A HYBRID LANGUAGE FOR MODELING, SIMULATION AND VERIFICATION

R.R.H. Schiffelers [*] **D.A. van Beek** [*] **K.L. Man** [**]
M.A. Reniers [**] **J.E. Rooda** [*]

[*] *Department of Mechanical Engineering*
[**] *Department of Mathematics and Computer Science*
Eindhoven University of Technology, P.O.Box 513, 5600 MB
Eindhoven, The Netherlands
{r.r.h.schiffelers, d.a.v.beek, k.l.man,
m.a.reniers,j.e.rooda}@tue.nl

Abstract: The χ language is a hybrid language for modeling, simulation and verification. As a result of the recently completed formal semantics, the language now consists of a number of orthogonal operators that operate on all process terms, including differential algebraic equations. The same χ model can be used for simulation and verification. Verification is possible after a straightforward syntactical translation of the model. Simulation related information is clearly separated from the other statements. *Copyright © 2003 IFAC*

Keywords: Simulation, Simulation Languages, Formal Verification, Formal Languages.

1. INTRODUCTION

Currently, there is a gap between simulation languages and verification formalisms. The χ language attempts to bridge this gap. The language was designed as a hybrid modeling and simulation language. The χ language and simulator have been successfully applied to a large number of industrial cases, such as an integrated circuit manufacturing plant, a brewery, and process industry plants (van Beek *et al.*, 2002). The formalization of the χ simulation language resulted in the χ_{σ_h} language. The language semantics of χ_{σ_h} are formally defined using a structured operational semantics (SOS) and a number of associated functions (Schiffelers *et al.*, 2003). A formal semantics is a prerequisite for reasoning about models as it unambiguously defines the meaning of the models and hence increases the understanding of these. It also helps in the development of the language itself, the construction of tools (e.g. the simulator), and the development of verification techniques, by clearly separating the issues of meaning, and implementation of this meaning into tools. As a result of the formalization of the χ language, the language is now a process algebra with orthogonal operators. Instead of defining a number of complex language elements such as selection $[b_0 \to S_0 \ [] \ \ldots \ [] \ b_\mathrm{n} \to S_\mathrm{n}]$ and selective waiting $[b_0; Ev_0 \to S_0 \ [] \ \ldots \ [] \ b_\mathrm{n}; Ev_\mathrm{n} \to S_\mathrm{n}]$ (van Beek and Rooda, 2000), the language is now defined by means of a small number of orthogonal operators that operate on all process terms and do not overlap. The formal semantics, the χ simulator (Fábián, 1999) and the model checker for discrete-event part of χ (Bos and Kleijn, 2002) are the basis of the new χ simulation and χ_{σ_h} verification tools that will be developed. The relation between χ and χ_{σ_h} is illustrated in Figure 1. A χ_{σ_h} process can be simulated and properties can be verified. A χ experiment can also be compiled directly to obtain a simulator. Reasons for this can be to gain speed, or to make it easier to provide user friendly error information related to the χ specification. Unlike most other hybrid formalisms such as hybrid automata (Alur *et al.*, 1995) or Petri nets (David and Alla, 2001), differential *algebraic* equations (DAEs) are fully supported by χ, including higher index DAEs (Fábián *et al.*, 2001). The χ language can also deal with purely

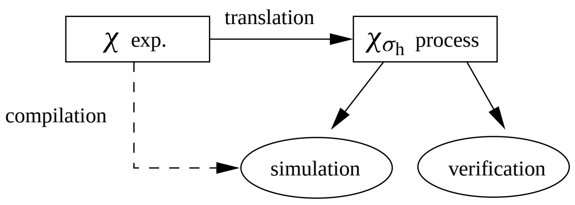

Fig. 1. From χ to χ_{σ_h}.

discrete-event systems, and hybrid systems that can dynamically change from lower to higher index, and vice versa, such as treated in (Mosterman and Ciolfi, 2002). Section 2 and 3 describe the χ (modeling) language, and its relation to other simulation languages. The syntax of the χ language is presented in Section 4. The translation from the modeling language to the formal language is explained in Section 5 and illustrated by means of an example in Section 6.

