LANGUAGES AND APPLICATIONS IN HYBRID MODELLING AND SIMULATION: POSITIONING OF CHI

D.A. van Beek [1], **J.E. Rooda**

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

Abstract: A widely used classification of modelling languages distinguishes the categories continuous-time (CT), discrete-event (DE), discrete-time (DT), and hybrid. For a better insight into the many different hybrid languages, a classification of five categories (CT, CT+, DE, DE+, and CT/DE) is proposed. Each category is explained, together with many of the included languages, simulators, and the associated application fields. Special interest is given to the Chi language used for specification, simulation and real-time control of industrial systems. Its CT part is based on (conditional) DAEs, its DE part on Communicating Sequential Processes. The suitability of the language for DE, CT, and CT/DE modelling is illustrated by two cases.

Keywords: Modelling; simulation languages; continuous-time systems; discrete-event systems; discrete-event dynamic systems

1. INTRODUCTION

A widely used classification of models and modelling languages is based on the relation between state changes and the progress of time in models. In this respect, three categories can be distinguished: continuous-time (CT), discrete-event (DE), and discrete-time (DT). In continuous-time models, an infinite number of state changes occurs in any given finite time interval, whereas there are no state changes at discrete time points. In discrete-event models, the state changes at discrete time points only; in between two adjacent discrete time points the state remains the same. In discrete-time models, the state changes at equidistant time points only. Therefore, discrete-time models can be regarded as a subset of discrete-event models. For this reason, the two concepts CT and DE are sufficient. By combination of CT and DE concepts, hybrid models and hybrid modelling languages are obtained.

A classification of CT, DE, and CT/DE models is sufficient to distinguish model differences with respect to their time-dependent behaviour. A similar classification of modelling languages, however, gives little insight into the diversity (e.g. possibilities and restrictions) of the many different hybrid modelling languages. Therefore, a more precise classification is proposed that entails the following five categories: CT, CT+, DE, DE+, and CT/DE. In the following section, the five categories are discussed, and a number of concrete hybrid languages are categorized. Only those hybrid languages for which there is a simulator available are discussed. No attempt is made to be complete in this respect. Emphasis is on general purpose languages. Only the current state of the field is described, no attempt is made to describe the history of hybrid languages. The last restriction is that only those languages that use high level language elements in at least one domain (CT or DE) are considered. High level language elements enable the modeller to specify systems in an intuitive way, independently of the implementation details of the language and solving algorithms used. As a consequence of the last restriction, languages based on a general purpose programming language such as Fortran (e.g. GASP, Pritsker, 1974 or SLAM II, Pritsker, 1986), C (e.g. Smile, 1999) or C++ (Bujakiewicz & Van den Bosch, 1991) are not considered. The relation between the application field of a hybrid modelling language and the choice of the hybrid language elements is also discussed. The hybrid χ (Chi) language is classified and explained in more detail, together

[1] Corresponding author. E-mail: d.a.v.beek@tue.nl.

with two illustrative examples of the wide application field of the language.

2. HYBRID LANGUAGES, SIMULATORS AND APPLICATIONS

The differences between CT, CT+, DE, DE+, and CT/DE languages are discussed below. CT languages are suited to CT model specification only. DE languages are suited to DE model specification only. Hybrid languages are divided into three categories: CT+, DE+, and CT/DE.

2.1 *CT+ languages*

CT+ languages are CT languages that are extended with DE language elements. The languages are not designed for the specification of pure DE models. Most of the hybrid languages that provide a simulator belong in this category. The languages are used for modelling physical systems by means of mathematical equations. The discrete-event additions enable modelling of discontinuities or discrete control actions. The facilities for DE modelling and simulation vary considerably among the languages in this category. Two sub classifications in the CT+ category are CT + discontinuity modelling and CT + operating procedure modelling. These two categories are elaborated below. More insight in the required functionality of hybrid simulators is given by Mosterman (1999), who compares the functionality of ten CT+ simulators and two CT/DE simulators.

2.1.1. *CT + discontinuity modelling* Different DE language elements for discontinuity specification may be provided, such as conditional equations (see Section 3.1), discontinuous functions, or language elements for time-event and/or state-event specification. A time-event is an event that occurs at a specified time point. A state-event is an event that occurs when a continuous variable, or an expression involving at least one continuous variable, crosses a certain threshold value. An important application for such languages is in the modelling and simulation of control systems for continuous systems with discontinuities. Other applications are found in avionics, robotics, and modelling of electrical networks and multi-body systems. Languages in this category are discussed below.

ACSL (Mitchell & Gauthier, 1976) is a hybrid language with a FORTRAN like syntax that follows the Continuous System Simulation Languages (CSSL) standard developed in 1967. Dymola (Elmqvist, 1994) comes with many ODE/DAE (Ordinary Differential Equations/Differential Algebraic Equations) solvers and makes extensive use of symbolic manipulation for solving and simplifying (non-causal) equations. Non-causal equations have the advantage that input and output variables are no longer necessary in models. The added DE language elements are mainly used for handling discontinuities in otherwise continuous systems. A simulator for the Modelica (Mattsson, Elmqvist & Otter, 1998) language is included with the newest version of the Dymola simulator. Discrete actions in Dymola and Modelica are modelled in an equation like manner. By means of Place and Transition library models in Modelica, a certain class of Petri nets can be integrated in the Modelica hybrid models (Mosterman, Otter & Elmqvist, 1998b). Neptunix (Neptunix, 1999) is a commercially available simulator that was first developed in 1980. The object oriented Omola language with the OmSim simulator (Andersson, 1994) from the Lund Institute of Technology was originally developed with the aim of providing better Computer Aided Control Engineering tools. HyBrSim (Mosterman, Biswas & Otter, 1998a) is an experimental hybrid bond graph modeling and simulation tool based on physical principles and developed at the DLR Oberpfaffenhofen. Simulink/Stateflow (Simulink/Stateflow, 1999) is a causal, block diagram based modelling and simulation environment for the development of control systems. Handling of discrete-events in Simulink/Stateflow differs from the other packages, since the Simulink simulator may switch to a continuous phase before all possible discrete events have been handled (Mosterman, 1999). Finally, 20-sim (Broenink, 1998) is a modelling and simulation tool developed at the University of Twente. Models can be bond graph based, block diagram based, or equation based.

2.1.2. *CT + operating procedure modelling* Operating procedures refer to the control actions required to operate plants of the so called ‘process type’, such as a chemical plant. Such control actions (e.g. switching valves on or off) can be prescribed by batch recipes (see Section 2.5). Using a CT+ language, the physical system is modelled as a CT system with discontinuities. The DE language elements are used to model the operating procedures.

