

## Multi-domain modelling, simulation, and control

D.A. van Beek\* and J.E. Rooda  
Eindhoven University of Technology  
Dept. Mechanical Engineering, WH 4.105  
P.O. Box 513, 5600 MB Eindhoven, Netherlands  
\* E-mail: d.a.v.beek@tue.nl

### Abstract

For the design of large industrial systems, usually many different modelling languages are required. Currently, different projects aim at the development of standards that enable interaction between simulation models written in different languages. The Chi project aims at providing one language suited to modelling, simulation and control of systems from different application domains. Models may range from pure continuous-time models to pure discrete-event models, and any combination of the two. Due to the orthogonal design of the Chi language and careful selection of the language primitives, the core of the language is small. At the same time, the high-level data modelling constructs and high-level behaviour modelling constructs provide the modeller with expressive power. Verification of Chi models is currently limited to discrete-event models. The Chi language and simulator have proved themselves in many industrial projects involving discrete-event modelling and simulation. For continuous-time and hybrid systems, the current simulator provides time-event and state-event handling, root location, initial state calculation, handling of discrete-event sub-phases, and index 1 DAE solving. For real-time control, Chi models can be compiled to run on real-time operating systems such as VxWorks.

*Keywords:* Multi-domain modelling, simulation, verification, control, hybrid systems

### 1 INTRODUCTION

Production systems can be divided into two types: discrete production systems, and production systems of the so-called process type. Discrete production systems can be characterized by the fact that discrete products need to be positioned, e.g. in assembly lines. In production systems of the process type on the other hand, there is no positioning of intermediate products. This is especially clear when materials are in gaseous, liquid, or powder form. For 'process-type production' the distinction can be made between batch production and continuous production. In batch processes, several process steps are executed in one place, usually a tank, such as a batch reactor. In the reactor, operating conditions vary as a function of time. After execution of the process steps, the materials are transported. In continuous processes on the other hand, there is a continuous flow of materials and operating conditions are meant to vary only as a function of place in the production process, not as a function of time [32].

Production systems, especially discrete and batch production systems, are often far too complex for detailed mathematical analysis, because of the large size, stochastic behaviour, parallelism, shared resources, timing and other constraints, interactions between subsystems, and different possible routings for the products. Therefore, simulation is used for analysis and improvement of the time-dependent behaviour of these systems. Currently, different languages are required for simulation of discrete, batch, and continuous production

systems, respectively. Large industrial plants consist of systems from many different domains: electrical, mechanical, pneumatical, chemical systems; local and global (supervisory) control systems. For each of these systems, different application-specific simulation languages are available. In some cases, simulation models of control systems can be automatically converted to real-time control systems. In many other cases, such as control systems based on scheduling rules, simulation models cannot be reused for the development of real-time control systems. As a result, many different languages may be required for the design and operation of large industrial systems. The application of each language is limited to the design and analysis of some subsystem. An optimal design of several subsystems will not, however, necessarily lead to an optimal design of the total system.

### 2 INTEGRATION OF MODEL COMPONENTS

Several paths are being followed to enable communication between models of components from different domains. The Defense Modelling and Simulation Office of the USA leads an effort to establish a common technical framework to facilitate the interoperability of all types of models and simulations. This High Level Architecture (HLA [11]) has been accepted as a draft IEEE standard (IEEE 1516). Emphasis is on discrete-event models and simulations. A project which emphasizes continuous-time models and simulations, with or without discontinuities, is Global CAPE-

OPEN [8]. Its objective is to stimulate the development of open standard interfaces for component-based process simulation and computer-aided process engineering. The “Open Toolset for Mixed Simulation of Multi-domain Systems” (TOOLSYS) project attempts to use VHDL-AMS [18] as a means to integrate multi-domain system models [2]. Finally, for the design of embedded systems, several multi-language simulation environments exist. Two of these are Ptolomy [17] and ClearSim-MultiDomain [34].

The  $\chi$  (or Chi) project also aims at integration of model components. However, it does this in a way complementary to the HLA, Global CAPE-OPEN, TOOLSYS, Ptolomy and ClearSim-MultiDomain approaches. The aim is to provide *one* language suited to modelling, simulation, control, and verification of pure discrete-event systems, pure continuous-time systems and combined discrete-event / continuous-time systems. The application field is intended to be very wide: ranging from chemical or mechanical physical systems [38] to complete production systems such as a fruit-juice production plant [16], or an integrated circuit manufacturing plant [33]. Control systems can be integrated with the model parts that describe the physical behaviour and can range from control systems of individual machines [36] to control systems based on complex scheduling rules [26]. One of the goals is to be able to reuse the control algorithms of the simulation models for actual real-time control [41], which requires, among others, a high-level exception handling mechanism [37].