2. INTEGRATION OF DISCRETE AND CONTINUOUS TIME BEHAVIOR

A simulation run of a hybrid model is an alternating sequence of discrete and continuous phases. Hybrid languages, generally make a distinction between the description of discrete behavior and continuous time behavior. For instance, in gPROMS (Barton, 1992), tasks are used to describe the discrete manipulations that a schedule may impose on the underlying continuous model. The continuous model is specified in an equation section and consists of variables and (conditional) equations only. The stream section is used to connect models and form hierarchical models. In VHDL-AMS (IEEE, 1999), there are three types of statements; sequential statements which define algorithms for the execution of a subprogram or process; they execute in the order in which they appear. Concurrent statements are used to define interconnected blocks and processes. Simultaneous statements express explicit differential and algebraic equations. Concurrent statements together with simultaneous statements describe the overall model. In Modelica 2.0 (Modelica Association, 2002), statements are executed in an algorithm section and equations are specified in an equation section. Connections between objects are introduced by the connection statement in the equation part.

One of the most important concepts of χ is that of a process. Usually, a single entity of a production system is modeled by one process, for instance a buffer or a machine. In order to support structural design of production systems, groups of related entities can be modeled by processes as aggregates of concurrent processes. To this end, χ incorporates parallel composition of processes. Given that in production systems many activities happen at the same time, concurrent processes are very well suited to model these systems. The χ language is a process algebra with an interleaving concurrency semantics, which enables us to use existing process algebra techniques for the analysis of production systems.

In χ, there is no distinction between a discrete part and a continuous part of a model, since equations and statements are both process terms. The operators can be applied orthogonally to all process terms. For instance, equations can be added and removed dynamically using the alternative composition operator. Shared variables are used instead of connection equations. Using the scope operator, processes can be created and removed dynamically.

3. BEHAVIORAL AND SIMULATION STATEMENTS

In a continuous phase, model time advances, and the values of continuous variables are determined by the equations as a function of time. Since most of the sets of equations cannot be solved symbolically, most simulators are equipped with one or more numerical solvers. Numerical solution methods usually require an absolute and a relative tolerance for each unknown, and sometimes an initial guess value in order to start the iteration. This solver specific information can be specified at the type declaration (gPROMS, Modelica), or by the definition of tolerance groups (VHDL-AMS). During simulation, reinitialization of continuous variables may be necessary. Using an ideal solver, no initial guesses need to be provided for the initialization iteration. However, since such an ideal solver does not exist, sometimes the modeler has to provide values for some variables in order to start the iteration, so called initial guesses. These initial guesses can be changed during simulation using several language elements see for example (gPROMS, Modelica and VHDL-AMS).

In χ, the simblock is used for the specification of simulation related information. For this purpose, a variable of type real is extended with an attribute mag, which specifies the order of magnitude of the variable. The attribute mag has a default value of 1.0. The declarations, assignment statements and communication statements (send and receive) may be postfixed by a simblock. The statements of a simblock are executed when the declarations or the statement to which the simblock is postfixed is processed. Using the simblock, the simulation related statements are syntactically separated from the statements which describe the (behavioral) model, which improves the readability. It may be obvious that a simblock may not contain modeling information. For instance, consider the following equation: $x^2 = 1$. This equation has two solutions $x = 1$ and $x = -1$. In general, a numerical solver returns one solution. Pointing the solver to one particular solution by means of initial guesses should be avoided. Otherwise, removing the simblocks (for verification purposes) leads to different model behavior.

4. SYNTAX DEFINITION OF χ

In this section, the syntax of the χ language is introduced in an extended BNF-like notation.

A process definition has syntax

$$PD ::= \textbf{proc}\ p_\mathrm{i}\ \text{‘('}\ D_\mathrm{f}\ \text{‘)'}\ =\ \text{‘}[\![\text{'}\ D\ [S]\ \text{‘|'}\ P\ \text{‘}]\!]\text{'}$$

where p_i denotes a process identifier, D_f and D denote declarations, S denotes a simblock, and P denotes a process term. Note that $[S]$ denotes zero or one occurrence of the nonterminal S.