A language in this category is gPROMS (Barton, 1992), which is used for chemical process modelling. Its CT part allows the non-causal specification of implicit DAEs, that have the general form of $\mathbf{f}(\mathbf{x}', \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$. The simulator contains a very efficient and optimized index 1 DAE solver, that can handle very large sets of equations. A nonlinear solver for calculating the initial state and the state after a discontinuity is also included. Modularity of continuous components is provided by means of streams. There is no comparable mechanism for DE components, since the DE model components are designed to interact with CT model components only, not with other DE components. A language that has been derived from gPROMS is ABACUSS (Advanced Batch and Continuous Unsteady-State Simulation). It is developed at the Massachusetts Institute of Technology (Abacuss, 1995).

2.2 *DE+ languages*

DE+ languages are DE languages that are extended with elementary language elements for modelling continuous systems. The languages are not designed for the specification of pure CT models. An application field of such languages is in the modelling of certain batch plants, where the scheduling of different products that are produced and/or stored in different tanks is important. The main continuous aspects of such models would be constant (on/off) flows from one vessel to another. Examples of such plants are beer breweries and fruit juice packing plants. Operations like mixing, stirring and brewing can be modelled as delay statements (see Section 3.2).

The data flow based modelling language SIMAN (Pegden, Shannon & Sadowski, 1995) is an example of a DE+ language. Because of the low level CT elements in this language, the user must code the equations (ODEs or explicit DAEs of index 0) into a C programming language function. The user must also specify the 'equations' in the right order. Currently, the SIMAN modelling language and Cinema visualization environment have been combined in the commercial product Arena. Another commercially available DE+ simulation environment is Simple++ (Simple++, 1999). It contains some special purpose predefined hybrid objects, such as buffer tanks and conveyors. The Personal Prosim (Sierenberg & De Gans, 1992) language is a process based language, where data at the top level is global. The equations are of the ODE type, including DAEs of index 0, but without conditional equations.

2.3 *CT/DE languages*

CT/DE languages are equipped with high level language elements in both the CT and DE domain, and can be used for the specification of pure CT models as well as pure DE models. This makes it possible to model one part of the physical processes in a plant as a CT sub model, and another part as a DE sub model. The preparation of ice-cream, for example, could be described as a CT or as a batch process, but the subsequent packing of the individual ice-cornets could be described as a DE process. Such DE modelling requires high level data structures such as lists or queues, arrays, and tuples or records. High level CT elements should include conditional differential equations, where the equation chosen depends on a boolean condition. Languages in this category are treated below.

The CT part of COSMOS (Kettenis, 1994) allows the specification of ODEs and explicit DAEs of index 0. The DE part is based on the process-interaction world view. Processes are explicitly activated and deactivated. Shift (Deshpande, Göllü & Semenzato, 1998) is a modelling language for describing dynamic networks of hybrid automata that can be created, interconnected and destroyed as the system evolves. The CT part of Shift is based on ODEs that are solved with a fixed step size solver. The discrete-event parts of sub models can interact with one another by means of synchronous events. VHDL (Very high speed integrated circuit Hardware Description Language) is a language for the specification of integrated circuits. In 1999, the VHDL super set called VHDL 1076.1 (VHDL-AMS, 1999), or informally VHDL-AMS, was first released. The AMS (Analog & Mixed-Signal) analog extensions enable modelling of continuous systems by means of DAEs. Combined with the concurrently executing processes from the original VHDL language, this results in a CT/DE language. Although the language has been designed as a special purpose language, VHDL-AMS has a sufficient number of general purpose language elements to be used for other purposes than integrated circuit hardware specification. SEAMS (SEAMS, 1999) is a hybrid simulator for VHDL-AMS. The χ (or Chi) language is developed at the Systems Engineering Group of the Eindhoven University of Technology. The CT part of the language allows non-causal specification of implicit DAEs up to index 1. Apart from a DAE solver, the simulator also contains a nonlinear equation solver for calculating the initial state and the state after discontinuities. The DE part is based on CSP (Communicating Sequential Processes) (Hoare, 1985).

2.4 *Languages for formal verification*

The main purpose of the languages treated so far is to help understand the dynamic behaviour of systems by means of modelling and simulation. Another category of languages is mainly used for the *analytical* derivation of model properties, such as proving the absence of dead-lock, or proving that certain states will be or cannot be reached (e.g. Alur, 1995). These languages are usually based on a finite state automaton, where each state can be associated with a set of ODEs. Petri nets are also used for the analytical derivation of model properties. Many different hybrid Petri nets have been proposed. Several of these can be found in the Petri net survey of (David & Alla, 1994). Many of the hybrid Petri nets are flow nets, that incorporate extensions to the usual Petri nets to enable the modelling of flows. A more generally applicable approach is taken by allowing DAEs to be associated with each place in a Petri net (Champagnat, Esteban, Pingaud & Valette, 1998). The DAEs associated with a place are activated if the place contains one or more tokens. The continuous variables in this net are global. Structured data types, modularity and reuse of components are generally not provided for in hybrid Petri nets. First versions of simulators for hybrid Petri nets have been described by Champagnat, Esteban, Pingaud & Valette (1998), and Valentin-Roubinet (1998).

2.5 *Special purpose languages*

An example of a special purpose package is BaSiP (Fritz, Preuss & Engell, 1998). It provides a graphical interface for modelling batch processing plants that are controlled using batch recipes. An example of such a plant is a beer brewery. The batch recipes would then specify for each product what tanks to use, in what order, for how long, and under what conditions. For simulation, BaSiP has an interface to the gPROMS (Barton, 1992) simulator. In cases when the discrete events can be predetermined, it provides for discrete-event simulation. A hybrid language for electrical network simulation is Simplorer (Simplorer, 1999). It also has some general purpose capabilities. In (Hohmann & Zanne, 1998) it is compared with seven other hybrid simulation languages. The DOORS (Kasper & Koch, 1995) project aims to develop a distributed real-time simulator for mechatronic system design.

3. THE χ LANGUAGE

The χ language has been designed from the start as a hybrid language that can be used for the specification, verification, simulation and real-time control of discrete-event systems, continuous-time systems and combined discrete-event / continuous-time (hybrid) systems. It has been successfully applied to a large number of industrial cases, such as an integrated circuit manufacturing plant (Rulkens, Van Campen, Van Herk & Rooda, 1998) and a beer brewery.

The language is based on mathematical concepts with well defined semantics (meaning of the language elements). It is easy to learn and to use, because it is based on a small number of orthogonal language elements. The symbolic notation of χ models improves the readability (Van Beek & Rooda, 1997b). For simulation purposes, the symbols are replaced by their ASCII equivalents.

The continuous-time part of χ is based on differential algebraic equations (DAEs). The discrete-event part of χ (e.g. Van de Mortel-Fronczak & Rooda, 1996, Van de Mortel-Fronczak, Rooda & Van den Nieuwelaar, 1995) is based on CSP (Communicating Sequential Processes) (Hoare, 1985). Where possible, the continuous-time and discrete-event parts of the language are based on similar concepts. Processes are parametrized and can be grouped into systems, which enables the construction of hierarchical models. Discrete and continuous channels are used for inter-process communication and synchronisation.

In this paper, only a part of the language is treated. The syntax and operational semantics of the language elements are explained in an informal way. The first version of the χ language was presented, together with design considerations, by Arends (1996). Recent developments of the hybrid language elements and simulator are covered by Fábián (1999).

