In *Proceedings of the 1995 EUROSIM Conference*,
Vienna, Sept. 1995, pp. 1029–1034.

A Combined Continuous-Time / Discrete-Event Approach to Modelling and Simulation of Manufacturing Machines

D.A. van Beek[a]*, J.E. Rooda[†], and S.H.F. Gordijn

[a]Eindhoven University of Technology, Department of Mechanical Engineering,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Using simulation models for the development and testing of control systems can have significant advantages over using real machines. This paper demonstrates the suitability of the language χ for machine modelling. The language integrates a small number of powerful orthogonal continuous-time and discrete-event concepts. The discrete-event part of χ is based on a CSP-like concurrent programming language; the continuous-time part is based on DAEs. Models are specified in a symbolic mathematical notation. A case study is presented of a transport system consisting of conveyor belts. The model of the system illustrates how the combined continuous-time/discrete-event approach simplifies accurate modelling of manufacturing machines.

1. INTRODUCTION

Using simulation models for the development and testing of control systems can have significant advantages over using real machines: using real machines may be hazardous because of the possibility of damage due to errors, or the real machines can still be under development. In such cases it is important that accurate models of machines can be developed relatively easily and fast. This is possible with the combined continuous-time/discrete-event language χ presented in this paper.

Many modelling languages are primarily suitable for the specification of either discrete-event or continuous-time models. In order to be able to specify combined systems, some languages have been extended with continuous-time or discrete-event constructs, respectively. As a consequence, the continuous-time and discrete-event parts of such languages are not equally powerful and are often based on different concepts (see for example [1] and [2]).

2. THE LANGUAGE χ

The modelling language χ has been designed with the objective of integrating continuous-time and discrete-event concepts in such a way, that the resulting language is equally well suitable for the modelling of pure continuous-time or discrete-event systems as well as for combined systems. Controlling systems and controlled systems can both be specified in χ. The language

*E-mail address: vanbeek@wtb.tue.nl; phone: +31-40-2472892; fax: +31-40-2452505.
[†]E-mail address: rooda@wtb.tue.nl; phone: +31-40-2474553; fax: 2452505.

consists of a small number of orthogonal constructs. In this paper only a small part of the language is used and explained.

The model of a system consists of a number of process (or system) instantiations, and channels connecting these processes. Processes and systems are defined as follows:

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

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

Processes have local variables only; all interaction between processes takes place by means of channels. A channel connects two processes and must be used for output in one process and input in the other. Channels are declared in systems and are either discrete (e.g. p : int) or continuous (e.g. p : [m/s]). Channels are also declared as process or system parameters in which case the usage of the channel must be declared as either output (e.g. $p : \uparrow$ [m/s] or p : ! int) or input (e.g. $p : \downarrow$ [m/s] or p : ? int). A continuous channel is represented graphically by a line ending in a small circle, a discrete channel by an arrow; processes are represented by circles.

A process may consist of a continuous-time part only, a discrete-event part only, or a combination of both.

2.1. The continuous-time part of χ

The continuous-time part of χ (see [3] for a preliminary description) is based on Differential Algebraic Equations (DAEs). The DAEs are specified between braces {}, separated by commas. 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 can be used: $[b_1 \longrightarrow DAE_1 \mid b_2 \longrightarrow DAE_2 \mid \ldots \mid b_n \longrightarrow DAE_n]$. The boolean expressions b_j $(1 \leq j \leq n)$ 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 of the DAEs 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 (e.g. *var* $-\circ$ *channel*). Consider two variables x_a, x_b in different processes linked to a continuous channel c ($x_a -\circ c$ and $x_b -\circ c$). The channel and links cause the equation $x_a = x_b$ to be added to the set of DAEs of the system.

2.2. The discrete-event part of χ

The discrete-event part of χ is a CSP-like ([4]) real-time concurrent programming language, described in [5] and [6]. The interaction between discrete-event parts of processes takes place by means of synchronous message passing. 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. Consequently the value of e is assigned to x. Repetition of a statement S is denoted by $*[S]$.

2.3. 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 – causing different DAEs to be

selected.

By means of the nabla operator (∇) used in the discrete-event part, a process can synchronize with a state event. Execution of $\nabla\, b$, where b is a boolean expression involving at least one or more *continuous* variables, causes the process to be blocked until b becomes true.

2.4. Data types

Some data types are predefined like real, nat (positive integers) and bool (boolean). Data types can be postfixed with units 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. 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].

3. AN EXAMPLE

This section illustrates the use of the language χ by an example of a transport system shown in Figure 1. The system consists of a line of conveyor belts driven by motors. The system is used for the transportation of trays. Only the model of the physical system is presented. The discrete-event control system has been specified in χ, but is not treated in this paper.