The declaration of the formal parameters D_f of a process definition has the following syntax, where *vis* is a comma separated list of variable identifiers, *cis* is a comma separated list of channel identifiers, and t is a type.

$$\begin{aligned} D_\mathrm{f}\ ::=\ & \mathit{vis}\ \text{‘:'}\ t \quad \mid \mathit{cis}\ \text{‘: !'}\ t \mid \mathit{cis}\ \text{‘: ?'}\ t \\ \mid\ & \textbf{ext}\ \mathit{vis}\ \text{‘:'}\ t \mid D_\mathrm{f}\ \text{‘,'}\ D_\mathrm{f} \end{aligned}$$

The declaration *vis* : t denotes the declaration of variables of type t; *cis* : $!t$ and *cis* : $?t$ denote the declaration of channels *cis* of type t used for sending or receiving, respectively; and **ext** *vis* : t declares *vis* as external (shared) variables of type t.

The declaration of variables and channels in a process definition has the following syntax. These variables and channels are by definition local, and cannot be used outside the process in which they are declared:

$$\begin{aligned} D\ ::=\ & \textbf{var}\ \mathit{vis}\ \text{‘:'}\ t\ \{\text{'='}\ c\} \\ \mid\ & \textbf{cont}\ \mathit{vis}\ \text{‘:'}\ t\ \{\text{'='}\ c\} \\ \mid\ & \textbf{chan}\ \mathit{cis}\ \text{‘:'}\ t \mid D\ \text{‘,'}\ D \end{aligned}$$

where c is a constant expression. The declarations **var** *vis* : t, **cont** *vis* : t and **chan** *cis* : t denote the declaration of discrete variables, continuous variables and channels of type t respectively. Optionally, variables can be initialized at their declaration.

At the declaration of local variables, simulation attributes can be postfixed optionally by means of a simblock S. The purpose of the simblock is explained in Section 3. The simblock has syntax:

$$\begin{aligned} S\ &::=\ \text{‘ /\$ '}\ \mathit{SimStat}\ \text{‘ \$/ '} \\ \mathit{SimStat}\ &::=\ \mathit{vi}\ \text{‘:='}\ e_r \\ &\quad\mid \mathit{vi}\ \text{‘.'}\ \textbf{mag}\ \text{‘:='}\ e_r \\ &\quad\mid \mathit{SimStat}\ \text{‘,'}\ \mathit{SimStat} \end{aligned}$$

where *SimStat* denotes a simulation statement, and e_r denotes a real-valued expression.

The body of a χ process definition consists of a process term (statements). A subset of the basic building blocks of χ process terms are introduced below.

- Assignment process terms of the form $vi := e$. The value of the expression e is assigned to variable vi.
- Send and receive process terms of the form $m!e$, and $m?vi$, respectively, are relevant only when used in parallel composition. A send action $m!e$ blocks until a receive action on the same channel m ($m?vi$) can be executed, and vice versa. The parallel composition (defined below) $m!e \parallel m?vi$ assigns the value of expression e to variable vi. This behavior is known as synchronous communication.
- Equation process terms of the form eq, where eq denotes an equation. Equations eq are solved for the continuous variables.
- Delta process terms of the form Δe_r, where e_r represents a real-valued expression. Delta statements terminate by means of an internal action after e_r time units.
- Nabla process terms of the form ∇b_n, where b_n represents a boolean variable or a comparison of real-valued expressions using $\leq$, or $\geq$. The nabla process ∇b_n terminates by means of an internal action if b_n is *true*.

Summarizing, in an extended BNF-like notation, the atomic process terms are the following:

$$\begin{array}{rll} AP\ ::= & vi\ \text{‘:='}\ e\ [S] & \text{Assignment} \\ \mid & ci\ \text{‘!'}\ e\ [S] & \text{Sending} \\ \mid & ci\ \text{‘?'}\ vi\ [S] & \text{Receiving} \\ \mid & eq & \text{Equation} \\ \mid & \text{‘}\Delta\text{'}\ e_r & \text{Delta} \\ \mid & \text{‘}\nabla\text{'}\ b_\mathrm{n} & \text{Nabla} \end{array}$$

