Hybrid Modeling and Simulation of plant/controller Combinations

R.R.H. Schiffelers, A.Y. Pogromsky, D.A. van Beek, and J.E. Rooda

Abstract— **In order to design controllers, models of the system to be controlled (the plant), and models of the controller are developed, and the performance of the controlled system is evaluated by means of, for instance, simulation. The plant and controller can be modeled in the continuous-time domain, the discrete-event domain, or in a combination of these two domains, the so-called hybrid domain. It is very convenient to have all these combinations available in one single formalism. In this paper, the hybrid χ (Chi) formalism is introduced and used to model a simple manufacturing system consisting of a production machine that is controlled by a PI controller with anti-windup. The plant is modeled in the continuous-time domain as well as in the discrete-event domain. Likewise, the controller is modeled in both domains. Then, several (hybrid) plant/controller combinations are made. It is shown that the χ language facilitates modeling of these combinations, because the individual plant and controller specifications remain unchanged. The χ simulator is used to obtain the respective simulation results.**

I. INTRODUCTION

Today's manufacturing systems have become highly dynamic and complex. In order to stay competitive, manufacturing systems must use a good control strategy to rapidly respond to demand fluctuations. Simple discrete-event manufacturing systems can be controlled by policies such as PUSH, CONWIP, or Kanban (see e.g. [1]). However, as manufacturing systems become more complex, these policies become less effective.

Supervisory control theory [2], which is also based on a discrete-event specification of the manufacturing system, was proposed in the 1980's as a more structured approach for the control of manufacturing systems. A disadvantage of this approach, however, is that for the control of large manufacturing systems (or networks of such systems), supervisory control is not very suitable due to the large state space involved, which causes the corresponding control problem to grow intractably large.

A different control approach is based on the use of ordinary differential equations (ODE's) to model a manufacturing system (see e.g. [3], [4]). Such ODE models are a continuous-time approximation of the discrete-event system and as a result the control problem is much simpler. Moreover, control theory for ODE's is widely available, which makes it attractive to work with such models.

This work was partially done as part of the Darwin project under the responsibility of the Embedded Systems Institute, partially supported by the Netherlands Ministry of Economic Affairs under the BSIK program, as part of the ITEA project Twins 05004, and as part of the Collaborative Project MULTIFORM, contract number FP7-ICT-224249.

Eindhoven University of Technology Postbus 513, 5600 MB Eindhoven, The Netherlands. {R.R.H. Schiffelers, A.Y. Pogromsky, D.A. van Beek, J.E. Rooda}@tue.nl

In order to design a discrete-event controller for a discrete-event manufacturing system (plant) one could use the following design flow.

- First, the plant is modeled in the continuous-time domain by means of ODE's. Using control theory for ODE's, a continuous-time controller can be designed. By means of simulation, proper controller parameters can be chosen.
- Then, the continuous-time controller can be discretized to the discrete-event domain. Combining the continuous-time model of the plant and the discrete-time model of the controller enables studying the effect of the sampling time of the controller.
- Then, the plant is modeled in the discrete-event domain. Combining the discrete-event model of the plant and the designed continuous-time controller, the effect of the continuous-time approximation of the plant on the controller design can be investigated.
- Finally, the discrete-event model of the plant and the discrete-event model of the controller can be combined. By means of simulation, the performance of the controlled plant can be evaluated, and the controller parameters can be tuned.

Clearly, it is very convenient to have all these plant and controller combinations available in one single formalism. The combination of continuous-time models and discrete-event models leads to so-called hybrid models. Modeling and simulation of these hybrid models require a hybrid formalism and associated (simulation) tools. Nowadays, there exists a variety of hybrid modeling formalisms and associated simulation tools, see [5] for a classification and overview.

The hybrid χ language [6], [7] is such a hybrid formalism. It has a relatively straightforward and elegant syntax and formal semantics that is highly suited to modeling. The intended use of hybrid χ is for modeling, simulation, verification, and real-time control of industrial systems. It integrates continuous-time and discrete-event concepts, and enables analysis of the dynamic behavior of hybrid processes, of hybrid embedded systems, as well as of complete hybrid plants. In this paper, the hybrid χ language is introduced and used to model a simple manufacturing system consisting of a production machine that is controlled by a PI controller with anti-windup. The plant is modeled in the continuous-time (CT) domain as well as in the discrete-event (DE) domain. Likewise, the controller is modeled in both domains. Without changing any of these individual models, the following plant/controller combinations are made. The CT plant controlled by the CT controller, the DE plant controlled by the CT controller, CT plant controlled by the DE controller, and the DE plant controlled by the DE controller.

The outline of this paper is as follows. In Section II, the simple manufacturing system is described. The syntax and semantics of the hybrid χ language are described in Section III. Using the hybrid χ language, the manufacturing system and its controller are modeled both in the continuous-time domain as well as in the discrete-event domain, see Section IV. The four plant/controller combinations and their simulation results are given in Section V. Finally, Section VI concludes the paper.