3.1. The model of a motor

The system uses a process M to model the conveyor motors. The motor is switched on when it receives the value 1 via channel m and is switched off when it receives the value 0. The parameter v_{set} represents the value of the velocity of the conveyor motor when it is switched on. The velocity v of process M is made available to three processes via the continuous ports v_1, v_2 and v_3 (see Figure 2).

$$\textsf{proc}\ \ M(v_1, v_2, v_3 : \uparrow [\mathrm{m/s}],\ m : ?\,\mathrm{nat},\ v_{set} : \mathrm{real}[\mathrm{m/s}]) =$$
$$[\![\ v : [\mathrm{m/s}],\ n : \mathrm{nat} \mid v \multimap v_1,\ v \multimap v_2,\ v \multimap v_3 \mid \{v = v_{set} \cdot n\} \mid n := 0\,;\ *[m?n]\ ]\!]$$

3.2. The model of a conveyor

The assumptions regarding the behaviour of a conveyor are:

- The length of the trays lies in between half of the length ($l/2$) and the length (l) of a conveyor.
- If a tray is positioned on two conveyors, the movement of the tray is determined by the conveyor which supports the major part of the tray.
- Trays do not come into contact with each other.
- Initially there are no trays on the conveyors.

A conveyor is modelled by two processes: the first half of the conveyor by *CF* and the second half by *CB*. Each tray is modelled by a positive integer which is stored in the variable *id*. The variables x, v and v_p represent the position of the front of a tray, the velocity of the conveyor

belt and the velocity of the preceding conveyor belt, respectively. The variable p represents the presence of a tray within the process, i.e. $p =$ true when a tray is present. The variable q equals true when the conveyor supports the major part of the tray, and false otherwise.

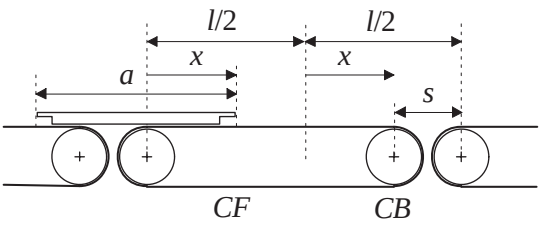

Figure 1. The conveyors.

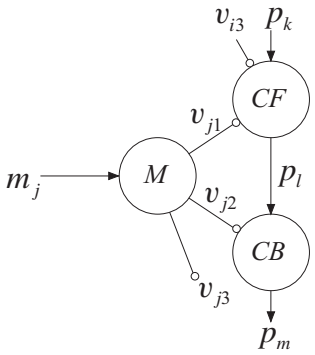

Figure 2. Model of a conveyor.

The specification of the process *CF* is shown below. On line (2) of this specification the entry of a tray on the conveyor is modelled by the communication $p_k?id$. As a result x is initialized at 0 and p becomes true. The trajectory of x is determined by the value of its derivative. On line (1) this value equals either the velocity v_p of the preceding conveyor (if $p \wedge \neg q$) or the velocity v of the conveyor itself (if $p \wedge q$), depending on the position of the tray. If no tray is present ($\neg p$) the derivative is set to zero. In the discrete-event section, two state-events are used. The first one for detection that the major part of the tray is supported by the conveyor, i.e. $\nabla x \geq a/2 - s/2$. By assigning the value true to q, the DAE $x' = v$ is selected. The second event represents the moment at which the tray reaches the process boundaries, i.e. $\nabla x \geq l/2$. Consequently the communication $p_l!id$ is executed. From this point on, the position of the tray is determined by the next process.

```
proc CF (v_i3, v_j1 : ↓[m/s], p_k : ?nat, p_l : !nat, l, a, s : real[m]) =
⟦[ id : nat, x : [m], v, v_p : [m/s], p, q : bool
| v_p ⊸ v_i3, v ⊸ v_j1
| { [ p ∧ ¬q ⟶ x' = v_p | p ∧ q ⟶ x' = v | ¬p ⟶ x' = 0 ]}                (1)
| p := false; q := false;
  *[ p_k?id; x := 0; p := true; ∇x ≥ a/2 − s/2; q := true                  (2)
   ; ∇x ≥ l/2; p := false; q := false; p_l!id
   ]
⟧
```

Process *CB* contains the specification of the second half of the conveyor. Its specification is slightly different from the one of process *CF*, because a tray the position of which is determined by process *CB* is always driven by its own conveyor. This is reflected by the absence of the variable q used in the specification of process *CF*.

```
proc CB (v_j2 : ↓[m/s], p_l : ?nat, p_m : !nat, l, a : real[m]) =
⟦[ id : nat, x : [m], v : [m/s], p : bool
| v ⊸ v_j2
| { [ p ⟶ x' = v | ¬p ⟶ x' = 0 ]}
```

| $p :=$ false ; $*[\ p_l?id\ ;\ x := 0\ ;\ p :=$ true ; $\nabla x \geq l/2\ ;\ p :=$ false ; $p_m!id\ ]$
]‖