Process terms are built from atomic process terms and boolean expressions using operators for guarding ($\rightarrow$), sequential composition ($;$), alternative composition ($[\!]$), parallel composition ($\parallel$), reinitialization ($\gg$), repetition ($*$), the scope operator $[\![\ \mid\ ]\!]$, and a process instantiation $l_\mathrm{i} : p_\mathrm{i}(d_\mathrm{a})$.

$$\begin{aligned} P\ ::=\ & AP \quad \mid b\ \text{‘}\rightarrow\text{'}\ P \mid P\ \text{‘;'}\ P \mid P\ \text{‘}[\!]\text{'}\ P \\ \mid\ & P\ \text{‘}\parallel\text{'}\ P \mid \text{‘}*\text{'}\ P \quad \mid i\ \text{‘}\gg\text{'}\ P \\ \mid\ & [l_\mathrm{i}\ \text{‘:'}]\ \text{‘}[\![\text{'}D\ [S]\ \text{‘|'}\ P\ \text{‘}]\!]\text{'} \mid l_\mathrm{i}\ \text{‘:'}\ p_\mathrm{i}\text{‘('}d_\mathrm{a}\text{‘)'} \end{aligned}$$

where b represents a boolean expression, and i a non-empty comma separated list of reinitialization equations. A guarded process term $b \rightarrow p$ can do whatever p can do when b is initially true. When b is false, $b \rightarrow p$ deadlocks. An alternative composition process term $p\ [\!]\ q$ is reduced to p' when p executes an action to p' (likewise for q). A parallel composition process term $p \parallel q$ becomes $p' \parallel q$ when p executes an action to p' (likewise for q). The delay behavior of $p\ [\!]\ q$ and $p \parallel q$ is equivalent: p and q delay simultaneously to $p'\ [\!]\ q'$ and $p' \parallel q'$, respectively. The reinitialization process term $i \gg p$ can perform an action or a delay if p can perform that action or delay from a state in which the variables are reinitialized according to i. A scope process term $[\![\ d \mid p\ ]\!]$, where $d \in D$, is used to declare a local scope. The declared variables and channels in d are not visible outside this scope. A process instantiation, $l_\mathrm{i} :\ p_\mathrm{i}(d_\mathrm{a})$, where l_i denotes a label identifier, p_i refers to a process definition, and d_a denotes a comma separated list of expressions (actual parameters), is used to instantiate process definitions. It is assumed that instantiation is finite and therefore recursive process instantiations are not allowed. The operators are listed in descending order of their binding strength as follows:

$$\{*,\quad \gg,\quad ;\,,\quad \rightarrow\},\qquad \{[],\quad \parallel\},$$

where the repetition operator, the reinitialization operator, the sequential composition operator and the guarded operator bind equally strong. Also, the alternative composition operator [] and the parallel composition operator $\parallel$ bind equally strong. A simulation experiment is specified as follows:

$$XPER \ ::= \ \text{'xper '}\ p_{\mathrm{i}}\text{'('}d_{\mathrm{ax}}\text{')'}$$

where the actual parameter list d_{ax} may contain constant expressions only.

5. TRANSLATION OF χ INTO $\chi_{\sigma_{\mathrm{h}}}$

The behavior of a $\chi_{\sigma_{\mathrm{h}}}$ process depends, among others, on the values of the discrete and continuous variables. A simplified $\chi_{\sigma_{\mathrm{h}}}$ process $\langle p, \sigma, \Gamma\rangle$ (for the complete process definition see (Schiffelers *et al.*, 2003)) is therefore a $\chi_{\sigma_{\mathrm{h}}}$ process term $p \in P$, combined with a valuation $\sigma \in V \mapsto \Sigma$, where V denotes the set of all variables and Σ denotes the set of all values, and a set of continuous variables $\Gamma \subseteq V$. The $\chi_{\sigma_{\mathrm{h}}}$ language contains all process terms and operators of χ, apart from the process and experiment instantiations, which need to be translated as shown in the remainder of this section. Simblocks do not exist in the formal semantics. They only have a meaning for simulations. Therefore, in the translation of process term P, the simblock disappears. Because of the fact that simblocks are attached to the declaration block and statements, and thus execute atomically with them, they do not introduce new transitions. Therefore, removing the simblocks in the translation of a χ specification to a $\chi_{\sigma_{\mathrm{h}}}$ specification does not change the semantics of the model. In $\chi_{\sigma_{\mathrm{h}}}$, there are two additional operators: the encapsulation operator ∂ which is used to enforce synchronous communication behavior by excluding asynchronous actions, and the maximum progress operator π which adds priority of action transitions over time transitions. The remainder of this section describes the translation of a χ specification into a $\chi_{\sigma_{\mathrm{h}}}$ specification, which is illustrated in Fig. 1.

