DAE_2, \ldots, DAE_n.$$

A time derivative is denoted by a prime character (e.g. x'). If the set of DAEs depends on the state of a system, *guarded DAEs* can be used:

$$[\, b_1 \longrightarrow DAEs_1 \,[]\, \ldots \,[]\, b_n \longrightarrow DAEs_n \,].$$

The boolean expression b_i ($1 \leq i \leq n$) denotes a *guard*, which is open if b_i evaluates to true and is otherwise closed. At any time at least one of the guards must be open so that the *DAEs* associated with an open guard can be selected.

A continuous channel relates a variable of one process to a variable of another process by means of an equality relation. *Links* are used to associate a channel with a variable:

$$channel \multimap var.$$

The variable and the channel must be of the same (continuous) data type. Consider two variables x_a, x_b in different processes linked to a continuous channel c ($c \multimap x_a$ and $c \multimap x_b$). The channel and links cause the equation $x_a = x_b$ to be added to the set of DAEs of the system.

The discrete-event part of χ

The discrete-event part of χ is a CSP-like (Hoare 1985) real-time concurrent programming language, described in (Mortel-Fronczak and Rooda 1995) and (Mortel-Fronczak et al. 1995). Our notation is derived from (Hooman 1991). An informal explanation of the constructs follows below.

Interaction between discrete-event parts of processes takes place by means of synchronous message passing or by synchronization only:

$$c!e, \quad c?x \quad \text{or} \quad c\tilde{}\,.$$

Consider the channel c connecting two processes. Execution of $c!e$ in one process causes the process to be blocked until $c?x$ is executed in the other process, and vice versa. Subsequently the value of expression e is assigned to variable x. Synchronization is denoted by $c\tilde{}$. Execution of $c\tilde{}$ in one process causes the process to be blocked until $c\tilde{}$ is executed in the other process.

Time passing is denoted by

$$\Delta\, t,$$

where t is a real expression. A process executing this statement is blocked until the time is increased by t time-units. The state event statement $\nabla\, rel$ is explained in the following section.

Selection ($[GB]$) is denoted by

$$[\, b_1 \longrightarrow S_1 \,[]\, b_2 \longrightarrow S_2 \,[]\, \ldots \,[]\, b_n \longrightarrow S_n \,].$$

The boolean expression b_i ($1 \leq i \leq n$) denotes a *guard*, which is open if b_i evaluates to true and is otherwise closed. At least one of the guards must be open. After evaluation of the guards, one of the statements S_i associated with an open guard b_i is executed.

Selective waiting ($[GW]$) is denoted by

$$[\, b_1 ;\ E_1 \longrightarrow S_1 \,[]\, \ldots \,[]\, b_n ;\ E_n \longrightarrow S_n \,].$$

An event statement E_i which is prefixed by a guard b_i ($b_i ;\ E_1$) is enabled if the guard is open and the event specified in E_i can actually take place. Continuous variables are not allowed in these guards. There are five types of event statement: input ($c?e$), output ($c!x$), synchronization ($c\tilde{}$), time ($\Delta\, t$), and state ($\nabla\, rel$, explained in the following section). The process executing $[GW]$ remains blocked until at least one event statement is enabled. Then, one of these (E_i) is chosen for execution, followed by execution of the corresponding S_i. Time-outs are event statements of the form $\Delta\, t$ that are prefixed by a guard ($b ;\ \Delta\, t$). A process cannot be blocked in a selective waiting statement with time-outs for longer than the smallest time-out time t_s (provided the associated time-out guard is open). If no other event statements are enabled within that period, a statement S associated with an enabled time-out of time t_s is executed. Please note that guards that are always true may be omitted together with the succeeding semicolon. Therefore true; E may be abbreviated to E.

Repetition of the statements $[GB]$ and $[GW]$ is denoted by

$$*[GB] \text{ and } *[GW],$$

respectively. The repetition terminates when all guards are closed. The repetition $*[$ true $\longrightarrow S\,]$ may be abbreviated to $*[\, S\,]$.