II. A SIMPLE MANUFACTURING SYSTEM

This section introduces a simple manufacturing system that is considered throughout this paper. The system, as depicted in Fig. 1, consists of a single machine producing lots from an infinite capacity buffer. It is assumed that the supply of raw materials to the buffer is always sufficient, such that the machine never starves.

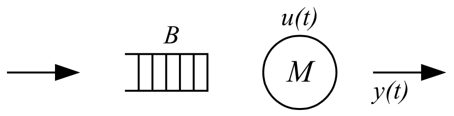

Fig. 1. Manufacturing system with buffer B and machine M.

The machine processes lots from the buffer with a process rate $u(t)$, which can be interpreted as the velocity at which the machine operates. It is assumed that the process rate cannot become negative and that it cannot exceed some maximum rate $u_{\max}$. The relation for the cumulative number of products that has been processed by the machine, $y(t)$, is given as

$$\dot{y}(t) = u(t). \tag{1}$$

The machine can be interpreted as a pure integrator. By using a feedback controller to set the production rate $u(t)$ one can control the cumulative output $y(t)$ such that the machine can track a given desired production $y_{\mathrm{d}}(t)$. In this paper, the reference production is given by

$$y_{\mathrm{d}}(t) = u_{\mathrm{d}}t + y_{\mathrm{d}0} + r(t), \tag{2}$$

which has a part that is linear, where u_{d} is the desired production rate and $y_{\mathrm{d}0}$ is the desired production at $t = 0$ and a bounded term $r(t) = b \cdot \sin(\omega t)$, which can be interpreted as a – for instance seasonal – fluctuation of the demand.

It can be argued by means of the final value theorem from linear control theory (see for instance [8]) that for $r(t) = 0$ a controller with integral action should be used to track the error given by

$$e(t) = y_{\mathrm{d}}(t) - y(t) \tag{3}$$

to zero. Therefore, we also choose to use an integral action for the case where $r(t) \neq 0$. The simplest controller with integral action is a PI controller.

The combination of input saturation and the integrator in the PI controller leads to a phenomenon called integrator windup. When the actuator saturates, the effective control signal cannot exceed some value, which affects the system behavior and therefore again the control signal. As a result of this, the closed-loop performance of the system can deteriorate, and in some situations the system can even become unstable. By adding a so-called anti-windup controller to the system, this loss of performance can be counteracted by "turning off" the integrator in the controller when the machine saturates. In the past, many anti-windup controllers have been proposed in literature. In this paper we use the anti-windup design as presented in [9]. This leads to the following equation for the control signal $u(t)$:

$$u(t) = \begin{cases} 0, & y_{\mathrm{c}}(t) < 0 \\ y_{\mathrm{c}}(t), & 0 \leq y_{\mathrm{c}}(t) \leq u_{\max} \\ u_{max}, & y_{\mathrm{c}}(t) > u_{\max} \end{cases} \tag{4}$$

where $y_{\mathrm{c}}(t)$ is given by

$$\begin{aligned} y_{\mathrm{c}}(t) &= k_{\mathrm{P}}e(t) + y_{\mathrm{I}}(t), \\ \dot{y_{\mathrm{I}}}(t) &= k_{\mathrm{I}}(e(t) - k_{\mathrm{A}}(y_{\mathrm{c}}(t) - u(t))) \end{aligned} \tag{5}$$

A specific choice of the controller parameters k_{P}, k_{I}, and k_{A} (satisfying $k_{\mathrm{P}} > 0$, $k_{\mathrm{I}} > 0$, and $k_{\mathrm{P}}k_{\mathrm{A}} > 1$, see [9]) has to be made based on *performance* criteria, for instance certain demands for the sensitivity and complementary sensitivity.

III. THE HYBRID χ LANGUAGE

In this section, the syntax and semantics of (a subset of) the hybrid χ language are discussed informally. A more detailed explanation of hybrid χ can be found in [6], [7]. Information about the freely available χ toolset can be found at [10]. This toolset includes, amongst others, a stand-alone simulator [11] equipped with symbolic and numerical solvers such as MAPLE [12] and DASSL [13], and a co-simulator [14] for simulation of models that consist of subsystems modeled using MATLAB / SIMULINK [15] and subsystems modeled in χ.

A. Syntax

A χ model identified by $name$ is of the following form:

model $name() =$
$[\![$ var $s_1 \ : type_{s_1} \ = c_1, \ldots, s_k : type_{s_k} \ = c_k$
, cont $x_1 : type_{x_1} \ = d_1, \ldots, x_m : type_{x_m} \ = d_m$
, alg $y_1 : type_{y_1} \ = e_1, \ldots, y_n : type_{y_n} \ = e_n$
, chan $h_1 : type_{h_1}, \ldots, h_q : type_{h_q}$
, mode $X_1 = p_1, \ldots, X_r = p_r$
:: p
$]\!]$

Here, $type_i$ denotes a type, for instance bool, nat, or real. Notation var $s_1 \ : type_{s_1} \ = c_1, \ldots, s_k : type_{s_k} \ = c_k$ denotes the declaration of discrete variables $s_1, \ldots, s_k$ with their respective types $type_{s_1}, \ldots, type_{s_k}$ and initial values $c_1, \ldots, c_k$. Similarly, notations cont $x_1 : type_{x_1} \ = d_1, \ldots, x_m : type_{x_m} \ = d_m$ and alg $y_1 : type_{y_1} \ = e_1, \ldots, y_n : type_{y_n} \ = e_n$ are used to declare continuous and algebraic variables, respectively.