Another multi-domain modelling and simulation language is Modelica [27]. A difference with the  $\chi$  language is that Modelica is directed more towards the continuous-time domain, and incorporates many different equation solvers. Discrete actions in Modelica are modelled in an equation-like manner. Higher level discrete components are made available by means of component libraries, such as a Place and Transition library [28] for Petri net modelling.

Although the intended application field of  $\chi$  is very wide, there are still many application areas for which special purpose environments are preferable. Modelling and simulation of integrated circuits is an example of an application area that is best dealt with using a language such as VHDL-AMS [18], that has been developed especially for this purpose.

### 3 DESIGN OF THE $\chi$ LANGUAGE

This section extends the treatment of the  $\chi$  syntax and semantics in [38] by including design considerations, background information, and major recent changes.

#### 3.1 Requirements

The most essential requirements for the design of the  $\chi$  language are:

- *Multi-domain* modelling should be supported. Although there are many quite different specialized modelling languages, the required language constructs for modelling of physical systems are relatively independent of the application domain. A goal of the  $\chi$  project is to find a minimal set of language elements suited to modelling of a wide variety of systems. In some domains, a continuous-time model is best suited. In other domains, a pure discrete-event model, or a hybrid model is best suited.
- All models should be executable by means of *simulation*, see Section 4.
- Model *verification* should be supported, see Section 5.
- A ‘direct’ transition from simulation to *real-time control* should be possible, see Section 6.
- The language should consist of a *small* number of *orthogonal* (not overlapping and free to combine) basic language elements.
- The language should be sufficiently *expressive* to support the modeller in the development of models of complex systems in a limited amount of time. Expressive power is required both for behaviour modelling and for data modelling. In order to provide sufficient expressive power, the language should support *modular composition*.
- The language should be readable and usable by a wide range of users, not just by expert computer scientists.

#### 3.2 Background

The continuous-time part of  $\chi$  is based on DAEs (differential algebraic equations). The discrete-event part is based on a combination of communicating sequential processes (CSP) with Dijkstra’s guarded commands [12], such as described in [21]. However, for communication, the channel concept from [22] is used. The  $\chi$  language differs considerably from other CSP variants. The reason for this is that most CSP variants are based on Hoare’s CSP version of 1985 [22]. This includes the CSP variants described in [20], and the hybrid CSP versions [24, 10]. The main purpose of these CSP variants is formal analysis. Processes are specified in a functional way, using recursion, while support for data modelling is minimal. Specification of  $\chi$  processes on the other hand, is done in an imperative way, using a sequence of statements, including assignments, conditional statements, and repetition; and advanced data modelling constructs are available. In this way, the  $\chi$  language is readable and usable (after getting acquainted with the basic principles) by a wide range of industrial users, while at the same time a formal semantics is available [7]. The Ptolomy project [17]

also includes a basic CSP-based language [35]. The difference with  $\chi$  is that, where the Ptolomy project aims at obtaining high expressivity by integrating several different languages into one modelling environment, the  $\chi$  project aims at obtaining a highly expressive language by integration of a carefully selected small number of orthogonal language elements. This orthogonal integration of language elements is especially evident in the highly expressive selective waiting statement, explained in Section 3.6.

### 3.3 Modular composition

Modular composition is provided by means of processes and systems that interact by means of channels and shared variables. The basic building block of a  $\chi$  model is a process, which may consist of a continuous-time part only, a discrete-event part only, or a combination of both. A system consists of a number of process (or system) instantiations, and channels and/or shared variables connecting these processes. The fact that the model of a system may contain instantiations of other systems, makes it possible to model systems in a hierarchical way. Interaction between continuous-time parts of processes, consisting of (differential algebraic) equations, takes place by means of shared variables. Interaction between the discrete-event part of a process and the continuous-time part of the same, or another process, also takes place by means of shared variables: in the discrete-event part of a process, assignments can be made to variables that occur in the equations of the continuous-time part; and by means of the state-event statement  $\nabla b$  (see Section 3.6), the discrete-event part of a process can synchronize with the continuous part of a process. Interaction between discrete-event parts of processes typically takes place by means of (synchronous communication) channels. In some cases, however, interaction is more conveniently specified using a discrete shared variable (such as a boolean variable representing a sensor), and a state-event statement  $\nabla b$  involving the shared variable. A model of a conveyor system based on modular composition according to the most recent  $\chi$  syntax and semantics is presented in [39].

