In *Proceedings of the First World Conference on Integrated Design and Process Technology*, Austin Texas, Dec. 1995, pp. 197–204.

INTEGRATING CONTINUOUS-TIME AND DISCRETE-EVENT CONCEPTS IN PROCESS MODELLING, SIMULATION AND CONTROL

D.A. van Beek **S.H.F. Gordijn** **J.E. Rooda**
Eindhoven University of Technology
Department of Mechanical Engineering
P.O. Box 513, 5600 MB Eindhoven
The Netherlands

ABSTRACT

Currently, modelling of systems in the process industry requires the use of different specification languages for the specification of the discrete-event and continuous-time subsystems. In this way, models are restricted to individual subsystems of either a continuous-time or discrete-event nature. It is our aim to integrate such models, by using one language for the specification of complete plants or production units. For this purpose, we introduce the language χ in this paper. This language integrates a small number of orthogonal continuous-time and discrete-event concepts. The continuous-time part of χ is based on DAEs; the discrete-event part is based on a CSP-like concurrent programming language. Models are specified in a symbolic mathematical notation. A case study is presented of a plant for the biochemical production of ethanol. The production takes place in a number of fermentors in a fed-batch fashion. The fermentation process and the control system, which controls the various valves and pumps and schedules the different batches, are both specified in χ. The example illustrates the relevance of integrating continuous-time and discrete-event concepts.

NOMENCLATURE

c_d	=	non-viable cells mass concentration
c_p	=	ethanol mass concentration
c_s	=	substrate mass concentration
c_{si}	=	incoming substrate mass concentration
c_v	=	viable cells mass concentration
f_i	=	ingoing flow
f_o	=	outgoing flow
K_s	=	saturation constant
m_s	=	maintenance constant
R_d	=	cell decay rate
V	=	volume
Y_{ps}	=	yield of product on substrate
Y_{xs}	=	yield of growth on substrate
α, β	=	metabolic constants
μ	=	specific growth rate
μ_{max}	=	maximal specific growth rate

INTRODUCTION

Although systems in the process industry are often modelled using continuous-time languages such as Speedup (Aspen, 1991) or ACSL (Mitchell and Gauthier, 1976), in many cases discrete-event concepts are also needed. Discrete-event concepts are required among others when systems operate in batch or fed-batch mode, in the start-up or shutdown phase of continuous processes, in supervisory control systems, and in the case of erroneous process conditions requiring safety actions from the control system.

Current common practice in industrial systems modelling is that different specification languages are used for the specification of the discrete-event and continuous-time subsystems. In this way, the different models cannot be integrated. The modelling language χ, 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 the modelling of continuous-time, discrete-time or discrete-event systems as well as for combined systems.

Another language for modelling combined systems is gPROMS, which is described in (Barton, 1992). The most important difference between the two languages is found in their discrete-event parts. The discrete-event part of χ is based on concurrent programming, offering communication, synchronization and selective waiting; whereas the discrete-event part of gPROMS is based on composition of sequential and parallel blocks.

There is also a considerable difference between languages used for control and simulation languages. The language χ aims to reduce the gap between simulation and control of industrial systems. It can be used for specification and simulation of industrial systems in the time domain, but it is also suited to modelling of discrete-event sequence control systems and discrete-time or continuous-time control systems such as PID control.

The remainder of this paper is organized as follows. First, the syntax and semantics of the language χ is introduced. Second, the model of a simple plant for the production of ethanol is treated as an example. Both the physical processes and the control system are modelled in χ. Finally, some concluding remarks are given.

THE LANGUAGE χ

The language consists of a small number of orthogonal constructs. In this paper, only those constructs are treated that are used in the example. 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 decls*) $=$
$[\![$ *channel decls* | *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 decls*) $=$
$[\![$ *variable decls* | *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, q : $\sim$) or continuous (e.g. p : [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. $p : \uparrow$ [m/s] or p : ! int), input (e.g. $p : \downarrow$ [m/s] or p : ? int), or synchronization (p : $\sim$). 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

Some data types are predefined like bool (boolean), int (integer) and real. A 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 units m/s (a velocity). There are also continuous variables, the value of which is determined by DAEs or by assignments. Since all continuous data types 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].