Interaction between the continuous and discrete parts of χ

In the discrete-event part of a process, assignments can be made to discrete variables occurring in DAEs or in the boolean guards of guarded DAEs. In the first case the DAEs will be evaluated with new values, in the latter case different DAEs may be selected. Continuous variables are initialized immediately after the declarations, and may be reinitialized in the discrete-event part. In both cases the symbol ::= (e.g. $x ::= 0$) is used.

By means of the *state event* statement

$$\nabla\, rel,$$

the discrete-event part of a process can synchronize with the continuous part of a process. Execution of ∇ *rel*, where *rel* is a relation involving at least one continuous variable, causes the process to be blocked until the relation becomes true.

AN EXAMPLE

This example treats the specification of an industrial barrel filling system using the χ language. The barrel filling system consists of two product tanks which feed a stirred reservoir. Liquid in this reservoir is used to fill barrels which are placed on a conveyor belt (see Figure 1). The liquids are transported by means of positive-displacement pumps.

The specification is best understood in combination with Figure 2 which shows the processes and channels.

Liquid Transport

The following types are used in the definitions of the processes and the system:

```
type flow   = [m^3 · s^-1]
,    press  = [N · m^-2]
,    height = [m]
,    barrel = real[m^3] -- contents of barrel
```

The following constants are used:

```
const g   = 9.8 [m · s^-2]
,     ρ   = 1 · 10^3 [kg · m^-3]
,     μ   = 1 · 10^-3 [N · s · m^-2]
```

The specification of filing reservoir *FR* follows below. This process uses one DAE for the height of the fluid in the reservoir.

```
proc FR( q2, q4 : ↓ flow, q5 : ↑ flow, h1 : ↑ height
       , h0, A : real
       ) =
[[ qi1, qi2, qo : flow, h : height
; h ::= h0
| q2 -o qi1
, q4 -o qi2
, q5 -o qo
, h1 -o h
| Ah' = qi1 + qi2 - qo
]]
```

Initially, flow q through a pump P is set to zero by assigning false to boolean *run*. When switched on, pump P causes a flow with flow-rate q_{set}. The process then continuously tries to receive a new setpoint from reservoir-controller *RC* via channel c.

```
proc P( qin : ↓ flow, qout : ↑ flow
      , c : ? bool
      , qset : real
      ) =
[[ q : flow, run : bool
; run := false
| qin  -o q
, qout -o q
| [   run --> q = qset
  [] ¬run --> q = 0
  ]
| *[ c?run ]
]]
```

Product tanks *PT1* and *PT2* are modelled as infinite capacity tanks:

```
proc PT(qx : ↑ flow) =
[[ q : flow
| qx -o q
]]
```

Barrel Transport

Barrel transport takes place by means of synchronous communication. Barrel supplier *BS* continuously tries to send empty barrels to the filler *BF*. Forty barrels are sent in total.

```
proc BS(a : ! barrel) =
[[ n : int
| n := 40
; *[ n > 0 --> a!0; n := n - 1 ]
]]
```

The continuous part of the specification of barrel-filling process *BF* contains one DAE for the height of the barrel which is currently being filled. In the discrete part, first a barrel is received from *BS* and continuous variable h is reinitialized. Then, a synchronization with the controller *RC* takes place via f_{start} in order to start filling of a barrel. When it has been filled, the controller synchronizes via f_{done} to notify *BF* that the barrel can be sent to *BP*. After t seconds the next barrel is received from *BS*.

```
proc BF( q6 : ↓ flow, h2 : ↑ height
       , a : ? barrel, b : ! barrel
       , fstart, fdone : ~ void
       , A, t : real
       ) =
[[ q : flow, h : height
, x : barrel
; h ::= 0
| q6 -o q
, h2 -o h
| Ah' = q
| *[ a?x; h ::= x; fstart~; fdone~; b!Ah; Δ t ]
]]
```

Barrels are stored in the pile *BP*. For reasons of simplification the actual storing is not modelled.

```
proc BP(b : ? barrel) =
[[ x : barrel
| *[ b?x ]
]]
```

The Controller

Reservoir controller *RC* controls the level of the liquid contained in *FR*. When this level drops below the lower boundary h_l, pumps *P1* and *P2* are switched on by sending true via channels c_1 and c_2. When the level has reached the upper boundary h_h, the pumps are switched off again.