Continuous channels are no longer used in  $\chi$ . The main reason for this is that shared variables are a more straightforward mechanism than continuous channels. Shared variables lead to systems that have less variables, and that are easier to understand. For instance, consider a control system and a physical system interacting by means of a sensor and an actuator. Using shared variables, a sensor can be represented by one shared variable, that is referred to both in the physical system and in the control system. Using continuous channels, the control system and the physical system each have their own local sensor variable. By connecting these two sensor variables by means of a continuous channel, an equality is introduced between the two sensor variables. The difference between the two modelling techniques is clarified by comparing the  $\chi$  specification of a conveyor

system using continuous channels in [36], with the  $\chi$  specification of a similar conveyor system using shared variables in [39].

Channels are used for synchronization and communication between processes. They are essential in two ways: to model communication and synchronization between control systems, and to model material transport between physical processes. The transported material can be conveniently modelled as a data object in  $\chi$ . Such a data object could, for example, include information on the production steps (recipe) that the material needs to undergo. Transportation of the material requires the sending and receiving processes to be ready at the same time. Therefore, the synchronous communication channels in  $\chi$  are very well suited to the modelling of material transport.

### 3.4 Data modelling

Besides the basic data types such as bool (boolean), int (integer), real, and string, structured data types are available in the form of array, set, list and tuple types. Tuples are comparable to records. In order to provide the modeller with sufficient expressive power, many operations on these data types are available. Examples are the built-in language elements for operations on sets and lists, such as set/list comprehension [5] that derive a subset/sublist of a set/list for which all elements satisfy a specified condition. Besides this, functions may be defined at the outer level of the model. Such user defined functions can be used in expressions both in the continuous-time and discrete-event parts of  $\chi$  models. All variables are declared as either continuous using a double colon (e.g.  $V :: \text{real}$ ), or discrete using a single colon (e.g.  $n : \text{int}$ ). The continuous variables are the unknowns in the equations, whereas the discrete variables are known in the equations. Therefore, the value of a discrete variable is determined by assignments only.

### 3.5 Discussion of the continuous-time part of $\chi$

The continuous-time part is clearly separated from the discrete-event part. In this way, pure continuous-time processes can be specified using only equations, shared variables, and processes and systems as required language elements. A time derivative is indicated by a prime character (e.g.  $x'$ ). DAEs (Differential Algebraic Equations) are separated by commas:  $DAE_1, DAE_2, \dots, DAE_n$ . A DAE can be a normal equation or a *guarded equation* (*[GE]*), also referred to as a conditional equation. The latter is used when the set of active equations depends on the state of the system. The syntax is  $[b_1 \longrightarrow DAE_{s_1} \parallel \dots \parallel b_n \longrightarrow DAE_{s_n}]$ , where  $DAE_{s_i}$  ( $1 \leq i \leq n$ ) 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 equations  $DAE_{s_i}$  associated with an open guard can be selected to become part of the set of active equations. The normal, non-guarded equations are always part of the active equations. When all

discrete processes are blocked, the active equations are solved. For this purpose, the number of active equations must be equal to the number of continuous variables. A summary of the continuous-time language constructs in Backus-Naur Form (BNF) is given in Table 1, where  $r$  is an expression of type real.

Table 1: BNF syntax of the continuous-time part of  $\chi$ .

$DAE$	$::=$	$r = r \mid [GE] \mid DAE, DAE$
$GE$	$::=$	$b \longrightarrow DAE \mid GE \parallel GE$

### 3.6 Discussion of the discrete-event part of $\chi$

A summary of the discrete-event language constructs [40] is given in Table 2. Besides these elementary language constructs, there are also a number of language constructs that can be considered as ‘syntactic sugar’ (e.g.  $c!$  and  $c?$ ). The meaning of these language elements can be expressed in terms of the elementary language elements.

Table 2: BNF syntax of the discrete-event part of  $\chi$ .

$E$	$::=$	$c!e \mid c?x \mid \Delta r \mid \nabla b$
$GB$	$::=$	$b \longrightarrow S \mid GB \parallel GB$
$GW$	$::=$	$b; E \longrightarrow S \mid GW \parallel GW$
$G$	$::=$	$[GB] \mid [GW]$
$S$	$::=$	$x := e \mid E \mid G \mid *G \mid S; S$

*Event statements*, denoted by non-terminal  $E$ , are statements in which time may pass. There are three types of event statements: *communication statements* ( $c!e$ ,  $c?x$ ), *time passing statements* ( $\Delta t$ ), and *state-event statements* ( $\nabla b$ ).