Structured types are defined by means of tuple types. A tuple type $(T_1 \times T_2 \cdots \times T_n)$, where $T_1 \cdots T_n$ are predefined types, defines tuples $(t_1, t_2, \ldots, t_n)$, where t_i is an expression of type T_i $(1 \le i \le n)$.

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). A summary of the syntax in BNF notation is given in Table 1. In this table the four symbols ::= ‘ ’ | are BNF symbols. All identifiers are non-terminals, but the lower case non-terminals are not further defined; the non-terminal *eqx* denotes an equation expression. The symbols that are enclosed in quotes ‘’ are terminals (symbols of the language χ). An informal explanation of the constructs follows below.

The DAEs (*EQD*) are declared as a set.

$$\{EQ_1, EQ_2, \ldots, EQ_n\}.$$

A time derivative is denoted by a prime character (e.g. x'). If the form of a DAE depends on the state of a system, a *guarded DAE* ([*GE*]) can be used. The non-terminal *EQ* represents one or more DAEs separated by commas.

$$[\, b_1 \longrightarrow EQ_1 \mid b_2 \longrightarrow EQ_2 \mid \ \ldots \mid b_n \longrightarrow EQ_n \,].$$

The boolean expressions b_i $(1 \le i \le n)$ are called *guards*. A guard which evaluates to true is called open. When a guarded DAE is evaluated, at least one of the guards must be open. Then one EQ_i $(1 \le i \le n)$ which is associated with an open guard is 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 variable with a channel:

$$var \multimap channel.$$

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 ($x_a \multimap c$ and $x_b \multimap c$). The channel and links cause the equation $x_a = x_b$ to be added to the set of DAEs of the system.

It is also possible to link different variables to one channel. This is done by means of tuples. Consider the declaration of a channel $c : (T_1 \times T_2 \cdots \times T_n)$ in a system, a link declaration $(x_{1a}, x_{2a}, \ldots, x_{na}) \multimap c$ in a process, and a link declaration $(x_{1b}, x_{2b}, \ldots, x_{nb}) \multimap c$ in another process. If the identifier c refers to the same channel in all three declarations, then the result of linking the variables is that the equations $x_{1a} = x_{1b}$, $x_{2a} = x_{2b}$, $\ldots$, $x_{na} = x_{nb}$ are 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). Our notation is derived from (Hooman, 1991). A summary of the syntax in BNF notation is given in Table 2. 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:

$$c!e \qquad c?x \qquad c^{\sim} .$$

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 $c^{\sim}$. Execution of $c^{\sim}$ in one process causes the process to be blocked until $c^{\sim}$ 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 $\nabla\, bc$ is explained in the following section.

Non-deterministic 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. If all guards are closed, the statement terminates after evaluation of the guards. Otherwise, a statement S_i associated with a (nondeterministically chosen) open guard b_i is executed.

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

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

A *guard* of the form $b_i\, ;\; E_i$ is open if b_i evaluates to true; it is enabled if it is open and the event statement specified in E_i can actually take place. There are five types of event statement: input ($c?e$), output ($c!x$), synchronization ($c^{\sim}$), time ($\Delta\, t$), and state ($\nabla\, bc$, explained in the following section). If none of the guards is open, the statement terminates after evaluation of the boolean parts of the guards. Otherwise, the process executing $[GW]$ remains blocked until at least one guard is enabled. Then, one of the E_i statements belonging to an enabled guard is chosen for execution, followed by execution of the corresponding S_i. Time-outs are guards of the form $b;\, \Delta\, t$. 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 guards are enabled within that period, a statement S associated with an open time-out of time t_s is executed. Guards of the form 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 variables occurring in DAEs or to variables occurring in the boolean guards of guarded DAEs. In the first case the DAEs will be evaluated with new initial conditions, in the latter case different DAEs may be selected.