The translation is defined as a function from syntactic entities in χ to syntactic entities in $\chi_{\sigma_{\mathrm{h}}}$: $\mathcal{T} : \chi \rightarrow \chi_{\sigma_{\mathrm{h}}}$.

$$\begin{array}{l}
\mathcal{T}(\text{xper } p_{\mathrm{i}}(d_{\mathrm{ax}})) = \\
\langle \pi(\partial(\dot{\tau} = 1 \parallel \mathrm{pt}(\mathcal{T}(\mathrm{body}(p_{\mathrm{i}}(d_{\mathrm{ax}})))))) \\
, \{\tau \mapsto 0\} \cup \mathrm{val}(\mathcal{T}(\mathrm{body}(p_{\mathrm{i}}(d_{\mathrm{ax}})))) \\
, \{\tau\} \cup \mathrm{cont}(\mathcal{T}(\mathrm{body}(p_{\mathrm{i}}(d_{\mathrm{ax}})))) \\
\rangle \\
\mathcal{T}(l :\ [\![\ \text{var } x : t_0 = c_0, \text{cont } y : t_1, \text{chan } m : t_2 \mid P\ ]\!]) \\
= [\![\ \{x \mapsto c_0, y \mapsto \perp\}, \{y\} | \mathcal{T}(P)\ ]\!]
\end{array}$$

$$\begin{array}{lll}
\mathcal{T}(l :\ p_{\mathrm{i}}(d_{\mathrm{a}})) & = & \mathcal{T}(\mathrm{body}(l :\ p_{\mathrm{i}}(d_{\mathrm{a}})) \\
\mathcal{T}(P[S]) & = & \mathcal{T}(P) \\
\mathcal{T}(P) & = & P \quad \text{otherwise}
\end{array}$$

where the functions pt : $\chi_{\sigma_{\mathrm{h}}} \rightarrow \chi_{\sigma_{\mathrm{h}}}$, val : $\chi_{\sigma_{\mathrm{h}}} \rightarrow V \mapsto \Sigma$ and cont : $\chi_{\sigma_{\mathrm{h}}} \rightarrow \mathcal{P}(V)$ return the process term, a valuation and the set of continuous variables respectively:

$$\begin{array}{ll}
\mathrm{val}([\![\ \sigma, \Gamma \mid P\ ]\!]) & = \sigma \\
\mathrm{cont}([\![\ \sigma, \Gamma \mid P\ ]\!]) & = \Gamma \\
\mathrm{pt}([\![\ \sigma, \Gamma \mid P\ ]\!]) & = P
\end{array}$$

The reserved variable τ which is not explicitly defined in a χ specification is made explicit in the accompanying $\chi_{\sigma_{\mathrm{h}}}$ process.

The function body : $\chi \rightarrow \chi$, when applied to a process instantiation, replaces this instantiation by the corresponding process body, such that the variables declared in the formal parameter list are substituted by the corresponding actual parameters of the process instantiation. The application of the function body is illustrated by means of an example:

$$\begin{array}{l}
\text{proc } p_{\mathrm{i}}(m : ?t_0, \text{ext } y : t_2, x : t_1) = \\
[\![\ \text{var } a : t_3 = c_1, \text{cont } b : t_4 \\
\mid \ \mathrm{P} \\
]\!]
\end{array}$$