3.3. The system

The model of the transport system T is given below. The trays enter the system at process *CFF*. Because this process is not preceded by a conveyor belt, its specification differs from the other *CF* processes. The differences are minimal and therefore the specification of *CFF* is omitted. The specification of the last process *CBL* is not given for a similar reason. The system T communicates with its environment via the discrete channels p_1 and p_7 – via which trays enter and leave the system – and via the discrete channels m_1, m_2 and m_3 which communicate with the actuators of the motors. A graphical representation of the structure of the system is given in Figure 3.

syst $T\ (p_1, m_1, m_2, m_3 :?\,\text{nat},\ p_7 :!\,\text{nat},\ l, a, s : \text{real[m]},\ v_{set} : \text{real[m/s]}) =$
‖[$p_2, p_3, p_4, p_5, p_6 :$ nat, $v_{11}, v_{12}, v_{13}, v_{21}, v_{22}, v_{23}, v_{31}, v_{32} :$ [m/s]
| $CFF(v_{11}, p_1, p_2, l, a, s) \parallel CB(v_{12}, p_2, p_3, l, a) \parallel CF(v_{13}, v_{21}, p_3, p_4, l, a, s)$
$\parallel CB(v_{22}, p_4, p_5, l, a) \parallel CF(v_{23}, v_{31}, p_5, p_6, l, a, s) \parallel CBL(v_{32}, p_6, p_7, l, a)$
$\parallel M(v_{11}, v_{12}, v_{13}, m_1, v_{set}) \parallel M(v_{21}, v_{22}, v_{23}, m_2, v_{set}) \parallel M(v_{31}, v_{32}, m_3, v_{set})$
]‖

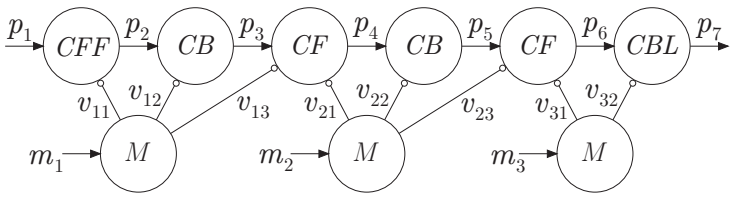

Figure 3. Model of the transport system.

4. CONCLUSIONS

The application of the language χ to modelling and simulation of manufacturing machines has been illustrated using a simple conveyor based transport system. The integration of continuous-time and discrete-event concepts in the language make it highly suitable for machine modelling. In the continuous-time parts of the models, positions of moving parts are described by simple DAEs. In order to simplify models, many actions of machines can be abstracted as discrete-event actions. For this purpose, the language χ provides CSP-based discrete-event constructs. Changes in the velocity of moving parts can be simplified as discontinuous changes, which are modelled in the discrete-event part of the processes. The crossing of process boundaries by products is modelled by synchronous communications. Interaction between the continuous-time and discrete-event parts of processes is achieved by means of assignments to variables occurring in DAEs or by means of the nabla operator in order to synchronize with state events.

The language is relatively simple because of the small number of orthogonal language constructs. Combined with the symbolic mathematical notation, this leads to concise models. Good modularity is achieved by the use of processes and systems and the absence of global variables.

The language is also being used with success for the specification of control systems. Such a control model can be tested by connecting it to a model of the controlled machines. The combined system can then be simulated. In this way, the combined continuous-time / discrete-event approach to modelling and simulation of manufacturing machines can be an important aid for the development of machine control systems.

REFERENCES

1. E.E.L. Mitchell and J.S. Gauthier, Advanced Continuous Simulation Language (ACSL). *Simulation 26*, No.3, 1976, pp. 72-78.
2. M. Andersson, Combined Object-Oriented Modelling in Omola, *Proceedings of the 1992 European Simulation Multiconference,* York, U.K., June 1-3, The Society for Computer Simulation International, pp. 246-250.
3. N.W.A. Arends, J.M. van de Mortel-Fronczak, and J.E. Rooda, Continuous systems specification language, *CISS - First joint conference of international simulation societies proceedings*, Zurich, Switzerland, August 1994, pp. 76-79.
4. C.A.R. Hoare, *Communicating Sequential Processes*, Communications of the ACM, August 1978, pp. 666-677.
5. J.M. van de Mortel-Fronczak and J.E. Rooda, Application of concurrent programming to specification of industrial systems, Accepted for INCOM'95.
6. J.M. van de Mortel-Fronczak, J.E. Rooda, and L.A.J. de Greeff, Real-time concurrent programming as a structured approach to modelling of manufacturing systems, Accepted for FAIM'95.