By means of the *state event statement*

$$\nabla\, bc,$$

the discrete-event part of a process can synchronize with the continuous part of a process. Execution of $\nabla\, bc$, where bc is a boolean expression involving at least one or more continuous variables, causes the process to be blocked until bc becomes true.

AN EXAMPLE

A considerable amount of research has been done in the field of modelling stand-alone biochemical processes. In many cases, however, the production of (bio)chemical products involves a number of different production steps in different reactors. Therefore, for the analysis of the complete production process, integrated models of plants are required instead of stand-alone models of subprocesses.

The example treats the specification of an industrial system for batch production of industrial ethanol. The ethanol is produced as a result of the biodegradation of glucose by yeast in a fed-batch mode. The specification can be simulated for the determination of the required capacities of reactors, product tanks etc.

The plant described here consists of two fermentors, two product tanks, a feed tank, a sterilizer, pumps and valves (see Figure 1). A real plant would be considerably more complex.

The Fermentor

The behaviour of the fermentor is described by a set of DAEs. The growth model used is that of Monod (Baily and Ollis, 1977). There are four different concentrations in the fermentor: c_s, c_p, c_d and c_v. These represent the concentrations of the substrate, product, non-viable cell and viable cell, respectively. The general form of the differential equations which describe these variables is given in Equation (1), where R_{net} is the net consume or decay rate of quantity x. The concentration of quantity x is denoted by c_x.

$$\frac{dVc_x}{dt} = f_i c_{xi} - f_o c_x - R_{net} V c_v \qquad (1)$$

The types that are used in the processes and systems are specified below. All times are expressed in hours.

```
type  flow  = $[\mathrm{m^3h^{-1}}]$
,     conc  = $[\mathrm{kgm^{-3}}]$
,     press = $[\mathrm{Nm^{-2}}]$
,     vol   = $[\mathrm{m^3}]$
,     fc    = (flow $\times$ conc)
,     fcp   = (flow $\times$ conc $\times$ press)
,     fc4p  = (flow $\times$ conc$^4$ $\times$ press)
```

The complete specification of the fermentor follows below. The specifications of the various processes are best understood in combination with Figure 2: 'System F'.

```
proc  *Fermentor*
($f_p : \downarrow$ fc, $f_f : \uparrow$ fc4p, $V_f : \uparrow$ vol, $c : \uparrow$ conc, $i : ^{\sim}$,
 $\alpha, \beta, K_s, R_d, Y_{ps}, Y_{xs}, m_s, \rho, g, A, p_\infty$,
 *mean*, *sd* : real
) =
$[\![$ $f_i, f_o$ : flow, $c_v, c_d, c_s, c_p, c_{si}$ : conc, $p$ : press
, $V$ : vol, $\mu_{dist} : \rightarrow$ , $\mu_{max}$ : real
| $(f_i, c_{si}) \multimap f_p$, $(f_o, c_v, c_d, c_s, c_p, p) \multimap f_f$
, $c_s \multimap c$, $V \multimap V_f$
| { $\mu$      $= \mu_{max} c_s / (K_s + c_s)$
  , $(Vc_v)' = -f_o c_v + (\mu - R_d) V c_v$
  , $(Vc_d)' = -f_o c_d + R_d V c_v$
  , $(Vc_s)' = f_i c_{si} - f_o c_s - (\frac{\mu}{Y_{xs}} + m_s + \frac{\alpha\mu+\beta}{Y_{ps}}) V c_v$
  , $(Vc_p)' = -f_o c_p + (\alpha\mu + \beta) V c_v$
  , $p$      $= p_\infty + \rho g V / A$
  , $V'$     $= f_i - f_o$
  }
| $\mu_{dist}$ := nor *mean sd*
; $V := 0$; $c_v := 0$; $c_d := 0$; $c_s := 0$; $c_p := 0$
; $*[$ $i^{\sim}$; $\mu_{max} := \sigma\ \mu_{dist}$
    ; $c_v := 0.001$; $c_d := 0.001$; $c_s := 100$; $c_p := 0.001$
    ; $V := 0.001$; $\nabla V = 0$
    ]
$]\!]$
```