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*) $=$
$\|[$ *process and system declarations*
, *variable and channel declarations*
| *process and 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: differential algebraic equations), a discrete-event part only, or a combination of both.

proc *name*(*parameter declarations*) $=$
$\|[$ *variable declarations* ; *initialization* | *links*
| *DAEs* | *discrete-event statements*
$]\|$

Processes have local variables only; all interactions between processes take place by means of channels. A channel connects two processes or systems. It is declared in systems using the $-$ symbol followed by the type of data that is carried via the channel. A channel is either a discrete communication channel (e.g. $p : -\text{int}$), a discrete synchronization channel (e.g. $s : -\text{void}$) or a continuous channel (e.g. $Q :: -[\text{m}^3/\text{s}]$). Discrete channels are declared using a single colon (:), whereas continuous channels are declared using a double colon (::). The special void data type indicates the absence of data. If a channel is declared as a process or system parameter, the usage of the channel is declared as either discrete output (e.g. $p : !\,\text{int}$), discrete input (e.g. $p : ?\,\text{int}$), synchronization without direction (e.g. $s : \sim \text{void}$), or continuous without direction (e.g. $Q :: \multimap [\text{m}^3/\text{s}]$). A continuous channel is represented graphically by a line (optionally ending in a small circle to indicate either the direction of a flow, or a cause-effect relationship), a communication channel by an arrow, and a synchronization channel by a dotted line (optionally ending in an arrow head to indicate a cause-effect relationship). Processes and systems are represented by circles.

All data types and variables are declared as either continuous using a double colon (e.g. $V :: [\text{m}^3]$), or discrete using a single colon (e.g. $n : \text{int}$). The value of a discrete variable is determined by assignments (e.g. $n := 1$). 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 (e.g. initialization $V ::= 0$) determines its value for the current point of time only.

Some discrete data types are predefined like bool (boolean), int (integer) and real. Variables may be declared with units (e.g. $v :: [\text{m}/\text{s}]$). All variables with units are of type real.

Structured types are defined by means of array, set, list and tuple types. Tuples are comparable to records. A list (queue) *xs* is declared as $xs : T^*$. This defines a list of elements of type T. The first element (head) is obtained by $\text{hd}(xs)$, the list without the first element (tail) by $\text{tl}(xs)$. Initialization may be done by $xs := [x_1, \ldots, x_n]$, where x_i is an expression of type T.

3.1 *The continuous-time part of χ*

A time derivative is indicated by a prime character (e.g. x'). DAEs are separated by commas: $DAE_1, DAE_2, \ldots, DAE_n$. A DAE can be a normal equation or a *guarded equation*, also referred to as a conditional equation. The latter is used when the set of equations depends on the state of the system. The syntax is $[\, b_1 \longrightarrow DAEs_1 \,[]\, \ldots \,[]\, b_n \longrightarrow DAEs_n \,]$, where $DAEs_i$ $(1 \le i \le n)$ represents one or more DAEs. The boolean expression b_i represents a *guard.* At any time, at least one of the guards must be open (true), so that the DAEs $DAEs_i$ 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 continuous channel with a continuous 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.

3.2 *The discrete-event part of χ*

Interaction between discrete-event parts of processes takes place only by means of synchronous communication ($c \,!\, e$ or $c \,?\, x$) or by synchronization ($s \,^{\sim}$). Consider channel c (or s) connecting two processes. Execution of $c \,!\, e$ or $c \,?\, x$ in one process causes the process to be blocked until $c \,?\, x$ or $c \,!\, e$ is executed in the other process, respectively. Subsequently the value of expression e is assigned to variable x. Execution of $c \,^{\sim}$ in one process causes the process to be blocked until $s \,^{\sim}$ is executed in the other process.

Time passing is specified by delay statement $\Delta\, t$, where t is an expression of type real. A process executing this statement is blocked until the time is increased by t time-units.

Selection ([GB]) is specified by [$b_1 \longrightarrow S_1$ [] $b_2 \longrightarrow S_2$ [] ... [] $b_n \longrightarrow S_n$]. The boolean expression b_i ($1 \le i \le n$) represents a *guard*, which is open if b_i evaluates as true and is closed otherwise. 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 specified by [b_1; $E_1 \longrightarrow S_1$ [] ... [] b_n; $E_n \longrightarrow S_n$]. There are five types of event statement E_i: output ($c \,!\, e$), input ($c \,?\, x$), synchronization ($c \,^{\sim}$), time ($\Delta\, t$), and state ($\nabla\, r$, explained in the following section). An event statement E_i which is prefixed by a guard b_i (b_i; E_i) is enabled if the guard is open and the event specified in E_i can actually take place. A time event statement $\Delta\, t$ can take place when t time units have passed. 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. 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 statement [GB] or [GW] is specified by $*$[GB] or $*$[GW], respectively. In this case, it is not necessary for at least one of the guards to be open. The repetition terminates when all guards are closed. The repetition $*$[true $\longrightarrow S$] may be abbreviated to $*$[S].

3.3 *Continuous / discrete interaction*

In the discrete-event part of a process, assignments can be made to discrete variables occurring in DAEs (e.g. $a := 0$ in process Fill, see Section 4.1) or in the boolean guards of guarded DAEs (e.g. $s :=$ "stop" in process Friction$_1$, see Section 4.2). 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 $::=$ is used.

By means of the *state event* statement $\nabla\, r$, the discrete-event part of a process can synchronize with the continuous part of a process. Execution of $\nabla\, r$, where r is a relation involving at least one continuous variable, causes the process to be blocked until the relation becomes true.

3.4 *The χ simulator*

Development of the χ simulator has so far taken place along two paths. The discrete-event χ simulator has been described by Naumoski & Alberts (1998), the hybrid χ simulator has been described by Fábián (1999), and Fábián, Van Beek & Rooda (1998). The syntax and semantics of the χ language has also changed considerably since the first versions of the χ simulators. Currently, work is being carried out to integrate the discrete-event and hybrid χ simulators based on the newest syntax and semantics.

The Chi models in this paper are specified in a symbolic notation. For simulation purposes, the symbols are replaced by their ASCII equivalents. A conversion table is presented by Van Beek & Rooda (1999). Conversion is straightforward. For example, the symbols $\longrightarrow$, Δ, ∇, $[\![$, $\neq$, x^2 are converted to `->`, `delta`, `nabla`, `|[`, `/=`, `x^2`, respectively.

An ASCII χ model is entered in a file. This file can be compiled by the χ compiler by entering a command in a command window. The χ compiler then translates the model into a C++ program, that is subsequently linked to the χ kernel. The result is an executable file. Executing this file runs the simulation model. The simulation can read input data from ASCII input files and writes output data into ASCII output files. The resultant ASCII output data can be analyzed using any kind of data analysis software. The χ simulator is available for Linux and for MS-Windows.