Synchronous *communication* is specified by a send statement  $c!e$  and a receive statement  $c?x$ , where  $c$  is a channel 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  $x$ . As syntactic sugar, the language provides two shorter notations for the case that no data needs to be exchanged between processes (pure synchronization). The send and receive statements  $c!$ ,  $c?$  may be used, or both processes may use  $c\sim$ . The semantics is analogous to  $c!e$  and  $c?x$ , but without data exchange.

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

*State-events* are specified by means of the *state-event* statement  $\nabla b$ , where  $b$  is a boolean expression. A process executing  $\nabla b$  is blocked until expression  $b$  becomes true. There are two syntactically different forms of statement  $\nabla b$ . If at least one continuous variable is used in  $b$ , the syntax of  $b$  must be of the form  $r_1 \text{ rop } r_2$ , where  $r_1$  and  $r_2$  are expressions of type real, and *rop* is a relational operator ( $\leq, <, >, \geq$ ). An

example is  $\nabla V > 5$ , where  $V$  can be a continuous variable representing the volume of a tank. By means of this form, the discrete-event part of a process can synchronize with the continuous-time part of a process. The reason for the restricted form  $r_1 \text{ rop } r_2$  is that general purpose numerical solvers, such as DASSL [31], cannot directly detect the fact that an expression becomes true. They can only detect that a user supplied ‘root function’ crosses zero. The restriction of the form  $r_1 \text{ rop } r_2$  makes it possible to use  $r_1 - r_2$  as a root function. When it crosses zero, the state-event is assumed to have taken place, and the solver stops solving. State-event statement  $\nabla r_1 \geq r_2$  is given to the solver as root function  $r_1 - r_2$ . State-event statement  $\nabla r_1 > r_2$  is given to the solver as  $\nabla r_1 \geq r_2 + \varepsilon$  (under the condition that  $r_1 \leq r_2$ , otherwise the statement  $\nabla r_1 > r_2$  terminates immediately), so as root function  $r_1 - r_2 - \varepsilon$ , where  $\varepsilon$  is a very small positive value. If no continuous variables are used in expression  $b$  (in  $\nabla b$ ), there are no additional restrictions on the form of the expression. This case is especially useful for interaction with sensors that are modelled by means of a boolean variable; e.g.  $\nabla s$ , where  $s$  is a boolean variable representing the state of a sensor [39].

*Selection* ( $[GB]$ ) is specified by  $[b_1 \longrightarrow S_1 \parallel \dots \parallel b_n \longrightarrow S_n]$ . The boolean expression  $b_i$  ( $1 \leq i \leq n$ ) 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  $[b_1; E_1 \longrightarrow S_1 \parallel \dots \parallel b_n; E_n \longrightarrow S_n]$ . 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. Guards are evaluated only once, directly when the selective waiting statement is executed. The time-event specified by  $\Delta t$  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 non-deterministically chosen for execution, followed by execution of the corresponding  $S_i$ . The selective waiting statement is the most flexible and powerful statement of the  $\chi$  language. It can be used to model simultaneous waiting for an arbitrary number of different events: either a synchronization event ( $c?$  or  $c!$ ), a communication event ( $c?x$  or  $c!e$ ), a state-event ( $\nabla r$ ), or a time-event ( $\Delta t$ ). The statement is a good example of the orthogonal design of the  $\chi$  language. The additional boolean guards of the selective waiting statement further increase its expressivity, so that it can also be used to model a hybrid automaton [1]. Below, an example of a  $\chi$  model of a hybrid automaton with two states is given, where  $s$  is a variable describing the state. The  $|$  symbol separates the continuous and the discrete parts of the specification. The continuous part consists of a guarded equation; the discrete part consists of a repetitive selective waiting statement:

$$\begin{array}{l}
| [ s = 0 \longrightarrow f_0(x', x) = 0 \\
\quad \parallel s = 1 \longrightarrow f_1(x', x) = 0 \\
\quad ] \\
| *[ s = 0; \nabla b_{01} \longrightarrow s := 1; x := k_1 \\
\quad \parallel s = 0; \Delta 2 \longrightarrow s := 1; x := k_1 \\
\quad \parallel s = 1; \nabla b_{10} \longrightarrow s := 0; x := k_0 \\
\quad \parallel s = 1; c?y \longrightarrow s := 0; x := k_0 \\
\quad ]
\end{array}$$

State 0 ( $s = 0$ ) of the automaton is associated with equation  $f_0(x', x) = 0$ , and state 1 ( $s = 1$ ) with equation  $f_1(x', x) = 0$ . A transition from state 0 to 1 occurs when boolean expression  $b_{01}$  becomes true, or after a time-out of 2 time-units. In both cases,  $x$  is set to  $k_1$ . A transition from state 1 to state 0 occurs when boolean expression  $b_{10}$  becomes true, or when a receive action on channel  $c$  occurs. In both cases,  $x$  is set to  $k_0$ . Although a receive action is not a primitive operation of a hybrid automaton, the action is included in the example to show the versatility of the selective waiting statement.