The main differences between discrete, continuous, and algebraic variables are as follows: First, the values of discrete variables remain constant when model time progresses, the values of continuous variables may change according to a continuous function of time when model time progresses, and the values of algebraic variables may change according to a discontinuous function of time. Second, the values of the discrete and continuous variables do not change in action transitions unless such changes are explicitly specified, for example by assigning a new value. The values of algebraic variables can change arbitrarily in action transitions, unless such changes are explicitly restricted, for example by assigning a new value. Third, there is a difference between the different classes of variables with respect to how the resulting values of the variables in a transition relate to the starting values of the variables in the next transition. The resulting value of a discrete or continuous variable in a transition always equals its starting value in the next transition. For algebraic variables, there is no such relation. In most models, the values of discrete variables are defined by assignments, whereas the values of algebraic variables are defined by equations ((in)equalities).

Notation $\mathsf{chan}\ h_1 : type_{h_1}, \ldots, h_q : type_{h_q}$ declares the channels $h_1, \ldots, h_q$, and $\mathsf{mode}\ X_1 = p_1, \ldots, X_r = p_r$ declares mode variables $X_1, \ldots, X_r$ with their respective statement definitions $p_1, \ldots, p_r$. The χ language consists of the following statements $p, q, r \in P$:

P	$::=$	$\mathbf{x}_n := \mathbf{e}_n$	(multi-) assignment
	$\mid$	$\mathsf{eqn}\ u$	equation
	$\mid$	$\mathrm{delay}\ d$	delay statement
	$\mid$	X	mode variable
	$\mid$	$b \to P$	guard operator
	$\mid$	$*P$	repetition
	$\mid$	$P;\ P$	sequential composition
	$\mid$	$P \,[\!]\, P$	alternative composition
	$\mid$	$P \parallel P$	parallel composition
	$\mid$	$h\,!\,\mathbf{e}_n$	send statement
	$\mid$	$h\,?\,\mathbf{x}_n$	receive statement
	$\mid$	$l_\mathrm{p}(\mathbf{x}_n, \mathbf{h}_n, \mathbf{e}_n)$	process instantiation

Here, $\mathbf{x}_n$ denotes the (non-dotted) variables $x_1, \ldots, x_n$ such that $\mathsf{time} \notin \{\mathbf{x}_n\}$, $\mathbf{e}_n$ denotes the expressions $e_1, \ldots, e_n$, u and b are both predicates over variables (including the variable time) and dotted continuous variables, d denotes a numerical expression, h denotes a channel, , l_p denotes a process identifier, and $\mathbf{h}_n$ denotes the channels $h_1, \ldots, h_n$. The operators are listed in descending order of their binding strength as follows $\to$, ; , $\{\parallel , [\!]\ \}$. The operators inside the braces have equal binding strength. For example, $x := 1;\ y := x \,[\!]\, x := 2;\ y := 2x$ means $(x := 1;\ y := x) \,[\!]\, (x := 2;\ y := 2x)$. Parentheses may be used to group statements. To avoid confusion, parenthesis are obligatory when alternative composition and parallel composition are used together. E.g. $p \,[\!]\, q \parallel r$ is not allowed and should either be written as $(p \,[\!]\, q) \parallel r$, or as $p \,[\!]\, (q \parallel r)$.

A *multi-assignment* $\mathbf{x}_n := \mathbf{e}_n$ denotes the assignment of the values of the expressions $\mathbf{e}_n$ to the variables $\mathbf{x}_n$.

A *equation* $\mathsf{eqn}\ u$, usually in the form of a differential algebraic equation, restricts the allowed behavior of the continuous and algebraic variables in such a way that the value of the predicate u remains true over time.

A *delay statement* $\mathrm{delay}\ d$ delays for d time units and then terminates by means of an internal action.

Mode variable X denotes a mode variable (identifier) that is defined at the declarations. Among others, it is used to model repetition. Mode variable X can do whatever the statement of its definition can do.

The *guarded process term* $b \to p$ can do whatever actions p can do under the condition that the guard b evaluates to true. The guarded process term can delay according to p under the condition that for the intermediate valuations during the delay, the guard b holds. The guarded process term can perform arbitrary delays under the condition that for the intermediate valuations during the delay, possibly excluding the first and last valuation, the guard b does not hold.

Repetition $*p$ denotes the infinite repetition of p.

Sequential composition operator term $p;\ q$ behaves as process term p until p terminates, and then continues to behave as process term q.

The *alternative composition operator term* $p \,[\!]\, q$ models a non-deterministic choice between different actions of a process. With respect to time behavior, the participants in the alternative composition have to synchronize.