Besides controlling the level of *FR*, *RC* also controls the filling process by switching pump *P3* off when the level in the barrel reaches the value h_{bs}, so that the barrel is filled.

Therefore, during filling of a barrel, two conditions need to be monitored: $h \leq h_l$ and $h_b \geq h_{bs}$. When the level of reservoir *FR* drops below h_l when a barrel is being filled, filling of the barrel is stopped and the reservoir is filled first. When the height of the liquid in *FR* has reached its maximum value h_h and it has been stirred for t_{stir} seconds, filling of the barrel is resumed. Boolean *fb* is true when a barrel is being filled.

```
proc RC( h1, h2 : ↓ height
       , c1, c2, c3 : ! bool, fstart, fdone : ~ void
       , hl, hh, hbs, tstir : real
       ) =
|[ hr, hb : height, fb : bool
| h1 −o h
, h2 −o hb
| fb := false
; *[ fstart ~ ; c3 ! true ; fb := true
   ; *[ fb; ∇ h ≤ hl   −→ c3!false ; c1!true ; c2!true
                        ; ∇ h ≥ hh
                        ; c1!false ; c2!false
                        ; Δ tstir ; c3!true
      [] fb; ∇ hb ≥ hbs −→ c3!false ; fdone ~
                        ; fb := false
      ]
   ]
]|
```

The System Definition

In system T, the processes specified previously are instantiated.

```
syst T =
|[ q1, q2, q3, q4, q5, q6 : flow, h1, h2 : height
, c1, c2, c3 : bool, a, b : barrel, fstart, fdone : ~ void
| RC(h1, h2, c1, c2, c3, fstart, fdone, 0.15, 1.35, 1, 20)
|| BF(q6, h2, a, b, fstart, fdone, .196, 8)
|| PT(q1) || PT(q3) || BS(a) || BP(b)
|| P(q1, q2, c1, 0.06)
|| P(q3, q4, c2, 0.03)
|| P(q5, q6, c3, 0.01)
|| FR(q2, q4, q5, h1, 0.75, 3.14)
]|
```

CONCLUDING REMARKS

We conclude by observing that the χ language is a high level language based on a small number of powerful concepts. It is suited to modelling and simulation of: (1) continuous-time processes occurring in reactors, tanks etc.; (2) discrete-event processes, such as transport and handling of individual products; and (3) combined continuous-time / discrete-event processes such as batch processes occurring in the process industry. In this way one formalism can be used for the specification and simulation of continuous-time, discrete-time, discrete-event, or hybrid processes.

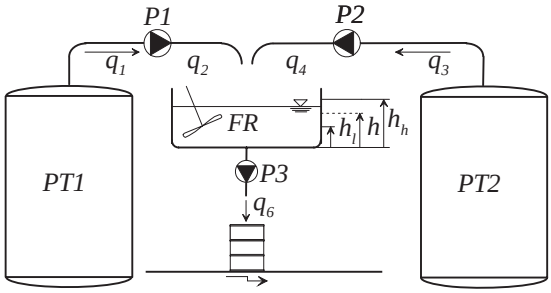

Figure 1: The Barrel Filling System.

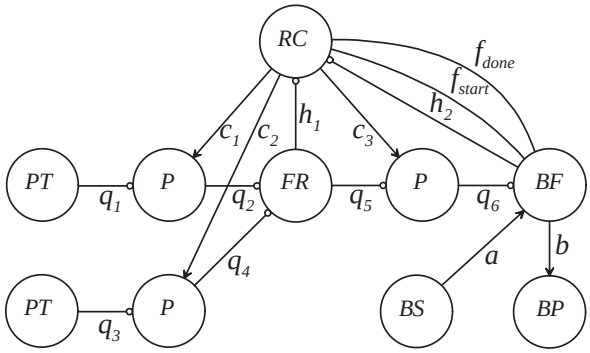

Figure 2: System T.

The hybrid nature of the language has been illustrated using a simple barrel filling system. A more complex example originating in the process industry is presented in (Beek et al. 1995). The application of the χ formalism to modelling and simulation of manufacturing machines is presented in (Beek et al. 1995).