The function body applied to a process instantiation $l_{\mathrm{i}} :\ p_{\mathrm{i}}(n, w, e_0)$ —where n is a channel identifier, w is a variable identifier, and e_0 an expression of type t_1— results in the following χ process body:

$$\begin{array}{l}
\mathrm{body}(l_{\mathrm{i}} :\ p_{\mathrm{i}}(n, w, e_0)) = \\
\quad [\![\ \text{var } l_{\mathrm{i}}.x : t_1, \text{var } l_{\mathrm{i}}.a : t_3 = c_1, \text{cont } l_{\mathrm{i}}.b : t_4 \\
\quad \mid\ l_{\mathrm{i}}^{+} = e_0 \gg \mathrm{P}[n, w, l_{\mathrm{i}}.x, l_{\mathrm{i}}.a, l_{\mathrm{i}}.b/m, y, x, a, b] \\
\quad]\!]
\end{array}$$

where $P[n, w, l_{\mathrm{i}}.x, l_{\mathrm{i}}.a, l_{\mathrm{i}}.b/m, y, x, a, b]$ denotes the syntactic substitution of n, w, $l_{\mathrm{i}}.x$, $l_{\mathrm{i}}.a$, $l_{\mathrm{i}}.b$ for m, y, x, a, b, respectively, in process term P. The function body applied to a process instantiation renames the variables and channels which are declared in the process definition using the label of the instantiation. Renaming is needed to allow an external variable/channel, used in the actual parameterlist (d_{a}) of a process instantiation, to have the same name as a local variable/channel in that process. E.g. $l_{\mathrm{i}} :\ p_{\mathrm{i}}(n, a, e_0)$ in the previous example. Here, identifier a refers to an external variable, while there is also a local variable a in the definition of process p_{i}. It is assumed that the expressions are type-correct, and that identifiers are not declared more than once in the same scope. This translation scheme is illustrated by means of an example in Section 6.

6. CONVEYOR EXAMPLE

In this section, the translation between the χ modeling language and the $\chi_{\sigma_{\mathrm{h}}}$ verification language is illustrated by means of an example. Fig. 2 shows a conveyor system that is used for transportation and buffering of boxes. It consists of a line of conveyor belts driven by motors. Each conveyor belt is equipped with a sensor

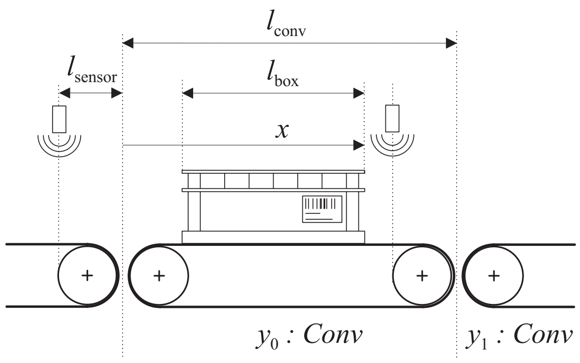

Fig. 2. The conveyor system.

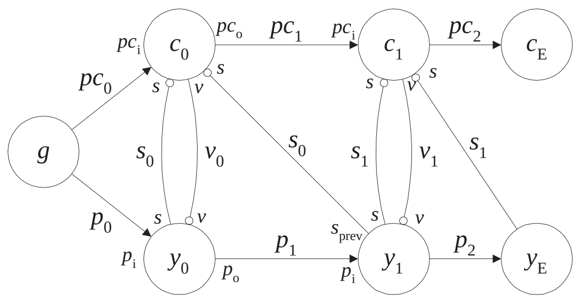

Fig. 3. Iconic model of the conveyor system.

(represented in the figure by a small rectangle), that detects the presence of a box.

Fig. 3 shows the iconic model of a generator g, two conveyor belts (y_0, y_1) and the associated control processes (c_0, c_1). Processes y_E and c_E are added to obtain a closed system; they do not model actual behavior. The model is a simplified version of the model treated in (van Beek *et al.*, 1997). The lines with arrow heads represent synchronization channels ($pc_0, pc_1, pc_2, p_0, p_1, p_2$), the lines ending in a small circle represent shared variables (s_0, s_1, v_0, v_1). The specification of the model follows at the end of this section.