*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. If all guards are closed, the repetition terminates.

The discrete-event part of a process consists of a statement  $S$ . Such a statement can be an assignment statement  $x := e$ , an event statement  $E$ , a guarded command  $G$ , a repetition  $*G$ , or a sequential composition of statements  $S$ ;  $S$  (see Table 2).

#### 4 SIMULATION

Information on the history of the  $\chi$  simulator and on the way to use the simulator can be found in [38, 13, 3], where [38] also positions the  $\chi$  language with respect to other hybrid simulation languages. Currently, the simulator features simulation of ODEs and DAEs of index 1, handling of time-events, detection of state-events, and iterative root location [29]. Two solvers are used: DASSL and NLEQ. DASSL [31] is used to solve the equations as a function of time. Discontinuities in the discrete-event phase may cause the set of equations to become inconsistent. The NLEQ solver [30] calculates a new consistent state when all discrete processes are blocked. This new state may trigger additional state-events that cause a new discrete-event sub-phase to be started [14], thus prolonging the current discrete-event phase. The simulator has been successfully applied to a large number of industrial cases, some of which have been mentioned in Section 2. For these cases, the high-level data and behaviour modelling constructs of the  $\chi$  language, that are all executable by means of simulation, are essential.

Currently, the number of continuous variables and the number of equations must remain constant during a simulation run, and a differential variable (a variable occurring with a prime in the equations) cannot change dynamically into an algebraic variable (variable that does not occur with a prime in the equations).

Future improvements will be directed towards enabling dynamical changes in the number and type of continuous variables, and towards speeding up simulation of models with equations, by means of block lower triangular partitioning (BLT) and tearing [9].

Libraries of components are currently not supported in  $\chi$ . An experimental version of libraries for the discrete-event  $\chi$  simulator was later removed, since it appeared that there was little overlap between components of accurate discrete-event models of large industrial production systems. For hybrid  $\chi$ , much work will need to be done on the syntax and semantics of the language, and on the implementation of the simulator before libraries are reconsidered.

The  $\chi$  simulator is available for the Linux and MS-Windows platform. However, since the implementation is based on the 1998 ISO/ANSI-C++ standard, it should be able to run on any platform that conforms to that standard. The simulator is currently being redesigned, so that the different parts of the simulator (compiler, kernel, discrete part, continuous part) are clearly separated from one another, and interact by means of well-defined interfaces. It is also redesigned to conform to the new syntax and semantics. It is expected to be available in the beginning of 2001.

#### 5 VERIFICATION

A joint Ph.D. project has been set up between the Systems Engineering Group [19] of the Mechanical Engineering Department and the Formal Methods Group of the Computing Science Department of the Eindhoven University of Technology. The project focuses on formal analysis of discrete-event  $\chi$  models only. Results of this project so far are described in [25] and [7]. The complete operational semantics of the discrete-event part of  $\chi$  is described in [6]. As a result of insights gained from this project, the language syntax and semantics has been improved in several aspects. The most important of these changes are listed below.

- The semantics of the delay statement ( $\Delta t$ ) has been changed. The effect of the change is best illustrated by the semantics of  $\Delta 0$ , a zero delay. The semantics of  $\Delta 0$  occurring in a sequential composition of statements (e.g.  $c?x; \Delta 0; d!x + 1$ ) is equivalent to a “skip” statement (a no-operation). Originally, the semantics of a  $\Delta 0$  as part of a selective waiting statement (e.g.  $[ c?x \longrightarrow \dots \parallel d!1 \longrightarrow \dots ] \Delta 0 \longrightarrow \dots$ ) was as follows: if one of the communications ( $c?x, d!1$ ) was possible, it would be executed. The delta zero would be executed only if none of the communications were possible at the current time point. In the new semantics, the choice between  $\Delta 0$  and communication statements that can be executed ( $c?x$  or  $d!1$ ) is non-deterministic.
- The “longest waiting communication first” priority

rule has been removed from the language. In the original semantics, if in a selective waiting statement (e.g.  $[c?x \longrightarrow \dots \parallel d!1 \longrightarrow \dots]$ ), two (or more) communications were possible at the same time ( $c?x$  and  $d!1$ ), that communication was selected for which the communicating partner process was waiting longest. In the new semantics, if several communications are possible, one of them is selected non-deterministically. Any kind of priority between different communications must now be modelled explicitly.