A χ simulator for the discrete-event part of the language is already in operation (Naumoski and Alberts 1995). It is being used for modelling and simulation of complex discrete-event manufacturing systems such as production facilities for integrated circuits. A first prototype of the χ simulator for combined continuous-time / discrete-event systems is available (Kamp and Voorbraak 1995); the final version is still under construction.

REFERENCES

Alur, R.; C. Courcoubetis; N. Halbwachs; T.A. Henzinger; P.H. Ho; X. Nicollin; A. Olivero; J. Sifakis; and S. Yovine. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138*, 3–34.

Arends, N.W.A.; J.M. van de Mortel-Fronczak; and J.E. Rooda. 1994. Continuous Systems Specification Language. In *CISS - First Joint Conference of International Simulation Societies Proceedings*, Zurich, 76–79.

Aspen Technology 1991. *Speedup User Guide*. Cambridge: Aspen Technology.

Augustin, D.C.; M.S. Fineberg; B.B. Johnson; R.N. Linebarger; F.J. Sansom; and J.C. Strauss. 1967. The SCi Continuous System Simulation Language (CSSL). *Simulation 9*, 281–303.

Barton, P.I. 1992. *The Modelling and Simulation of Combined Discrete/Continuous Processes*. Ph. D. thesis, University of London.

Beek, D.A. van; S.H.F. Gordijn; and J.E. Rooda. 1995. Integrating Continuous-Time and Discrete-Event Concepts in Process Modelling, Simulation and Control. In *Proceedings of the First World Conference on Integrated Design and Process Technology*, 197–204.

Beek, D.A. van; J.E. Rooda; and S.H.F. Gordijn. 1995. A Combined Continuous-Time / Discrete-Event Approach to Modelling and Simulation of Manufacturing Machines. In *Proceedings of the 1995 EUROSIM Conference*, Vienna, 1029–1034.

David, R. and H. Alla. 1994. Petri Nets for Modeling of Dynamic Systems–A Survey. *Automatica 30*(2), 175–202.

Elmqvist, H. 1994. *Dymola–Dynamic Modeling Language–User's Manual*. Lund, Sweden: Dynasim AB.

Fey, J.J.H. 1996. Control and Verification of Hybrid Systems Using Models Specified with the Formalism χ. Research report WPA 420090, Eindhoven University of Technology, The Netherlands.

Hlupic, V. 1995. A Comparison of Simulation Software Packages. In *Proceedings of the 1995 EUROSIM Conference*, Vienna, 171–175.

Hoare, C.A.R. 1985. *Communicating Sequential Processes*. Englewood-Cliffs: Prentice-Hall.

Hooman, J. 1991. *Specification and Compositional Verification of Real-Time Systems*. Springer-Verlag.

Kamp, G.F.J.W. van de and E.M.M. Voorbraak. 1995. Control of Hybrid Industrial Systems. Final report of the Postgraduate Programme Computational Mechanics, Stan Ackermans Institute, Eindhoven University of Technology, The Netherlands.

Mitchell, E.E.L. and J.S. Gauthier. 1976. Advanced Continuous Simulation Language (ACSL). *Simulation 26*(3), 72–78.

Mortel-Fronczak, J.M. van de and J.E. Rooda. 1995. Application of Concurrent Programming to Specification of Industrial Systems. In *Proceedings of the 1995 IFAC Symposium on Information Control Problems in Manufacturing*, Bejing, 421–426.

Mortel-Fronczak, J.M. van de; J.E. Rooda; and N.J.M van den Nieuwelaar. 1995. Specification of a Flexible Manufacturing System Using Concurrent Programming. *Concurrent Engineering: Research and Applications 3*(3), 187–194.

Naumoski, G. and W.T.M Alberts. 1995. The χ Engine: a Fast Simulator for Systems Engineering. Final report of the Postgraduate Programme Software Technology, Stan Ackermans Institute, Eindhoven University of Technology, The Netherlands.

Pegden, C.D.; R.E. Shannon; and R.P. Sadowski. 1995. *Introduction to Simulation Using SIMAN*. McGraw-Hill.