In the real system, the fermentor receives an initial charge of cells from a seed tank. In the specification above, this operation is represented by a synchronization on port i, followed by a corresponding initialization of the volume and the concentrations. The value of μ_{max} is drawn from a distribution μ_{dist} ($\mu_{max} := \sigma\ \mu_{dist}$). This distribution is initialized with a normal distribution nor with mean value *mean* and standard deviation sd. In this way a stochastic effect is modelled.

When the batch is finished, the fermentor is emptied. This will cause the volume to become equal to 0, which is detected with the statement $\nabla V = 0$. Consequently the discrete-event cycle is restarted at the beginning.

Port c is used to pass the value of c_s onto the process *Sampler* which models a sampling station with sample time δ (approximately 15 minutes).

```
proc  *Sampler*$(c : \downarrow$ conc, $c_z : \uparrow$ conc, $\delta$ : real$)$ =
$[\![$ $x, x_z$ : conc
| $x \multimap c$, $x_z \multimap c_z$
| $*[$ $x_z := x$ ; $\Delta\delta$ ]
$]\!]$
```

The valves and pumps

When a batch is ready, the valve is opened in order to feed the contents of the fermentor into the product tank. The flow is a result of the overpressure in the fermentor. The flow is determined by the pressure difference over the valve: $\Delta p = \frac{1}{2}\zeta\rho f^2$. The parameter ζ represents the overall friction factor of the (open) valve and the piping. The state of the valve is determined by the boolean b. When b equals true the valve is open. When b equals false, the valve is closed and the flow is zero.

```
proc  *Valve*
$(f_f : \downarrow$ fc4p, $f_v : \uparrow$ fc4p, $a_v$ : ?bool, $\zeta, \rho$ : real$)$ =
$[\![$ $f$ : flow, $c_v, c_d, c_s, c_p$ : conc, $p_i, p_o$ : press, $b$ : bool
| $(f, c_v, c_d, c_s, c_p, p_i) \multimap f_f$, $(f, c_v, c_d, c_s, c_p, p_o) \multimap f_v$
| { [ $\neg b \longrightarrow f = 0$
    | $b \longrightarrow f = \sqrt{2(p_i - p_o)/\zeta\rho}$
    ]
  }
| $b$ := false
; $*[$ $a_v ? b$ ]
$]\!]$
```

The pumps are used to pump the substrate solution from the feed tank into the fermentor. The pump receives the value of the required flow via channel a_p from the control system.

```
proc  *Pump*$(f_e : \downarrow$ fc, $f_p \uparrow$ fc, $a_p$ : ?real$)$ =
$[\![$ $f$ : flow, $c$ : conc, $f_{set}$ : real
| $(f, c) \multimap f_e$, $(f, c) \multimap f_p$
| { $f = f_{set}$ }
| $*[$ $a_p ? f_{set}$ ]
$]\!]$
```

The specification of the product tank is straightforward, and is omitted here.

The system F

The components are combined in system F. The graphical representation is shown in Figure 2. The values of the metabolical constants have been taken from (Aspen, 1991).

```
syst $F\,(f_e : \downarrow \mathrm{fc},\ V_f : \uparrow \mathrm{vol},\ c_z : \uparrow \mathrm{conc},\ i : {}^{\sim},$
        $a_p : ?\,\mathrm{real},\ a_v : ?\,\mathrm{bool}$
      $) =$
$[\![\ f_p : \mathrm{fc},\ f_f,\ f_v : \mathrm{fc4p},\ c_z : \mathrm{conc}$
$|\ \mathit{Pump}(f_e, f_p, a_p)$
$\|\ \mathit{Fermentor}(f_p, f_f, V_f, c, i, 4.4, 0.03, 10, 0.01$
$,\ 0.51, 0.5, 0.1, 10^3, 9.8, 4, 1.5 \cdot 10^5, 1.05, 0.05)$
$\|\ \mathit{Valve}(f_f, f_v, a_v, 0.45, 10^3)$
$\|\ \mathit{ProductTank}(f_v, 10^5)$
$\|\ \mathit{Sampler}(c, c_z, 0.25)$
$]\!]$
```