Process S is a textual specification of the iconic model from Fig. 3. Channels $pc_0, pc_1, \ldots, p_2$ are of type nat, which means that they are used to communicate naturals (box numbers). Variables s_0, s_1, v_0, v_1 are declared in process S because they are shared variables. Variables are either continuous or discrete. Continuous variables are declared using the cont keyword (e.g. cont x : real); they are the unknowns in the equations. Discrete variables are declared using the var keyword (e.g. var v_0, v_1 : real); their value is determined by assignment statements only (e.g. $v := 0$). Behind the bar "|", the two control processes and conveyor processes are specified. The control processes c_0 and c_1 (of type C) consist of an infinite loop ($*(\ldots)$). First, the conveyor is switched on: $v := 1$, where v is the velocity of the conveyor belt. The process subsequently waits until the sensor is off ($\nabla\neg s$, where $\neg$ means logical not). Initially, the conveyor is empty and the sensor is off. The process then waits until it can synchronize with the preceding control process by executing $pc_i?box$. This means that a box may enter the conveyor. Subsequently, the process waits until the box has reached the sensor position so that the sensor is on (value of s equals *true*). Then the conveyor is switched off ($\nabla s;\ v := 0$). Subsequently, the control process tries to synchronize with the next control process ($pc_o!box$). When the next control process executes the corresponding synchronization statement ($pc_i?box$), the repetition is re-executed, and the conveyor is switched on again ($v := 1$). Processes y_0 and y_1 (of type *Conv*) model the physical behavior of the conveyor belts. The position x of the front of the box is modeled by means of an ordinary differential equation ($\dot{x} = v$). When a box crosses the boundary of two conveyors y_0 and y_1, the box is transported from process y_0 to y_1 by means of a synchronization via channel p_1. From that point on, the position of the box is registered in process y_1 instead y_0. When a box enters a conveyor ($p_i?box$), a new continuous variable x with initial value of 0 is created by means of the scope operator $|[\text{cont } x : \text{real} = 0 \mid \cdots]|$. The position of the box is defined by equation $\dot{x} = v$ until $x \geq l_{\text{box}} - l_s$, where l_s represents l_{sensor} (see Fig. 2, where $l_{\text{conv}} = 20$, $l_{\text{box}} = 10$). When $x = l_{\text{box}} - l_s$, the rear of the box has just passed the sensor of the previous conveyor. Subsequently, the sensor of the previous conveyor (s_{prev}) is switched off. After that, the position of the box is again defined by equation $\dot{x} = v$, now until $x \geq l_{\text{conv}} - l_s$. Then, the sensor of the conveyor itself is switched on ($s := true$). Finally, when the box has reached the end of the conveyor ($x = l_{\text{conv}}$), the box is sent to the next conveyor ($p_o!box$), and the loop is re-executed. The χ_h specification of the conveyor system is given below ($l_{\text{conv}} = 20$, $l_{\text{box}} = 10$):

```
proc  $C(pc_i : ?\text{nat},\ pc_o : !\text{nat},\ \text{ext } v : \text{real}$
        $,\ \text{ext } s : \text{bool}$
        $) =$
$|[$ var $box$ : nat
$|\ *(v := 1;\ \nabla\neg s;\ pc_i?box;\ \nabla s;\ v := 0;\ pc_o!box)$
$]|$
```

```
proc  $Conv(p_i : ?\text{nat},\ p_o : !\text{nat},\ \text{ext } s_{\text{prev}}, s : \text{bool}$
           $,\ \text{ext } v : \text{real},\ l_s : \text{real}$
           $) =$
$|[$ var $box$ : nat
$|\ *(p_i?box$
    $;\ |[\ \text{cont } x : \text{real} = 0$
      $|\ (\dot{x} = v\ []\ \nabla x \leq 10 - l_s);\ s_{\text{prev}} := false$
      $;\ (\dot{x} = v\ []\ \nabla x \leq 20 - l_s);\ s := true$
      $;\ (\dot{x} = v\ []\ \nabla x \leq 20);\ p_o!box$
      $]|$
    $)$
$]|$
```

```
proc  $S\ (l_s : \text{real}) =$
$|[$ chan $pc_0, pc_1, pc_2, p_0, p_1, p_2 : \text{nat},\ \text{var } s_G : \text{bool}$
$,\ \text{var } s_0, s_1 : \text{bool} = false,\ \text{var } v_0, v_1 : \text{real}$
$|\ g : |[\ \text{var } box : \text{nat} = 0$
      $|\ *(pc_0!box;\ p_0!box;\ box := box + 1)$
      $]|$
$\|\ c_0 : C(pc_0, pc_1, v_0, s_0)$
```

```
$\| \; c_1 : C(pc_1, pc_2, v_1, s_1)$
$\| \; c_\mathrm{E} : \llbracket \; \mathsf{var}\; box : \mathrm{nat} \mid *pc_2?box \; \rrbracket$
$\| \; y_0 : Conv(p_0, p_1, s_\mathrm{G}, s_0, v_0, l_\mathrm{s})$
$\| \; y_1 : Conv(p_1, p_2, s_0, s_1, v_1, l_\mathrm{s})$
$\| \; y_\mathrm{E} : \llbracket \; \mathsf{var}\; box : \mathrm{nat}$
        $\mid *(p_2?box;\; \Delta(10 - l_\mathrm{s})/1;\; s_1 := false)$
        $\rrbracket$
$\rrbracket$
```