- Syntactical differences between assignments to discrete variables, continuous differential variables, and continuous algebraic variables (initial guesses) have been removed. There is now only one assignment operator ( $:=$ ). The reason for this is that there is no semantic difference between these assignments.

The effect of the changes in the semantics of the selective waiting statement, is that the choice between enabled event statements in a selective waiting statement is now non-deterministic. In this way, the operational semantics has been simplified considerably, which was necessary for the purpose of verification.

A similar joint Ph.D. project will be started for hybrid  $\chi$ . The aim of the project is to integrate the knowledge and demands from: 1) the field of modelling and simulation of hybrid systems, and 2) the field of verification of hybrid systems.

## 6 REAL-TIME CONTROL

Real-time control using  $\chi$  is treated in [23]. The (discrete-event)  $\chi$  model of a control system can be tested by means of simulation, and it can be analyzed by means of formal verification. A case study using  $\chi$  for real-time control of a scaled-down mock-up of a paint factory is presented in [41]. This case study is based on a discrete-event version of  $\chi$ , so without continuous variables and equations. The paint factory is controlled by means of binary actuators and sensors (mostly binary). For the purpose of simulation-based testing of the control system, the control system and the controlled system are both modelled in  $\chi$ . Processes in this model interact only by means of channels via synchronous communication. Part of the structure of this model is informally shown in Figure 1.

The binary sensors are modelled by means of the  $\chi$  processes  $S$ . These sensor processes interact with the  $\chi$  control processes  $C_1$  and  $C_2$  by means of three channels each, that are referred to as *on*, *off*, and *value*. When the status of the modelled sensor is ‘on’, the sensor process is prepared to synchronize via channel *on*; when the sensor is ‘off’, its process is prepared to synchronize via channel *off*; and for both states of the sensor, the sensor process is prepared to communicate the current status (on or off) via channel *value*.

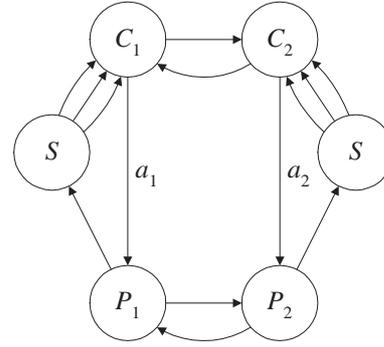


Figure 1: Part of the paintfactory model.

The control processes  $C_1$  and  $C_2$  can synchronize via channels *on* and *off* in order to wait until the sensor is on or off, respectively. The current status of the sensor can be received by means of communication via channel *value*. The status of a sensor model is determined by the processes  $P_1$  and  $P_2$ , that model a part of the paint factory. If a sensor needs to change its status, it receives a message via a channel from process  $P_1$  or  $P_2$ . Activation of the actuators is modelled by means of sending a boolean via channel  $a_1$  or  $a_2$ . In this way, the only means of interaction between the control system processes and the processes that model the paint factory is by means of channels. Of course, the control system processes also interact between themselves by means of channels. When the combined model of the control system and the paint factory model has been tested sufficiently by means of simulation, the model of the paint factory and the sensors and actuators are replaced by the real physical system. For this purpose, the  $\chi$  specification of the control processes needs to be changed in one respect only: all communication operations on the channels for interaction with the sensors and actuators need to be replaced by IO-operations that directly access the hardware sensors and actuators. For example, synchronizations with the channels *on* and *off* are replaced by calls to `waitBit()`; and a receive action on channel *value* is replaced by a call to `readBit()`. Future research will deal with simplification or automation of this conversion, and will also consider hybrid models of the controlled system, such as discussed in [36, 39].

The resulting  $\chi$  control system specification is compiled by a special version of the  $\chi$  compiler that generates C++ code. This code is linked to a real-time  $\chi$  kernel and to other libraries and drivers. The resulting code can be downloaded to the real-time operating system VxWorks [42] for real-time control. This transition from a  $\chi$  control system specification to a real-time executable was done in accordance with the POSIX series of standards, where POSIX is an acronym loosely defined as a portable operating system interface based on UNIX. It refers to a collection of international standards for UNIX-style operating system interfaces, and is specified by several IEEE 1003.xx standards.