Parallelism can be specified by means of the *parallel composition operator term* $p \parallel q$. Parallel processes interact by means of shared variables or by means of synchronous point-to-point communication/synchronization via a channel. The parallel composition $p \parallel q$ synchronizes the time behavior of p and q, interleaves the action behavior (including the instantaneous changes of variables) of p and q, and synchronizes matching send and receive actions. The synchronization of time behavior means that only the time behaviors that are allowed by both p and q are allowed by their parallel composition.

By means of the *send process term* $h\,!\,\mathbf{e}_n$, for $n \geq 1$, the values of expressions $\mathbf{e}_n$ are sent via channel h. For $n = 0$, this reduces to $h\,!$ and nothing is sent via the channel.

By means of the *receive process term* $h\,?\,\mathbf{x}_n$, for $n \geq 1$, values for $\mathbf{x}_n$ are received from channel h. We assume that all variables in $\mathbf{x}_n$ are different. For $n = 0$, this reduces to $h\,?$, and nothing is received via the channel. Communication in χ is the sending of values by one parallel process via a channel to another parallel process, where the received values (if any) are stored in variables. For communication, the acts of sending and receiving (values) have to take place in different parallel processes at the same moment in time. In case no values are sent and received, we refer to synchronization instead of communication.

Process instantiation process term $l_\mathrm{p}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$, where l_p denotes a process label, enables (re-)use of a process definition. A process definition is specified once, but it can be instantiated many times, possibly with different parameters:

external variables $\mathbf{x}_k$, external channels $\mathbf{h}_m$, and expressions $\mathbf{e}_n$.

Chi specifications in which process instantiations $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$ are used have the following structure:

$$\begin{array}{l} pd_1 \\ \vdots \\ pd_j \\ \\ \mathsf{model}\ \cdots = \\ \|[\ \mathsf{var}\ \ldots,\ \mathsf{chan}\ \ldots,\ \mathsf{mode}\ \ldots :: p\ ]\|, \end{array}$$

where for each process instantiation $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$ occurring in process term q, a matching process definition pd_i $(1 \leq i \leq j)$ of the form

$$\mathsf{proc}\ l_{\mathrm{p}}(\mathsf{var}\ \mathbf{x}'_k : \mathbf{t}_{x_k}, \mathsf{chan}\ \mathbf{h}'_m : \mathbf{t}_{h_m}, \mathsf{val}\ \mathbf{v}_n : \mathbf{t}_{v_n}) = D :: p$$

must be present among the j process definitions $pd_1 \ldots pd_j$. Here l_{p} denotes a process label, $\mathbf{x}_k$ denotes the 'actual external' variables $x_1, \ldots, x_k$, $\mathbf{h}_m$ denotes the 'actual external' channels $h_1, \ldots, h_m$, $\mathbf{e}_n$ denotes the expressions $e_1, \ldots, e_n$, $\mathbf{x}'_k : \mathbf{t}_{x_k}$ denotes the 'formal external' variable definitions $x'_1 : t_{x_1}, \ldots, x'_k : t_{x_k}$, $\mathbf{a}'_l$ denotes the 'formal external' action definitions $a'_1, \ldots, a'_l$, $\mathbf{h}'_m : \mathbf{t}_{h_m}$ denotes the 'formal external' channel definitions $h'_1 : t_{h_1}, \ldots, h'_m : t_{h_m}$, and $\mathbf{v}_n : \mathbf{t}_{v_n}$ denotes the 'value parameter definitions' $v_1 : t_{v_1}, \ldots, v_n : t_{v_n}$.

Notation D denotes declarations of local local (discrete, continuous or algebraic) variables, local channels, and local mode definitions. The only free (i.e. non-local) variables, and free channels that are allowed in process term p are the formal external variables $\mathbf{x}'_k$ and the value parameters $\mathbf{v}_n$, the formal external actions $\mathbf{a}'_l$, and the formal external channels $\mathbf{h}'_m$, respectively. We assume that the formal external variables $\mathbf{x}'_k$ and the value parameters $\mathbf{v}_n$ are different.

B. Formal semantics

The semantics of χ is defined by means of deduction rules in SOS style [16] that associate a hybrid transition system with a χ model as defined in [6]. The hybrid transition system consists of action transitions and time transitions. Action transitions define instantaneous changes, where time does not change, to the values of variables. Time transitions involve the passing of time, where for all variables their trajectory as a function of time is defined.

IV. Modeling the components of the manufacturing system using Chi

In this section, the components demand, plant, and controller are modeled. In Section V, these models are used to compose the different plant/controller combinations.