3.5 *Formal verification of χ models*

To date, properties of the dynamic behaviour of industrial production processes are mainly determined by means of simulation. Simulation, however, cannot be used to prove the correctness of a model. In order to be able to prove a χ model to be correct with respect to certain specifications, a joint Ph.D. project has been set up between the Systems Engineering Group of the Mechanical Engineering Department and the Formal Methods Group of the Computer Science Department of the Eindhoven University of Technology. The project focuses on formal analysis of discrete-event χ models. In future, a similar project will be started for verification of hybrid χ models.

Correctness of a χ model of a production system consisting of four processes has been proved by hand (Kleijn, Reniers & Rooda, 1998). The system was first simulated in χ. Subsequently, the χ model was translated into a process algebra model for which correctness was proved. In order for the proof to be sufficient, however, the translation from the χ model to the process algebra model must also be proved correct. To avoid this intermediate step, the project now focuses on direct formal reasoning in χ in order to enable the use of model checkers and interactive theorem provers. As a first step, the language semantics of the existing χ language has been formally defined (Bos & Kleijn, 1999). As a result of insights gained from this project, the language semantics has subsequently been improved in several aspects.

4. CASES

Two cases that illustrate different aspects of hybrid modelling are discussed. The first case is an example of a plant where continuous products (liquids) and discrete products (trucks and barrels) are handled. The flow of the liquids is modelled by means of equations, the transportation of the trucks and barrels is modelled by means of discrete statements, communication and synchronisation. This example also deals with stochastic phenomena, since the trucks arrive at irregular intervals. In this case, emphasis is on the discrete-event aspects. The second case is an example of a physical process with discontinuities; it deals with modelling of dry friction. The model consists of one process only. The model can be in three states. In each state, different equations are valid. The discrete-event part of the model switches from one state to another. In this case, emphasis is on continuous-time aspects.

4.1 *Filling station*

A filling station supplies liquid that is stored in a buffer tank of 2 m^3. From this buffer tank, 200 liter barrels can be filled with a flow rate of 1 liter/s. Trucks with empty barrels arrive at the filling station. The time between the arrival of two subsequent trucks is given by a negative exponential distribution with an average of 0.5 hours. Each truck carries exactly one barrel. The trucks line up to await their turn before the barrel filling station. The buffer tank is supplied with liquid from a processing plant. The flow is switched on when the volume of the buffer tank falls below 1.8 m^3, and is switched off when the tank is full. When the flow is switched on, the flow rate equals 0.5 m^3/hour. The time required for connecting or disconnecting a barrel to the filling station is 1 minute. Initially, the buffer tank is empty. Filling of an empty barrel does not start until there is at least 200 liters available in the buffer tank.

By means of a model, the dynamics of the filling station can be analyzed with respect to the number of trucks waiting to be filled and their waiting time. The waiting time of a truck is defined as the period of time from its arrival until the barrel starts to be filled. In this simplified example, no specific control strategies are required. The trucks are served on a 'first come first served' (FCFS) basis, so that the truck that arrives first is served first. If, however, the plant produced different kinds of liquid, and the number of barrels on a truck that needed to be filled could vary, then different control strategies would be useful. The trucks could for instance be served on a shortest processing time first (SPT) basis. This would mean that if there is more than one truck waiting to be served at the filling station, the truck to be served first is the truck that carries the smallest number of barrels (assuming that the filling times of all barrels are equal). By means of simulation, the different control strategies can then be compared. The model of the filling station is discussed below.

Below, the types that are used in the model are given. All times in the model are specified in hours.

```
type  hours = real
,     vol   = [m^3]
,     flow  = [m^3/hr]
```

Fig. 1 shows the filling station model structure. The formal textual definition follows below.

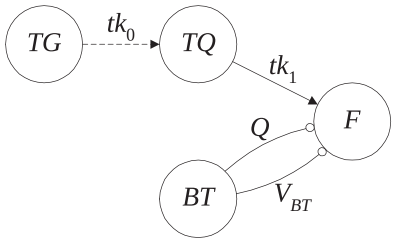

Fig. 1. System FillingStation.

```
syst FillingStation =
⟦ $TG$ : TruckGenerator, $TQ$ : TruckQueue
, $BT$ : BufferTank, $F$ : Fill
, $Q$ :: −flow, $V_{BT}$ :: −vol
, $trk_0$ : −void, $trk_1$ : −hours
| $TG(trk_0)$ ∥ $TQ(trk_0, trk_1)$
∥ $BT(Q, V_{BT}, 0.5)$ ∥ $F(Q, V_{BT}, trk_1, 0.2, 3.6)$
⟧
```

Process declaration TG : TruckGenerator declares a process variable TG of type TruckGenerator. The processes are instantiated (created with their actual parameters) after the | separator symbol. In this model, each process type is instantiated only once; there is one process variable for each process type. In larger models, there are often many process instantiations of the same type. There could for instance be two or more processes of type BufferTank.

The process definition of TruckGenerator follows below.

```
proc TruckGenerator($trk_0$ : ~ void) =
⟦ $d$ : → hours, $dt$ : hours
| $d$ := nex(0.5)
; ∗[ Δsample $d$ ; $trk_0$ ~ ]
⟧
```

Declaration d : → hours declares a distribution d that returns values of type hours. Distributions are used to model stochastic phenomena, where the outcome of an experiment cannot be predicted exactly. Only, the *probability* that the outcome is a certain value, or lies in a certain interval, is known. An example is the result of throwing a dice. Distribution d is initialized to the negative exponential distribution with an average value of 0.5 (hours) in statement d := nex(0.5). The negative exponential distribution is often used to model the period of time between the arrival of two subsequent clients in a queue, or the time between two subsequent failures of a device. The only possible operation on a distribution is the sample operation (e.g. sample d). The arrival of the trucks at the filling station is modelled by first waiting for the next truck to arrive (Δsample d). The delta statement Δ0.34 for instance, waits until the simulation time is increased with 0.34 time units (hours in this case). The result of sample d is a 'random' value in hours, such that the average of all subsequent samples converges to 0.5 in the end. After the required time has passed in Δ sample d, the arrival of the truck is modelled by synchronizing with process *TQ* of type TruckQueue via channel trk_0 ($trk_0\,^{\sim}$). After this synchronization, the loop $*[\ \ldots\ ]$ is re-executed, beginning with delta statement Δsample d.