## 7 COMPARISONS

Representative cases can be useful to get an impression of the possibilities of a language and simulator. The official membership journal “Simulation News Europe” (SNE) of EUROSIM and SCS Europe has published 12 comparison benchmarks [4], ranging from pure continuous-time cases to hybrid and discrete-event cases; e.g. comparisons entitled ‘Lithium-Cluster Dynamics under Electron Bombardment’, ‘Constrained Pendulum’, ‘Fuzzy Control of a Two Tank System’, ‘SCARA Robot’, and ‘Flexible Assembly System’. Results from different simulation languages are categorized and published by ARGESIM, both on the Internet and in Simulation News Europe. In [15] the tasks described in comparisons 1, 3, 5, 7, 9, 11, 12 and CP1 have been modelled and simulated using  $\chi$ . Comparison 3, and the third task from comparison CP1 could not be solved with the current version of the  $\chi$  simulator. All other comparison tasks were completely solved. The comparisons were all of a continuous-time or hybrid nature. The discrete-event comparisons 6 and 8 are currently being tackled. The remaining discrete-event comparisons 2 and 4 will be tackled in future.

## 8 CONCLUSIONS

The application domain of the  $\chi$  language is very wide. Models may range from pure continuous-time models to pure discrete-event models, and any combination of the two. Due to the orthogonal design of the language and careful selection of the language primitives, the core of the language is small. At the same time, the high-level data modelling constructs and high-level behaviour modelling constructs provide the modeller with expressive power. The language and simulator have been successfully applied in many industrial projects. The discrete-event  $\chi$  verification project has resulted in considerable improvements to the language. In future, verification will also aim at hybrid  $\chi$  models. For simulation of continuous-time and hybrid models, the  $\chi$  simulator provides time-event and state-event handling, root location, initial state calculation, handling of discrete-event sub phases, and index 1 DAE solving. For the purpose of real-time control,  $\chi$  models can be compiled to executables that run under real-time operating systems such as VxWorks.

## 9 ACKNOWLEDGMENTS

The authors would like to thank V. Bos, A. T. Hofkamp, J. J. T. Kleijn, J. M. van de Mortel-Fronczak, and the anonymous referees for their helpful comments on the drafts of the paper.

## REFERENCES

- [1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis

of hybrid systems. In *Theoretical Computer Science* 138, pages 3–34. Springer, 1995.

- [2] M. Andersson and P. Tsigari. Simulation of vehicle breaking stability, a model integration case study. In *Proceedings of the 11th. European Simulation Symposium*, pages 137–141, Erlangen, 1999.
- [3] N. W. A. Arends. *A Systems Engineering Specification Formalism*. PhD thesis, Eindhoven University of Technology, 1996.
- [4] ARGESIM. ARGESIM comparisons. <http://www.argesim.org/>, 2000.
- [5] R. Bird. *Introduction to Functional Programming using Haskell*. Prentice-Hall, London, 1998.
- [6] V. Bos and J. J. T. Kleijn. Discrete  $\chi$ : a production systems modelling language. Technical report, Eindhoven University of Technology, 2000. To appear.
- [7] V. Bos and J. J. T. Kleijn. Formalisation of a production systems modelling language: the operational semantics of  $\chi$  core. *Fundamenta Informaticae*, 41(4):367–392, 2000.
- [8] Global CAPE-OPEN. <http://www.global-cape-open.org/>, 2000.
- [9] E. Carpanzano and C. Maffezzoni. Symbolic manipulation techniques for model simplification in object-oriented modelling of large scale continuous systems. *Mathematics and Computers in Simulation*, pages 133–150, 1988.
- [10] Zhou Chaochen, Wang Ji, and Anders P. Ravn. A formal description of hybrid systems. In *Hybrid Systems III - Verification and Control*, Lecture Notes in Computer Science 1066, pages 511–530. Springer, 1996.
- [11] Defense Modelling and Simulation Office. High level architecture. <http://hla.dmsomil/>, 2000.
- [12] E. W. Dijkstra. Guarded commands, non-determinacy, and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.
- [13] G. Fábíán. *A Language and Simulator for Hybrid Systems*. PhD thesis, Eindhoven University of Technology, 1999.
- [14] G. Fábíán, D. A. van Beek, and J. E. Rooda. Integration of the discrete and the continuous behaviour in the hybrid Chi simulator. In *Proceedings of the 1998 European Simulation Multiconference*, pages 252–257, Manchester, 1998.
- [15] J. J. Marcos Fernandez. Application of  $\chi$  to the hybrid models of the ARGESIM comparisons. Systems Engineering Technical Report SE 420224, Eindhoven University of Technology, Dept. of Mechanical Engineering, 1999.