A. CT model of the demand

Process $DemandCT$ models the demand *yd*, as given by Equation 2. It is parameterized by the maximum processing rate $uMax$.

```
proc DemandCT( alg yd : real, val uMax : real ) =
|[  alg r :  real
 , var b        : real  = 12.5
   , omega  : real  =  1.0
   , yd0    : real  =  2.0
   , ud     : real  = uMax / 2.0
:: eqn yd  = ud ∗ time + yd0 + r
   ,  r    = b ∗ sin ( omega ∗ time )
]|
```

B. CT model of the plant

Process $PlantCT$ models the continuous-time approximation of the plant. The production rate is given by (input) variable u, and the cumulative number of processed products is modeled by (output) variable y. The relation between y and u is given by the equation eqn dot *y* = *u* that models Equation 1.

```
proc PlantCT( cont y :  real ,  alg  u :  real  ) =
|[  eqn dot y  = u ]|
```

C. CT model of the controller

Process $ControllerCT$ models the continuous-time controller. The demand *yd* and the cumulative number of processed products *y* are given as input. The unsaturated control signal is given by (local) variable yc, and the saturated control signal is given by (output) variable u. The system of equations model the controller as described in Section II by Equations 3, 4, 5. The controller is parameterized with the maximum processing rate *uMax*.

```
proc ControllerCT(  alg yd, y,  u :  real
                  ,  val  uMax, Kp, Ki, Ka :  real
                  ) =
|[  cont yI      :  real  = 0.0
 ,  alg yc,  e   :  real
::  eqn e         = yd − y
    ,   u         = ( yc   < 0.0                 → 0.0
                    | yc ≤  0.0  and yc ≤ uMax   → yc
                    | yc   > uMax                → uMax
                    )
    ,  yc         = Kp ∗ e + yI
    ,  dot yI     = Ki ∗ ( e −   Ka ∗ (yc − u))
]|
```

D. DE model of the plant

Figure 2 shows a graphical representation of the structure of the discrete-event model of the plant.

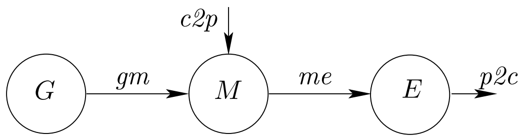

Fig. 2. Iconic model of DE plant.

It consists of processes modeling an infinite buffer G, the machine M, and an exit process E. The processes are represented by circles labeled with the name of the process. The processes are connected via (directed) channels, that are represented by arrows labeled with the name of the channel. The χ model of the discrete-event plant is given below.

```
proc PlantDE( chan p2c!, c2p? : real , val uMax : real ) =
|[ chan gm, me : real
:: G( gm )
|| M( gm, me, c2p, uMax )
|| E( me, p2c )
]|
```

Via channel *p2c* the cumulative number of processed products can be send to the controller, and processing rates can be received from the controller via channel *c2p*. The local channels *gm* and *me* connect the buffer, machine and exit process.

The infinite buffer is modeled by means of process G. Via channel *gm*, it is always possible to send a product to the machine. In this case, a product is modeled by means of the time-point that the product is send to the machine, (denoted by the value of pre-defined variable time)[1].

```
proc G( chan gm! : real ) =
|[ * gm!time ]|
```

The machine is modeled by means of process M. Similar to the CT model of the machine, the machine has a variable processing rate u, and new processing rates, that are determined by the controller, are taken into account immediately. The machine receives products *x* from the buffer via channel *gm*, and sends the processed products to the exit process via channel *me*. New processing rates *u* can be received from the controller via channel *c2p*. Local variables $tstart$, $tstop$, and $frac$ denote the time-point when processing a product is started (stopped), and the fraction of the total processing time of a product that already passed, respectively. The model consists of four mode definitions, $Idle$, $Start$, $Working$, and $ReStart$, that model the different states of the machine. Initially, the machine is Idle. After receiving a product via channel gm, the state of the machine changes to Start. If the processing rate exceeds 0, the machine starts processing the product (state Working). After processing the product, the product is sent to the exit process via channel me. During processing the product, the machine is interrupted when a new processing rate is received from the controller, and the state of the machine changes to Restart. After receiving a positive production rate, the remainder of the product is processed.

```
proc M( chan gm?, me!, c2p? : real , val uMax : real ) =
|[ var x, u : real = (0.0, 0.0)
     , tstart , tstop , frac : real = (0.0, 0.0, 0.0)
 , mode Idle    = ( gm?x; Start | c2p?u; Idle )
 , mode Start   = ( u >0
                      → (tstart , tstop ) := ( time, time + 1 / u)
                        ; Working
                  | c2p?u; Start
                  )
 , mode ReStart  = ( u >0
                      → tstop := (1.0 − frac ) ∗ 1 / u + time
                        ; Working
                    | c2p?u; ReStart
                    )
 , mode Working = ( time ≤ tstop  → me!!x; Idle
                  | c2p?u
                    ; frac := ( time−tstart ) / ( tstop −tstart )
                    ; ReStart
                  )
:: Idle
]|
```

[1]This value can be used, for instance, in the exit process to calculate the flow time of the product.