The feed tank

The feed tank is used as a buffer between the sterilizer and the fermentors. In this example the substrate concentration coming from the sterilizer is constant. The equation on line (1), however, is also correct for varying concentrations.

```
proc $\mathit{FeedTank}(f_s : \downarrow \mathrm{fc},\ f_{e1}, f_{e2} : \uparrow \mathrm{fc},\ V_e : \uparrow \mathrm{vol}) =$
$[\![\ f_i, f_{o1}, f_{o2} : \mathrm{flow},\ c, c_i : \mathrm{conc},\ V : \mathrm{vol}$
$|\ (f_i, c_i) \multimap f_s,\ (f_{o1}, c) \multimap f_{e1},\ (f_{o2}, c) \multimap f_{e2}$
$,\ V \multimap V_e$
$|\ \{\ [\ V > 0 \longrightarrow (Vc)' = f_i c_i - (f_{o1} + f_{o2})c$          (1)
    $|\ V = 0 \longrightarrow c = c_i$
    $]$
  $,\ V' = f_i - (f_{o1} + f_{o2})$
  $\}$
$|\ V := 0$
$]\!]$
```

The sterilizer

The feed tank is filled by a continuous sterilizer. The raw medium (glucose solution) is first heated in a heat exchanger and then maintained at a constant temperature for a certain period of time (the time necessary to kill most microbial contaminants) in the holding section. The medium, which is now aseptic, is then cooled down.

The specification of the sterilizer is simplified to the level of a discrete-event process. When activated, the sterilizer delivers an aseptic glucose solution at a constant rate. The system has a characteristic delay δ between the moment of activation and the actual moment at which the first sterilized medium enters the feed tank. The sterilizer is activated when the process receives the value true via channel a_s, and is deactivated when it receives the value false. There is no delay when the process is switched off. The sterilizer can be deactivated at any time, also during the activation delay period. The parameters c_{set} and f_{set} represent the substrate concentration and flow delivered by the sterilizer.

```
proc $\mathit{Sterilizer}$
$(f_s : \uparrow \mathrm{fc},\ a_s : \mathrm{bool},\ \delta, c_{set}, f_{set} : \mathrm{real}) =$
$[\![\ f : \mathrm{flow},\ b : \mathrm{bool}$
$|\ (f, c_{set}) \multimap f_s$
$|\ b := \mathrm{false};\ f := 0$
$;\ *[\ b;\ \Delta\delta \longrightarrow f := f_{set}$
   $[]\ a_s?b \quad \longrightarrow [\ \neg b \longrightarrow f := 0\ ]$
   $]$
$]\!]$
```