- [16] J. J. H. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. PhD thesis, Eindhoven University of Technology, 2000.
- [17] A. Girault, B. Lee, and E. A. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):742–760, 1999.
- [18] IEEE 1076.1 Working Group. VHDL-AMS. <http://vhdl.org/vi/analog/>, 1999.
- [19] Systems Engineering Group. <http://se.wtb.tue.nl>. Eindhoven University of Technology, 2000.
- [20] Michael G. Hinchey and Stephen A. Jarvis. *Concurrent Systems: Formal Developments in CSP*. McGraw-Hill, London, 1995.
- [21] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
- [22] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood-Cliffs, 1985.
- [23] A. T. Hofkamp. *Reactive Machine Control*. PhD thesis, Eindhoven University of Technology, 2000. To appear.
- [24] He Jifeng. From CSP to hybrid systems. In A. W. Roscoe, editor, *A Classical Mind, Essays in Honour of C.A.R. Hoare*, pages 171–189. Prentice Hall, 1994.
- [25] J. J. T. Kleijn, M. A. Reniers, and J. E. Rooda. A process algebra based verification of a production system. In *Proc. of the 2nd IEEE International Conference on Formal Engineering Methods*, pages 90–99, Brisbane, 1998.
- [26] B. Lemmen, E. J. J. van Campen, H. Roede, and J. E. Rooda. Clustertool optimization through scheduling rules. In *Eighth International Symposium on Semiconductor Manufacturing*, pages 82–92, Santa Clara, 1999.
- [27] S. E. Mattsson, H. Elmqvist, and M. Otter. Physical system modeling with Modelica. *Control Engineering Practice*, 6:501–510, 1998.
- [28] Pieter Mosterman, Martin Otter, and Hilding Elmqvist. Modeling Petri nets as local constraint equations for hybrid systems using Modelica. In *Proc. Summer Computer Simulation Conference 98*, pages 314–319, 1998.
- [29] Pieter J. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1569, pages 165–177. Springer, 1999.
- [30] U. Novak and L. Weiman. A family of newton codes for systems of highly nonlinear equations. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1991.
- [31] L. R. Petzold. A description of DASSL: A differential/algebraic system solver. *Scientific Computing*, pages 65–68, 1983.
- [32] J. E. Rijnsdorp. *Integrated Process Control and Automation*. Elsevier, Amsterdam, 1991.
- [33] H. J. A. Rulkens, E. J. J. van Campen, J. van Herk, and J. E. Rooda. Batch size optimization of a furnace and pre-clean area by using dynamic simulations. In *SEMI/IEEE Advanced Semiconductor Manufacturing Conference*, pages 439–444, Boston, 1998.
- [34] S. Scherber, K. Wöllhaf, and C. Müller-Schloer. An industrial approach for the simulation of complex heterogeneous systems. In *Proceedings of the 14th. European Simulation Multiconference*, pages 318–322, Ghent, 2000.
- [35] N. Smyth. Communicating sequential processes domain in Ptolemy II. Master’s thesis, University of California, Berkeley, 1998. UCB/ERL Memorandum M98/70.
- [36] D. A. van Beek, S. H. F. Gordijn, and J. E. Rooda. Integrating continuous-time and discrete-event concepts in modelling and simulation of manufacturing machines. *Simulation Practice and Theory*, 5:653–669, 1997.
- [37] D. A. van Beek and J. E. Rooda. A new mechanism for exception handling in concurrent control systems. *European Journal of Control*, 2(2):88–100, 1996.
- [38] D. A. van Beek and J. E. Rooda. Languages and applications in hybrid modelling and simulation: Positioning of Chi. *Control Engineering Practice*, 8(1):81–91, 2000.
- [39] D. A. van Beek and J. E. Rooda. Semantics of state-events in hybrid languages. In *Proceedings of the Third IMACS Symposium on Mathematical Modelling*, pages 421–424, Vienna, 2000.
- [40] J. M. van de Mortel-Fronczak, J. E. Rooda, and N. J. M. van den Nieuwelaar. Specification of a flexible manufacturing system using concurrent programming. *Concurrent Engineering: Research and Applications*, 3(3):187–194, 1995.
- [41] M. H. M. van Duin. From simulation using  $\chi$  to implementation using VxWorks. a case: The paint factory. Master’s thesis, Eindhoven University of Technology, Dept. of Mechanical Engineering, 2000. SE 420225.
- [42] Wind River Systems, Inc. *VxWorks, Programmer’s Guide*, first edition, 1997. VxWorks version 5.3.1.