Process *TQ* of type TruckQueue models a 'first come first served' (FCFS), also called 'first in first out' (FIFO), buffer.

```
proc TruckQueue($trk_0 : {}^{\sim}\mathrm{void},\ trk_1 : !\,\mathrm{hours}$) =
$[\![\ nc :: [-],\ xs : \mathrm{hours}^*$
$|\ nc = \mathrm{len}(xs)$
$|\ xs := [\,]$
$;\ *[\qquad\qquad\qquad trk_0\,^{\sim} \qquad\quad \longrightarrow xs := xs +\!\!+ [\tau]$
   $[]\ \mathrm{len}(xs) > 0;\ trk_1\,!\,\mathrm{hd}(xs) \longrightarrow xs := \mathrm{tl}(xs)$
   $]$
$]\!]$
```

The trucks that enter the buffer are stored in list (or queue) *xs*. A list is an ordered collection of elements of the same type. For example [0, 7, 7, 2] is a list containing the numbers 0, 7, 7, and 2, in that order. Operator $+\!\!+$ concatenates two lists. For example, $[0] +\!\!+ [7, 7] = [0, 7, 7]$. Function hd (head) returns the first element of the list. For example, $\mathrm{hd}([0, 7, 7, 2]) = 0$. Function tl (tail) returns the list without the first element. For example, $\mathrm{tl}([0, 7, 7, 2]) = [7, 7, 2]$. Function len (length) returns the number of elements in the list. For example $\mathrm{len}([0, 7, 7]) = 3$. Declaration $xs : \mathrm{hours}^*$ declares a list of elements of type hours. The list is initialized as empty ($xs := [\ ]$). The buffer is modelled by means of a selective waiting statement ($[\ \ldots\ []\ \ldots\ ]$). At any time point, a truck may enter or leave the buffer. A truck entering the buffer is modelled by means of the synchronization $trk_0\,^{\sim}$. If this synchronization with process *TG* succeeds, the 'entering truck' is added to list *xs*. In this model, only the arrival time of the truck is actually added to the list ($xs := xs +\!\!+ [\tau]$). Since all trucks are the same, there is no other information about the trucks that needs to be stored. Symbol τ represents the current simulation time. A truck leaving the buffer is modelled by means of communication statement $trk_1\,!\,\mathrm{hd}(xs)$, where $\mathrm{hd}(xs)$ is the first truck (in fact the arrival time of the first truck) in the list. Statement $trk_1\,!\,\mathrm{hd}(xs)$ can only be executed under the condition that the list *xs* is not empty ($\mathrm{len}(xs) > 0$). If the truck has been successfully sent to the filling process (process F of type Fill), the truck is removed from the list ($xs := \mathrm{tl}(xs)$). Continuous variable *nc* is used for data logging purposes only. The χ simulator logs the values of the continuous variables to an output file. In equation $nc = \mathrm{len}(xs)$, continuous variable *nc* is made equal to $\mathrm{len}(xs)$. In this way, the logged values of *nc* represent the number of trucks in the buffer. This data is used to create the graph that shows the number of trucks in the buffer as a function of time (see Figure 3). The equations are evaluated when all discrete actions at the current simulation time point have finished. If, for example, the first truck is generated in process *TG* at time 0.45, the truck enters the buffer ($trk_0\,^{\sim}$). However, immediately thereafter at the same time point, the truck leaves the buffer ($trk_1\,!\,\mathrm{hd}(xs)$). After that, the buffer process waits for a new truck to enter the buffer. This means that at the end of time point 0.45, the buffer is empty again, and equation $nc = \mathrm{len}(xs)$ evaluates to $nc = 0$. Therefore, the value of *nc* remains 0, and the temporary buffer size of 1 at time point 0.45 is not logged in the output file. This is exactly the desired behaviour.

Process *BT* of type BufferTank models the buffer tank. Continuous variables V and Q model the volume of the buffer tank, and the flow from the buffer tank to the filling process, respectively. Variable Q in process *BT* is equal to variable Q in process F (process type Fill). The value of Q is determined in process F by equation $Q = a \cdot Q_{\mathrm{set}}$. Equality of the two variables Q is achieved by means of continuous channel Q in system FillingStation (see the system definition and Fig. 1), and the links $Q\_ \multimap Q$ in processes *BT* and F. In a similar way, variable V in process *BT* equals variable V_{BT} in process F. Parameter Q_{set} represents the value of the flow into the buffer tank when it is being filled.

```
proc BufferTank( $Q\_ :: \multimap \mathrm{flow},\ V_{BT\_} :: \multimap \mathrm{vol}$
               , $Q_{\mathrm{set}} : \mathrm{real}$
               ) =
$[\![\ V :: \mathrm{vol},\ Q :: \mathrm{flow}$
$,\ Q_{\mathrm{i}} : \mathrm{real}$
$;\ V ::= 0;\ Q_{\mathrm{i}} := 0$
$|\ Q\_ \ \multimap Q$
$,\ V_{BT\_} \multimap V$
$|\ V' = Q_{\mathrm{i}} - Q$
$|\ *[\ Q_{\mathrm{i}} := Q_{\mathrm{set}};\ \nabla V > 2;\ Q_{\mathrm{i}} := 0;\ \nabla V < 1.8\ ]$
$]\!]$
```

Input flow Q_{i} is piece-wise constant. Therefore it is modelled by means of a discrete variable, instead of a continuous variable. Its value is changed by means of assignments ($Q_{\mathrm{i}} := 0$ and $Q_{\mathrm{i}} := Q_{\mathrm{set}}$). Equation $V' = Q_{\mathrm{i}} - Q$ is derived from the mass balance. This is possible because the incoming and outgoing liquids are incompressible and are of the same kind. The discrete-event part of the process consists of a loop. First, the incoming flow is switched on ($Q_{\mathrm{i}} := Q_{\mathrm{set}}$). In the next statement, the process waits until the tank is full ($\nabla V > 2$). When expression $V > 2$ becomes true, the incoming flow is switched off ($Q_{\mathrm{i}} := 0$). Subsequently, the process waits until the level of the tank drops below 1.8. When this happens, the loop is re-executed, and the incoming flow is switched on again.

The barrels are filled in process F of type Fill. The volume of the barrel to be filled is represented by continuous variable V. The pump is modelled by discrete variable a. This variable is of type nat (natural: positive integer). If $a = 0$, the pump is off, if $a = 1$, the pump is on (see equations $Q = a \cdot Q_{\mathrm{set}}$ and equation $V' = Q$). Discrete variables t_{arrive}, t_{sum}, tw_{avg}, and n_{sum} are used to determine the average waiting time of a truck.

```
proc Fill( $Q\_ :: \multimap \mathrm{flow},\ V_{BT\_} :: \multimap \mathrm{vol}$
         , $trk_1 : ?\,\mathrm{hours}$
         , $V_{\mathrm{barrel}}, Q_{\mathrm{set}} : \mathrm{real}$
         ) =
$[\![\ V, V_{BT} :: \mathrm{vol},\ Q :: \mathrm{flow},\ twc_{\mathrm{avg}} :: [-]$
$,\ t_{\mathrm{arrive}}, t_{\mathrm{sum}}, tw_{\mathrm{avg}} : \mathrm{hours},\ a, n_{\mathrm{sum}} : \mathrm{nat}$
$;\ V ::= 0;\ a := 0$
$|\ Q\_ \ \multimap Q$
$,\ V_{BT\_} \multimap V_{BT}$
|     $V' = Q$
,     $Q = a \cdot Q_{\mathrm{set}}$
, $twc_{\mathrm{avg}} = tw_{\mathrm{avg}}$
| $t_{\mathrm{sum}} := 0$; $n_{\mathrm{sum}} := 0$; $tw_{\mathrm{avg}} := 0$
; $*[$ $trk_1\ ?\ t_{\mathrm{arrive}}$
    ; $\Delta\, 1/60$; $\nabla\, V_{BT} \geq V_{\mathrm{barrel}}$
    ; $t_{\mathrm{sum}} := t_{\mathrm{sum}} + (\tau - t_{\mathrm{arrive}})$; $n_{\mathrm{sum}} := n_{\mathrm{sum}} + 1$
    ; $tw_{\mathrm{avg}} := t_{\mathrm{sum}}/n_{\mathrm{sum}}$
    ; $a := 1$; $\nabla\, V \geq V_{\mathrm{barrel}}$; $a := 0$; $V ::= 0$
    ; $\Delta\, 1/60$
    ]
]|
```