The control system

The process FC specifies the control actions for the fermentor, pump and valve. For each fermentor there is a separate control process FC (see Figure 3 for an overview of the structure of the complete plant). The specification of the process, which follows below, is now explained. After synchronizing with *BatchScheduler* via channel j, a new batch is initiated by the synchronization $i^{\sim}$. The pump is switched on at high speed ($a_p!f_{high}$), so that the fermentor is quickly charged with substrate until the volume equals V_{min}. The reaction then starts and the control system waits until the substrate concentration drops below the value c_{set}, which indicates a significant growth of the viable cell mass. In order to maintain the substrate concentration at a reasonable level, the substrate feed is started ($a_p!f_{low}$). The fermentor is now slowly fed a substrate solution until the substrate concentration drops below c_{min} (see line (1)), which indicates that the product concentration is maximal. Consequently the feed is switched off. In the case of insufficient growth (due to a low value of μ_{max}), the volume of the fermentor may exceed the maximum level ($V \geq V_{max}$) before the substrate concentration drops below c_{min}. At this point the feed is also switched off and the reaction is allowed to finish (see line (2)). The fermentor is emptied when $c \leq c_{min}$ because the product concentration is then almost at its maximum value. This is done by opening the valve until the fermentor is empty. After 0.5 hours, the time assumed necessary for sterilization of the equipment, the fermentor is ready to start a new batch.

```
proc $FC$
$(c_z : \downarrow \mathrm{conc},\ V_f : \downarrow \mathrm{vol},\ a_p : !\,\mathrm{real},\ a_v : !\,\mathrm{bool},$
 $i, j : {}^{\sim},\ f_{low}, f_{high}, c_{set}, c_{min}, V_{max}, V_{min} : \mathrm{real}$
$) =$
$[\![\ c : \mathrm{conc},\ V : \mathrm{vol}$
$|\ c \multimap c_z,\ V \multimap V_f$
$|\ *[\ j^{\sim}$
   $;\ i^{\sim};\ a_p!f_{high};\ \nabla\, V \geq V_{min};\ a_p!0$
   $;\ \nabla\, c \leq c_{set};\ a_p!f_{low}$
   $;\ [\ \nabla\, c \leq c_{min} \quad \longrightarrow a_p!0$                  (1)
     $[]\ \nabla\, V \geq V_{max} \longrightarrow a_p!0;\ \nabla\, c \leq c_{min}$          (2)
     $]$
```

```
    ; $a_v$!true ; $\nabla V = 0$ ; $a_v$!false ; $\Delta\, 0.5$
    ]
⟧
```

The process *LC* controls the level of the substrate solution in the feed tank. Channel V_e relates the variable V to the volume of the substrate in the feed tank. Initially, the sterilizer is switched on (a_s!true). When the volume in the feed tank exceeds V_{max} the sterilizer is switched off, and when the volume drops below V_{min} the sterilizer is switched on again. It is assumed that the feed tank is empty initially.

```
proc $LC(V_e : \downarrow \text{vol},\ a_s : !\,\text{bool},\ V_{max}, V_{min} : \text{real}) =$
⟦ $V$ : vol
| $V \multimap V_e$
| $*[\ a_s!\text{true} ; \nabla V \geq V_{max} ; a_s!\text{false} ; \nabla V \leq V_{min}\ ]$
⟧
```

The process *BatchScheduler* is used to start the different batches. The scheduler makes sure that different batches are not started at the same time. This is done by waiting a certain amount of time after each command to start a new batch. In this way the feed tank can be kept relatively small. Although this may not be very important when only two fermentors are used, in real systems the number of fermentors can be considerably larger. The process terminates when 10 batches have been produced ($n = 10$). Because it is not known in advance which of the two fermentors will be ready first, the scheduler waits in the selective waiting statement $[j_1{}^\sim\ [\,]\ j_2{}^\sim]$ to synchronize with either of the two fermentor controllers.

```
proc $BatchScheduler(j_1\,, j_2 : {}^\sim,\ t : \text{real}) =$
⟦ $n$ : int
| $n := 0$
; $*[\ n < 10 \longrightarrow [\ j_1{}^\sim\ [\,]\ j_2{}^\sim\ ]\ ;\ \Delta t\ ;\ n := n + 1\ ]$
⟧
```