The exit process can always receive a product x from the machine via channel *me*. Subsequently, the cumulative number of processed products y is calculated ($y := y + 1.0$) and send to the controller via channel *p2c*.

```
proc E( chan me?: real , p2c!: real ) =
|[ var x, y : real = (0.0, 0.0)
:: ∗( me?x; y := y + 1.0; p2c!!y )
]|
```

E. DE model of the controller

Process $ControllerDE$ models a discrete-event model of the controller. Via channel *p2c*, the cumulative number of products that has been processed by the machine can be received. New control signals are send to the plant via channel *c2p*. In addition to the control parameters Kp, Ki, and Ka, this controller is parameterized with the $sampletime$. Every sample time (delay *sampletime*), the controller calculates the new control signal and sends it to the plant (*c2p*!!*u*). Also, when a product is finished (*p2c*?*y*), the control signal is calculated immediately. The calculation of the control signal is straightforward.

```
proc ControllerDE( chan p2c?, c2p!: real
                 , alg yd : real
                 , val uMax, sampletime, Kp, Ki, Ka : real
                 ) =
|[ var y, e, u : real = (0.0, 2.0, 0.0)
     , yc, yI, yIdot, tprev : real = (0.0, 0.0, 0.0, 0.0)
:: ∗( ( p2c?y | delay sampletime )
    ; e  := yd − y
    ; yIdot := Ki ∗ ( e − Ka ∗ (yc − u))
    ; yI := yI + ( time − tprev) ∗ yIdot
    ; yc := Kp ∗ e + yI
    ; u  := ( yc  <0.0                 → 0.0
            | yc ≤ 0.0 and yc ≤ uMax  → yc
            | yc  >uMax               → uMax
            )
    ; tprev := time
    ; c2p!!u
    )
]|
```

V. Modeling the different plant/controller combinations

In this section, the models of the demand, plant and controller components are combined. Note that none of component models has to be changed/adapted, regardless of the combination it is used in.

A. CT model of the plant and CT model of the controller

Figure 3 shows a graphical representation of the continuous-time model of manufacturing system.

It consists of the continuous-time model of the demand D, the continuous-time model of the plant P_{CT}, and the continuous-time model of the controller C_{CT}. The plant and

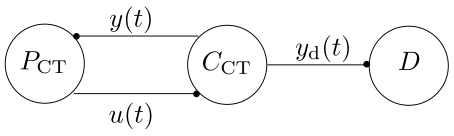

Fig. 3. Iconic model of CT plant CT controller.

controller are connected by means of the (shared) variables $y(t)$ and $u(t)$. These connections are visualized by means of lines between the processes. The solid circle on these lines denotes the process that 'determines/prescribes' the value of the variable (output variable), whereas the other process uses this variable as 'read-only' (input variable). Furthermore, the demand process and the controller are connected by means of variable $y_{\mathrm{d}}(t)$ that models the demand over time. The χ model is given below.

```
model CTCT() =
|[  var uMax : real  = 25.0
      , Kp, Ki,  Ka :  real  = (10.0, 20.0, 0.5)
 ,  cont y :  real  =  0.0
 ,  alg yd,  u :  real
::  DemandCT( yd, uMax )
||  PlantCT( y,  u )
||  ControllerCT(  yd,  y,  u,  uMax, Kp, Ki,  Ka )
]|
```

Figure 4 shows the trajectory of the error that is obtained by means of simulation of the continuous-time model of the manufacturing system.

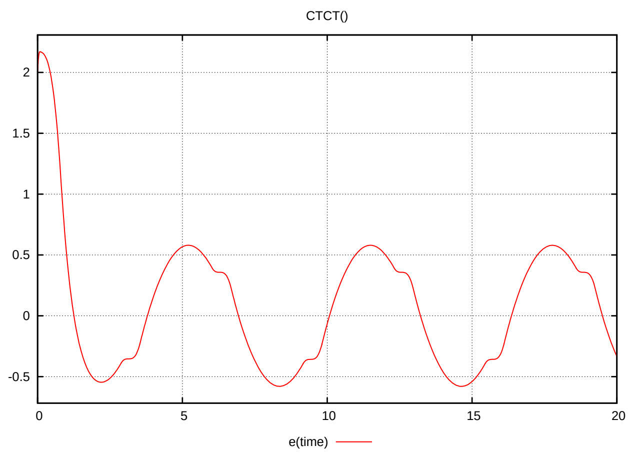

Fig. 4. CT plant CT controller

B. CT model of the plant and DE model of the controller

Figure 3 shows a graphical representation of the combined continuous-time / discrete-event model of the manufacturing system.

It consists of the continuous-time model of the demand D, the continuous-time model of the plant P_{CT}, and the discrete-event model of the controller C_{DE}. The χ model is given below.

```
model CTDE() =
|[  var uMax : real  = 25.0
```

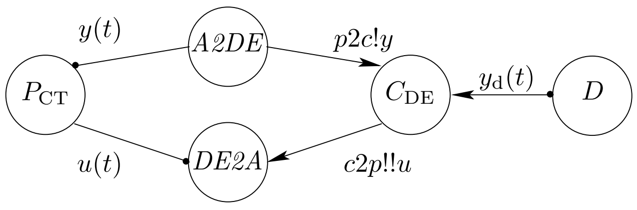

Fig. 5. Iconic model of CT plant DE controller.

```
      , Kp, Ki,  Ka    :  real  = (10.0, 20.0, 0.5)
      , sampletime     :  real  = 0.01
 ,  cont y :  real  = 0.0
 ,  alg   yd,  u :  real
 ,  chan c2p, p2c :  real
::  DemandCT( yd, uMax )
||  PlantCT( y,  u )
||  DE2A( u, c2p )
||  A2DE( y, p2c, sampletime )
||  ControllerDE(  p2c,  c2p,  yd,  uMax, sampletime
                   ,  Kp, Ki,  Ka )
]|
```