First, the process waits for the arrival of a truck in receive statement $trk_1\ ?\ t_{\mathrm{arrive}}$. The communication succeeds when there is a truck in buffer process TQ, and the buffer process executes the send action $trk_1\ !\ \mathrm{hd}(xs)$. As a result, the arrival time of the truck is stored in variable t_{arrive}. Next, process F waits for $1/60$ hours (1 minute), which models the time required for connecting the barrel to the filling station. Then, the process waits until there is sufficient liquid in the buffer tank to fill the barrel ($\nabla\, V_{BT} \geq V_{\mathrm{barrel}}$). After this, the average waiting time is calculated. The waiting time of the truck equals $\tau - t_{\mathrm{arrive}}$ (the current simulation time minus the arrival time of the truck at the buffer station). This time is added to the total waiting time of all trucks t_{sum}. The average waiting time is then set to the total waiting time divided by the total number of trucks that have waited ($tw_{\mathrm{avg}} := t_{\mathrm{sum}}/n_{\mathrm{sum}}$). Subsequently, the barrel filling flow is switched on ($a := 1$). When the barrel is filled ($\nabla V \geq V_{\mathrm{barrel}}$), the flow is switched off, and the volume V is reset tot 0 ($V ::= 0$), so that the next barrel can be filled. Disconnecting the barrel is modelled by delay statement $\Delta 1/60$. Continuous variable twc_{avg} and equation $twc_{\mathrm{avg}} = tw_{\mathrm{avg}}$ are used for data logging purposes only, in a way analogous to variable nc in process TruckQueue.

Figures 2 and 3 show the simulation results. In the last graph, flow Q shows the filling pattern of the barrels. The flow equals 3.6 [m^3/hr]. Since filling a barrel only takes 3 minutes approximately, the filling action results in a short vertical line in the graph. When the trucks arrive shortly after one another, they have to wait in the queue, and the volume of the buffer tank decreases. When the time between the arrival of the trucks increases, the queue (truck buffer) becomes empty, whereafter the volume of the buffer tank increases.

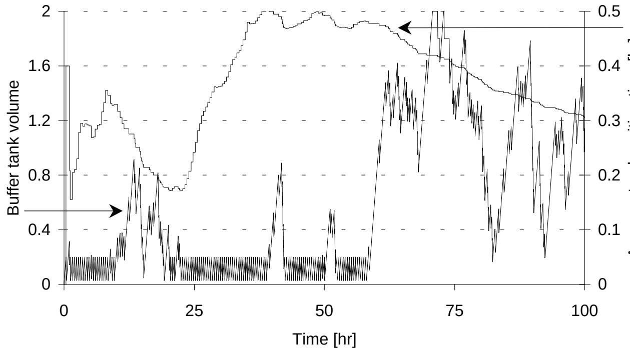

Fig. 2. Buffer tank volume V_{BT} [m^3], and average truck waiting time tw_{avg} [hr].

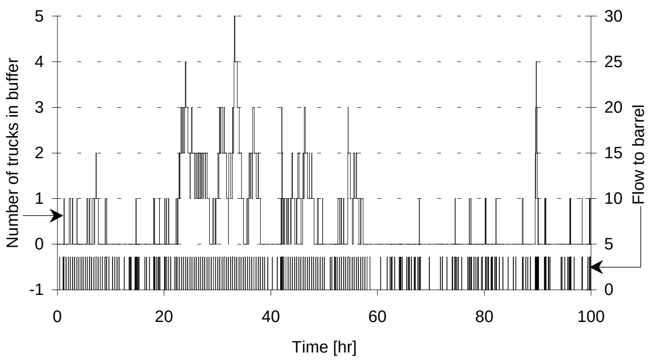

Fig. 3. Number of trucks in buffer n, and flow to barrel Q [m^3/hr].

The average waiting time of the trucks varies between 0.2 and 0.5 hours. Since this is relatively long, the volume of the buffer tank or the flow rate of the incoming flow should be increased.

4.2 *Dry friction*

Fig. 4 shows a body that can slide along a flat surface. A

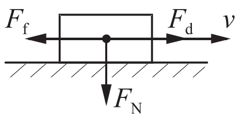

Fig. 4. Dry friction.

driving force F_d ($F_d = \sin 0.25\pi\tau$, where τ is the current time) is applied to the body. When the body is moving with positive velocity v, the frictional force is given by $F_f = \mu F_N$, where $F_N = mg$. When the velocity of the body is 0, the frictional force neutralizes the applied driving force. If the driving force becomes bigger than $\mu_0 F_N$, the body suddenly starts moving according to $F_d - F_f = mv'$, where $F_f = \mu F_N$ ($\mu < \mu_0$). In process Friction$_1$, defined below, discrete variable s represents the state of the process; it can have the values "neg", "stop", and "pos" (indicated by the comment -- {"neg", "stop", "pos"} in the model). These values correspond with negative, zero, and positive velocities respectively of the body. Variable F_N is declared as a discrete variable (F_N : real), because its value remains constant.

```
proc Friction$_1$($\varepsilon, \mu, \mu_0, m$ : real) =
|[ $F_f, F_d$ :: [N], $x$ :: [m], $v$ :: [m/s]
 , $F_N$ : real, $s$ : string    -- {"neg", "stop", "pos"}
 ; $x ::= -2$; $v ::= 0$; $F_N := mg$; $s :=$ "stop"
 | $F_d = \sin 0.25\pi\tau$
 ,  $v' = (F_d - F_f)/m$
 ,  $x' = v$
 , [ $s$ = "neg"  $\longrightarrow$ $F_f = -\mu F_N$
   [] $s$ = "stop" $\longrightarrow$ $F_f = F_d$
   [] $s$ = "pos"  $\longrightarrow$ $F_f = \mu F_N$
   ]
| $*[\ s =$ "stop" $;\ \nabla\, F_d > \mu_0 F_N \longrightarrow s :=$ "pos"
                                             $;\ v ::= \varepsilon$
   [] $s =$ "stop" $;\ \nabla\, F_d < -\mu_0 F_N \longrightarrow s :=$ "neg"
                                             $;\ v ::= -\varepsilon$
   [] $s \neq$ "stop" $;\ \nabla\, v = 0 \qquad\quad \longrightarrow s :=$ "stop"
   ]
]|
```

The discrete-event part of process Friction$_1$ consists of a selective waiting statement ([. . . [] . . . [] . . .]) that is repeated forever ($*$). If boolean expression $s =$ "stop" is true and driving force F_d becomes bigger than the threshold $\mu_0 F_N$, state event $\nabla\, F_d > \mu_0 F_N$ succeeds. As a result, the state switches to "pos" ($s :=$ "pos"), causing equation $F_f = \mu F_N$ to become active in the continuous-time part. The velocity v is then assigned a very small value ε ($v ::= \varepsilon$) in order to prevent the state event $\nabla\, v = 0$ from occurring immediately. Now that the state equals "pos", boolean expression $s \neq$ "stop" is true. This means that state event $\nabla\, v = 0$ is awaited. When v becomes equal to 0, the state changes to "stop" again.