The total system

The components presented in previous sections are combined in the system *Plant*. The graphical representation of the system is shown in Figure 3.

```
syst $Plant =$
⟦ $f_{e1}, f_{e2}, f_s$ : fc, $c_{z1}, c_{z2}$ : conc, $V_e, V_{f1}, V_{f2}$ : vol,
, $a_{p1}, a_{p2}$ : real, $a_{v1}, a_{v2}, a_s$ : bool, $i_1, i_2, j_1, j_2 : {}^\sim$
| $BatchScheduler(j_1, j_2, 1)$
‖ $FC(c_{z1}, V_{f1}, a_{p1}, a_{v1}, i_1, j_1, 1.2, 18, 95, 0.5, 15, 7.5)$
‖ $FC(c_{z2}, V_{f2}, a_{p2}, a_{v2}, i_2, j_2, 1.2, 18, 95, 0.5, 15, 7.5)$
‖ $LC(V_e, a_s, 10, 3)$
‖ $F(f_{e1}, V_{f1}, c_{z1}, i_1, a_{p1}, a_{v1})$
‖ $F(f_{e2}, V_{f2}, c_{z2}, i_2, a_{p2}, a_{v2})$
‖ $FeedTank(f_s, f_{e1}, f_{e2}, V_e)$
‖ $Sterilizer(f_s, a_s, 0.25, 100, 15)$
⟧
```

Concluding remarks

As the complexity of systems in the process industry is increasing, design aids are required for the analysis of such systems. Using the language χ, a precise and clear specification of such a system helps in understanding the system. Its symbolic mathematical notation enhances the readability of the specifications. In order to analyse the characteristics of the system, the specification needs to be simulated. For this purpose, the symbolic specification is transformed into a simulation program by simply replacing the mathematical symbols by ASCII equivalents.

The integration of continuous-time and discrete-event concepts enables the modelling and analysis of complete plants or production units, instead of concentrating on individual subsystems of either a continuous-time or discrete-event nature. Controllers and controlled systems can both be specified in χ; it is in fact possible to specify a continuous-time production facility with a discrete-event sequence control system. It is our aim that the specification of the control systems in χ can eventually be used for actual real-time control.

The specification of the plant for the production of industrial ethanol has illustrated the relevance of integrating continuous-time and discrete-event models. The continuous-time constructs are used for the specification of the DAEs. The discrete-event constructs are used for the specification of the discrete control actions, such as the scheduling of batches that are processed in parallel. Simulation of such a system is an important aid in the design of the system when, due to stochastic effects, the time needed for the completion of a process step is not known in advance.

Currently a χ simulator for the discrete-event part of the language has been developed. It is being used for modelling and simulation of complex discrete-event manufacturing systems such as production facilities of integrated circuits. A first prototype of the χ simulator for combined continuous-time / discrete-event systems has been used to simulate the specification of the plant presented in this paper. For the results we refer to (Gordijn, 1995). A more definitive version of the simulator is under construction.

REFERENCES

Aspen, 1991, ”Speedup User Guide”, Aspen Technology, Cambridge.

Arends, N.W.A., Mortel-Fronczak, J.M. van de, and Rooda, J.E., 1994, ”Continuous Systems Specification Language”, *CISS - First joint Conference of International Simulation Societies proceedings*, Zürich, Switzerland, pp. 76-79.

Bailey, J.E., and Ollis, D.F., 1977, ”Biochemical Engineering Fundamentals”, New York, McGraw-Hill.

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

Gordijn, S.H.F., 1995, "Combined Continuous-Time / Discrete-Event Specifications of Industrial Systems Using the Language χ", M.S. Thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

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.

Mitchell, E.E.L., and Gauthier, J.S., 1976, "Advanced Continuous Simulation Language (ACSL)". *Simulation 26*, No.3, pp. 72-78.

Mortel-Fronczak, J.M. van de, and Rooda, J.E., 1995, "Application of Concurrent Programming to Specification of Industrial Systems", Accepted for INCOM'95/IFAC, Bejing.