Since the modeling paradigms of the plant (continuous-time) and controller (discrete-event) differ, they cannot be connected directly. Using two simple 'converter' processes modeling an analog-to-digital converter (process *A2DE*), and a digital-to-analog converter (process *DE2A*), the plant and controller can be connected.

```
proc A2DE (  alg  x :  real ,  chan h! :  real
              ,  val  sampletime :  real
              ) =
|[ ∗(   h!x;  delay  sampletime ) ]|
```

Process $A2DE$ models an analog-to-digital converter. Variable x models the analog (input) signal. The process is parametrized with the sample-time, given by value parameter $sampletime$. Every $sampletime$, the value of x is sent via channel h.

```
proc DE2A( alg y :  real ,  chan h? :  real  ) =
|[  var  x :  real  = 0.0
:: ∗  h?x ||  eqn y = x
]|
```

Process $DE2A$ models a zero-order hold digital-to-analog converter. The value of the analog (output) signal modeled by variable y equals the last received value via channel h, resulting in a piecewise-constant signal.

Figure 6 shows the trajectory of the error. A detailed comparison between the simulation results of the different plant/controller combinations is beyond the scope of this paper.

C. DE model of the plant and CT model of the controller

Figure 7 shows a graphical representation of the hybrid system consisting of a discrete-event model of the plant and a continuous-time model of the controller.

The χ model is given below.

```
model DECT() =
|[  var uMax : real  = 25.0
      , Kp, Ki,  Ka    :  real  = (10.0, 20.0, 0.5)
```

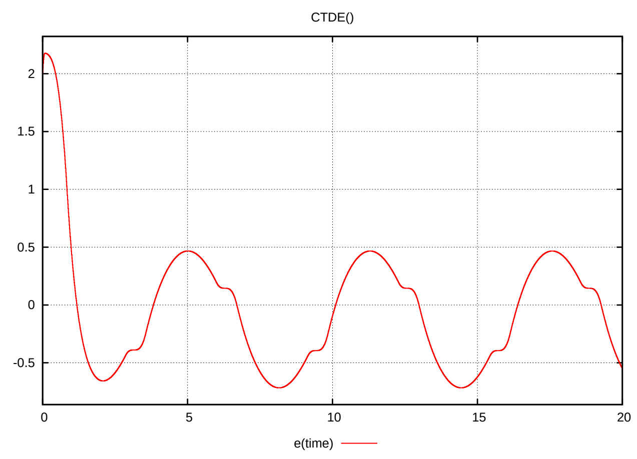

Fig. 6. CT plant DE controller

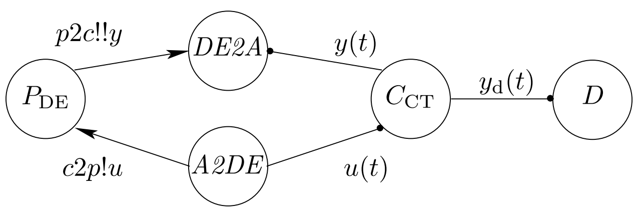

Fig. 7. Iconic model of DE plant CT controller.

```
      , sampletime   : real =  0.01
  , cont y :  real  = 0.0
   , alg yd,  u :  real
  , chan c2p, p2c :  real
::  DemandCT( yd, uMax )
||  PlantDE( p2c, c2p,  uMax )
||  DE2A( y, p2c )
||  A2DE( u, c2p, sampletime )
||  ControllerCT(  yd,  y,  u,  uMax, Kp, Ki,  Ka )
]|
```

Simulating the combination of the discrete-event plant and the continuous-time controller results in the trajectory of the error as shown in Figure 8.

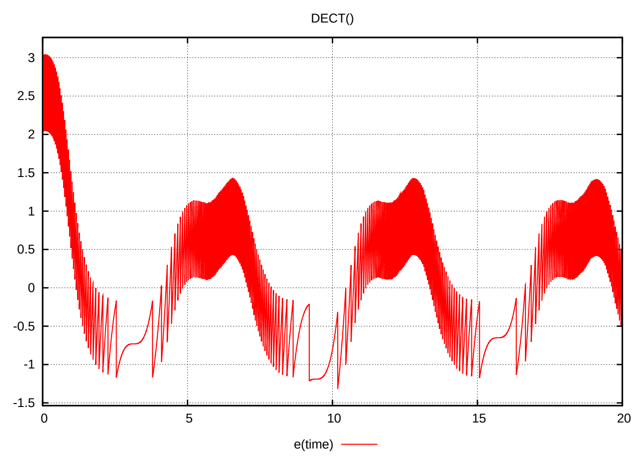

Fig. 8. DE plant CT controller

D. DE model of the plant and DE model of the controller

Figure 9 shows a graphical representation of the discrete-event system consisting of a discrete-event model of the plant and a discrete-event model of the controller.

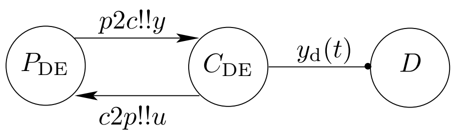

Fig. 9. Iconic model of DE plant DE controller.