Process Friction$_2$, that follows below, is a more detailed model. In this model, a small amount of energy is required to switch from state "stop" to state "pos" or "neg". For this purpose, a fourth state "try" is introduced. If $|F_d|$ becomes bigger than $\mu_0 F_N$, the state switches to "try". In this state, the frictional force equals $\mu_0 F_N$. If the driving force causes the velocity of the body to become bigger than ε (a small user defined value, e.g. 10^{-5}), the state switches to "pos". If, however, the driving force falls back below $\mu_0 F_N$, the state switches back to "stop". Fig. 5 shows the results of a 10 second simulation run for the process Friction$_1$ and Friction$_2$. The difference between the results of the two processes is too small to be shown graphically.

```
proc Friction$_2(\varepsilon, \mu, \mu_0, m$ : real$) =$
|[ $F_f, F_d$ : [N], $x$ : [m], $v$ : [m/s]
, $F_N$ : real, $s$ : string   −− {"neg", "stop", "try", "pos"}
; $x ::= -2;\ v ::= 0;\ F_N := mg;\ s :=$ "stop"
| $F_d = \sin 0.25\pi\tau$
, $v' = (F_d - F_f)/m$
, $x' = v$
, [ $s =$ "neg"  $\longrightarrow F_f = -\mu F_N$
  [] $s =$ "stop" $\longrightarrow F_f = F_d$
  [] $s =$ "try"  $\longrightarrow F_f = \mathrm{sign}(F_d) \cdot \mu_0 F_N$
  [] $s =$ "pos"  $\longrightarrow F_f = \mu F_N$
  ]
| $*[\ s =$ "stop" $;\ \nabla\, |F_d| > \mu_0 F_N$
      $\longrightarrow s :=$ "try"
         ; [ $\nabla\, v > \varepsilon$          $\longrightarrow s :=$ "pos"
           [] $\nabla\, v < \varepsilon$         $\longrightarrow s :=$ "neg"
           [] $\nabla\, |F_d| \leq \mu_0 F_N \longrightarrow s :=$ "stop" $;\ v ::= 0$
           ]
   [] $s \neq$ "stop" $;\ \nabla\, v = 0$
      $\longrightarrow s :=$ "stop"
   ]
]|
```

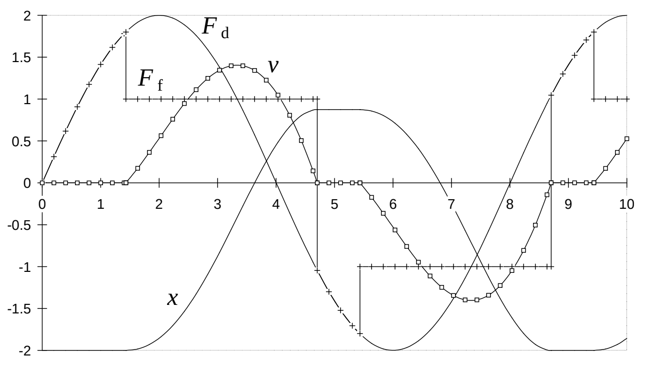

Fig. 5. Friction simulation ($\mu = 0.1$, $\mu_0 = 0.18$, $m = 1$, $\varepsilon = 10^{-5}$, $g = 10$).

4.3 *Additional examples*

Discrete-time controllers of a nonlinear tank system are discussed in Van Beek & Rooda (1997a). The nonlinearity is caused by a saturating actuator in the form of a valve. Several control strategies are modelled and simulated, including a PI controller with anti-windup (Bohn & Atherton, 1995). In Van Beek, Gordijn & Rooda (1995), a model of a plant for the biochemical production of ethanol is described. Detailed models of a conveyor line transportation system with actuators and sensors and the associated control system are presented in Van Beek, Rooda & Gordijn (1996) and Van Beek, Gordijn & Rooda (1997). Finally, a model of a bottle factory and other examples are presented in Van Beek & Rooda (1999).

5. CONCLUDING REMARKS

The proposed new classification of modelling languages into CT, CT+, DE, DE+, and CT/DE categories gives a better insight into the diversity of the so called hybrid modelling languages and simulators. The classification into different categories does not imply that languages from one category are better than languages from another. Depending on the type of applications that a modeller deals with, he or she should choose a language that fulfils his or her needs best. The CT/DE χ language and simulator have a wide field of application. They have been successfully applied to a large number of complex industrial cases, such as an integrated circuit manufacturing plant and a beer brewery.

References

Abacuss (1995). http://yoric.mit.edu/abacuss/abacuss.html. Massachussets Institute of Technology.

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

Andersson, M. (1994). *Object-oriented modeling and simulation of hybrid systems*. Ph.D. thesis, Department of Automatic Control. Lund Institute of Technology, Lund, Sweden.

Arends, N. W. A. (1996). *A systems engineering specification formalism*. Ph.D. thesis, Eindhoven University of Technology. The Netherlands.

Barton, P. I. (1992). *The modelling and simulation of combined discrete/continuous processes*. Ph.D. thesis. University of London.

Bohn, C., & Atherton, D. P. (1995). An analysis package comparing PID anti-windup strategies. *IEEE Control Systems, 15*(2), 34–40.

Bos, V., & Kleijn, J. J. T. (1999). *Structured operational semantics of Chi*. Computing Science Reports 99/01. Eindhoven University of Technology. Eindhoven.

Broenink, J. F. (1998). Modelling, simulation and analysis with 20-sim. *Journal A, 38*(3), 22–25.

Bujakiewicz, P., & Van den Bosch, P. P. J. (1991). A structured language for modelling and simulation of mixed continuous and discrete-event systems. *Proceedings of IFAC symposium on computer aided design in control systems*. (pp. 333–338) Swansea.

Champagnat, R., Esteban, P., Pingaud, H., & Valette, R. (1998). Modeling and simulation of a hybrid system through Pr/Tr PN-DAE model. *Hybrid dynamical systems—Proceedings of 3rd international conference on automation of mixed processes*. (pp. 131–137) Reims.