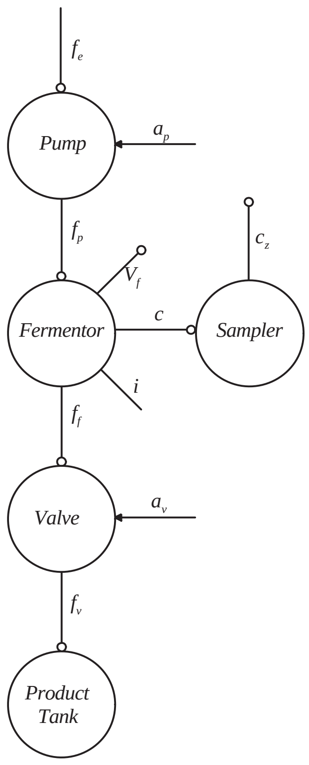

Fig. 2 System F

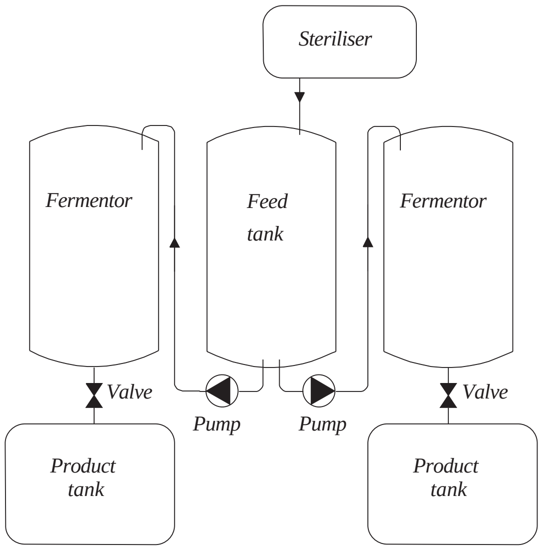

Fig. 1 Flowsheet of the plant

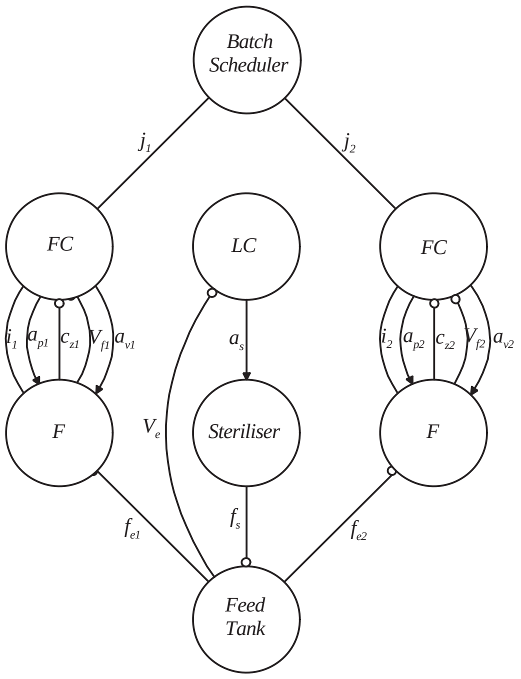

Fig. 3 System *Plant*

Table 1 BNF syntax of the continuous-time part of χ

GE	$::=$	b ‘$\longrightarrow$’ EQ $\mid$ GE ‘$\mid$’ GE
EQ	$::=$	eqx ‘$=$’ eqx $\mid$ ‘[’ GE ‘]’ $\mid$ EQ ‘,’ EQ
EQD	$::=$	‘{’ EQ ‘}’

Table 2 BNF syntax of the discrete-event part of χ

E	$::=$	c ‘!’ e $\mid$ c ‘?’ x $\mid$ c ‘$^{\sim}$’ $\mid$ ‘Δ’ t $\mid$ ‘∇’ bc
GB	$::=$	b ‘$\longrightarrow$’ S $\mid$ GB ‘$[]$’ GB
GW	$::=$	b ‘;’ E ‘$\longrightarrow$’ S $\mid$ GW ‘$[]$’ GW
G	$::=$	‘[’ GB ‘]’ $\mid$ ‘[’ GW ‘]’
S	$::=$	x ‘$:=$’ e $\mid$ E $\mid$ G $\mid$ ‘$*$’ G $\mid$ S ‘;’ S