The χ model is given below.

```
model DEDE() =
|[  var  uMax : real  = 25.0
       ,  Kp,  Ki,  Ka    :  real  = (10.0, 20.0, 0.5)
       ,  sampletime     :  real  = 0.01
 ,  alg  yd :   real
 ,  chan c2p, p2c :  real
::  DemandCT( yd, uMax )
||  PlantDE( p2c, c2p,  uMax )
||  ControllerDE(  p2c,  c2p,  yd,  uMax, sampletime
                 ,  Kp,  Ki,   Ka )
]|
```

Simulating the combination of the discrete-event plant and the discrete-event controller results in the trajectory of the error as shown in Figure 10.

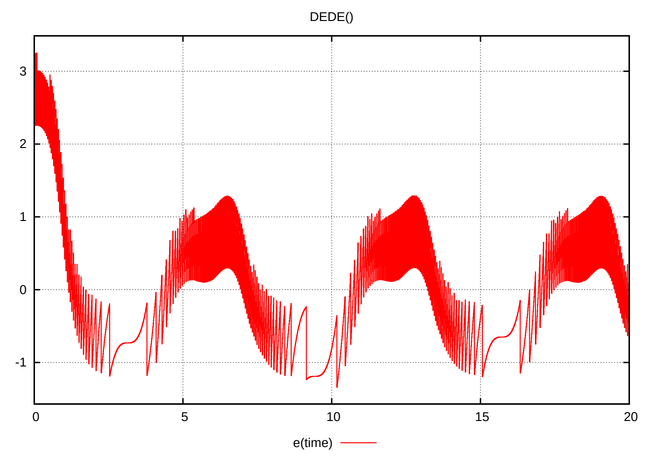

Fig. 10. DE plant DE controller

VI. Concluding Remarks

In this paper, the hybrid χ language is introduced and used to model a simple manufacturing system consisting of a production machine that is controlled by a PI controller with anti-windup. The plant is modeled in the continuous-time domain as well as in the discrete-event domain. Likewise, the controller is modeled in both domains. Then, several (hybrid) plant/controller combinations are made. It is shown that the χ language facilitates modeling of these combinations, because the individual plant and controller specifications remain unchanged. The χ simulator is used to obtain the respective simulation results.

Current and future work entails, amongst others, modeling and analysis of production networks and their controllers using the design flow as sketched in this paper.

References

[1] W. Hopp and M. Spearman, *Factory Physics*, 3$^{\text{rd}}$ ed. New York: McGraw-Hill, 2007.

[2] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event systems," *SIAM Journal on Control and Optimization*, vol. 25, pp. 206–230, 1987.

[3] R. Alvarez-Vargaz, Y. Dallery, and R. David, "A study of the continuous flow model of production lines with unreliable machines and finite buffers," *Journal of Manufacturing Systems*, vol. 13, no. 3, pp. 221–234, 1994.

[4] E. Boukas, "Manufacturing systems: LMI approach," *IEEE Transactions on Automatic Control*, vol. 51, no. 6, pp. 1014–1018, June 2006.

[5] D. A. v. Beek and J. E. Rooda, "Languages and applications in hybrid modelling and simulation: Positioning of Chi," *Control Engineering Practice*, vol. 8, no. 1, pp. 81–91, 2000.

[6] K. L. Man and R. R. H. Schiffelers, "Formal specification and analysis of hybrid systems," Ph.D. dissertation, Eindhoven University of Technology, 2006.

[7] D. A. v. Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers, "Syntax and consistent equation semantics of hybrid Chi," *Journal of Logic and Algebraic Programming*, vol. 68, no. 1-2, pp. 129–210, 2006.

[8] G. Franklin, D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[9] R. van den Berg, A. Pogromsky, and J. Rooda, "Convergent systems design: Anti-windup for marginally stable plants," in *Proceedings of 45th IEEE Conference on Decision and Control*, San Diego, 2006.

[10] Systems Engineering Group TU/e, "Chi toolset," http://se.wtb.tue.nl/sewiki/chi, 2008.

[11] D. A. v. Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers, "Deriving simulators for hybrid Chi models," in *IEEE International Symposium on Computer-Aided Control Systems Design*. Munich, Germany: IEEE, 2006, pp. 42–49.

[12] MapleSoft, "http://www.maplesoft.com," 2005.

[13] L. R. Petzold, "A description of dassl: A differential/algebraic system solver," *Scientific Computing*, pp. 65–68, 1983.

[14] D. A. v. Beek, A.T.Hofkamp, M. Reniers, J. E. Rooda, and R. R. H. Schiffelers, "Co-simulation of Chi and Simulink models," in *6th EUROSIM congress on Modelling and Simulation*, Ljubljana, Slovenia, 2007.

[15] The MathWorks, Inc, "Using Simulink, version 6," http://www.mathworks.com, 2005.

[16] G. D. Plotkin, "A structural approach to operational semantics," *Journal of Logic and Algebraic Programming*, vol. 60-61, pp. 17–139, 2004.