David, R., & Alla, H. (1994). Petri Nets for modeling of dynamic systems–a survey. *Automatica, 30*(2), 175–202.

Deshpande, A., Göllü, A., & Semenzato, L. (1998). The SHIFT programming language for dynamic networks of hybrid automata. *IEEE Transactions on Automatic Control, 43*(4), 584–587.

Elmqvist, H. (1994). *Dymola – dynamic modeling language – user's manual*. Dynasim AB. Lund, Sweden.

Fábián, G. (1999). *A language and simulator for hybrid systems*. Ph.D. thesis. Eindhoven University of Technology. The Netherlands.

Fábián, G., Van Beek, D. A., & Rooda, J. E. (1998). Integration of the discrete and the continuous behaviour in the hybrid Chi simulator. *Proceedings of the 1998 European simulation multiconference*. (pp. 252–257) Manchester.

Fritz, M., Preuss, K., & Engell, S. (1998). A framework for flexible simulation of batch plants. *Hybrid dynamical systems—Proceedings of 3rd international conference on automation of mixed processes*. (pp. 263–270) Reims.

Hoare, C. A. R. (1985). *Communicating sequential processes*. Englewood-Cliffs, NJ: Prentice-Hall.

Hohmann, S., & Zanne, C. (1998). Simulation tools for an induction machine modeled as a hybrid system. *Hybrid dynamical systems—Proceedings of 3rd international conference on automation of mixed processes*. (pp. 369–376) Reims.

Kasper, R., & Koch, W. (1995). Object-oriented behavioural modelling of mechatronic systems. *Proceedings of the 3rd conference on mechatronics and robotics '95*. Paderborn, Germany.

Kettenis, D. L. (1994). *Issues of parallelization in implementation of the combined simulation Language COSMOS*. Ph.D. thesis. Delft University of Technology.

Kleijn, J. J. T., Reniers, M. A., & Rooda, J. E. (1998). A process algebra based verification of a production system. *Proceedings of the 2nd IEEE international conference on formal engineering methods (ICFEM'98)*. (pp. 90–99) Brisbane.

Mattsson, S. E., Elmqvist, H., & Otter, M. (1998). Physical system modeling with Modelica. *Control Engineering Practice, 6*, 501–510.

Mitchell, E. E. L., & Gauthier, J. S. (1976). Advanced continuous simulation language (ACSL). *Simulation, 26*(3), 72–78.

Mosterman, P. J., Biswas, G., & Otter, M. (1998a). Simulation of discontinuities in physical system models based on conservation principles. *Proceedings of SCS summer simulation conference*. (pp. 320–325) Reno, Nevada.

Mosterman, P. J., Otter, M., & Elmqvist, H. (1998b). Modeling Petri nets as local constraint equations for hybrid systems using Modelica. *Proceedings of summer computer simulation conference 98*. (pp. 314–319).

Mosterman, P. J. (1999). An overview of hybrid simulation phenomena and their support by simulation packages. In: F. W. Vaandrager, & J. H. Van Schuppen. *Hybrid systems: computation and control*. Lecture Notes in Computer Science 1569. (pp. 165–177) Berlin: Springer.

Naumoski, G., & Alberts, W. (1998). *A discrete-event simulator for systems engineering*. Ph.D. thesis. Eindhoven University of Technology. The Netherlands.

Neptunix (1999). http://www.cisi.fr/vfr/produits/neptunix4.html. Cisi S.A. France.

Pegden, C. D., Shannon, R. E., & Sadowski, R. P. (1995). *Introduction to simulation using SIMAN*. London: McGraw-Hill.

Pritsker, A. A. B. (1974). *The GASP-IV simulation language*. London: Wiley.

Pritsker, A. A. B. (1986). *Introduction to simulation and SLAM II (3rd ed)*. New York: Wiley.

Rulkens, H. J. A., Van Campen, E. J. J., Van Herk, J., & Rooda, J. E. (1998). Batch size optimization of a furnace and pre-clean area by using dynamic simulations. *Proceedings of the SEMI/IEEE advanced semiconductor manufacturing conference*. (pp. 439–444) Boston.

SEAMS (1999). http://www.ececs.uc.edu/~mistie. Distributed Processing Laboratory, University of Cincinnati.

Sierenberg, R. W., & De Gans, O. B. (1992). Personal Prosim: A fully integrated simulation environment. *Proceedings of 1992 European simulation symposium*. (pp. 167–173) SCS, San Diego.

Simple++ (1999). http://www.aesop.de/simple. ASEOP GmbH.

Simplorer (1999). http://www.simplorer.com. SIMEC GmbH, Chemnitz, Germany.

Simulink/Stateflow (1999). http://www.mathworks.com. The MathWorks Inc.

Smile (1999). http://gargleblaster.cs.tu-berlin.de/~smile. Technical University of Berlin.

Valentin-Roubinet, C. (1998). Modelling of hybrid systems: DAE supervised by Petri nets the example of a gas storage. *Hybrid dynamical systems—Proceedings of 3rd international conference on automation of mixed Processes*, (pp. 142–149) Reims.

Van Beek, D. A., Gordijn, S. H. F., & Rooda, J. E. (1995). Integrating continuous-time and discrete-event concepts in process modelling, simulation and control. *Proceed-*

ings of the first world conference on integrated design and process technology. (pp. 197–204).

Van Beek, D. A., Gordijn, S. H. F., & Rooda, J. E. (1997). Integrating continuous-time and discrete-event concepts in modelling and simulation of manufacturing machines. *Simulation Practice and Theory*, *5*, 653–669.

Van Beek, D. A., & Rooda, J. E. (1997a). Design of discrete controllers for continuous systems using hybrid Chi. *Proceedings of IFAC 7th symposium on computer aided control systems design (CACSD'97)*. (pp. 9–14) Gent.

Van Beek, D. A., & Rooda, J. E. (1997b). Specification and simulation of industrial systems using an executable mathematical specification language. *Proceedings of the 15th. IMACS world congress vol. 2. Numerical Mathematics*. (pp. 721–726) Berlin.

Van Beek, D. A., & Rooda, J. E. (1999). *Production and process systems modelling*. Lecture notes. Eindhoven University of Technology, Department of Mechanical Engineering. The Netherlands.

Van Beek, D. A., Rooda, J. E., & Gordijn, S. H. F. (1996). Hybrid modelling in discrete-event control system design. *CESA'96 IMACS multiconference, symposium on discrete events and manufacturing systems*. (pp. 596–601) Lille.

Van de Mortel-Fronczak, J. M., & Rooda, J. E. (1996). On the integral modelling of control and production management systems. *Proceedings of APMS'96*. (pp. 171–176) Kyoto, Japan.

Van de Mortel-Fronczak, J. M., Rooda, J. E., & Van den Nieuwelaar, N. J. M. (1995). Specification of a flexible manufacturing system using concurrent programming. *Concurrent Engineering: Research and Applications*, *3*(3), 187–194.

VHDL-AMS (1999). http://vhdl.org/vi/analog/. IEEE 1076.1 Working Group.