xper $S(2)$

The experiment xper $S(2)$ is translated into a χ_{σ_h} process as follows:

```
$\mathcal{T}(\mathrm{xper}\; S(2)) =$
$\langle \pi(\partial(l_\mathrm{s}^+ = 2 \gg (\dot{\tau} = 1$
$\| \; \llbracket \; \{box \mapsto 0\}, \emptyset$
  $\mid *(pc_0!box;\; p_0!box;\; box := box + 1)$
  $\rrbracket$
$\| \; \llbracket \; \{c_0.box \mapsto \perp\}, \emptyset$
  $\mid *(v_0 := 1;\; \nabla\neg s_0;\; pc_0?c_0.box$
      $;\; \nabla s_0;\; v_0 := 0;\; pc_1!c_0.box$
      $)$
  $\rrbracket$
$\| \; \llbracket \; \{c_1.box \mapsto \perp\}, \emptyset$
  $\mid *(v_1 := 1;\; \nabla s_1;\; pc_1?c_1.box$
      $;\; \nabla s_1;\; v_1 := 0;\; pc_2!c_1.box$
      $)$
  $\rrbracket$
$\| \; \llbracket \; \{box \mapsto \perp\}, \emptyset$
  $\mid *(pc_2?box)$
  $\rrbracket$
$\| \; \llbracket \; \{y_0.box \mapsto \perp,\; y_0.l_\mathrm{s} \mapsto \perp\}, \emptyset$
  $\mid \; y_0.l_\mathrm{s}^+ = l_\mathrm{s} \gg$
  $*(p_0?y_0.box$
    $;\; \llbracket \; \{y_0.x \mapsto 0\}, \{y0.x\}$
      $\mid \; (y_0.\dot{x} = v_0 \;[]\; \nabla y_0.x \leq 10 - y_0.l_\mathrm{s});\; s_\mathrm{G} := false$
      $;\; (y_0.\dot{x} = v_0 \;[]\; \nabla y_0.x \leq 20 - y_0.l_\mathrm{s});\; s_0 := true$
      $;\; (y_0.\dot{x} = v_0 \;[]\; \nabla y_0.x \leq 20);\; p_1!y_0.box$
      $\rrbracket$
    $)$
  $\rrbracket$
$\| \; \llbracket \; \{y_1.box \mapsto \perp,\; y_1.l_\mathrm{s} \mapsto \perp\}, \emptyset$
  $\mid \; y_1.l_\mathrm{s}^+ = l_\mathrm{s} \gg$
  $*(p_1?y_1.box$
    $;\; \llbracket \; \{y_1.x \mapsto 0\}, \{y_1.x\}$
      $\mid \; (y_1.\dot{x} = v_1 \;[]\; \nabla y_1.x \leq 10 - y_1.l_\mathrm{s});\; s_0 := false$
      $;\; (y_1.\dot{x} = v_1 \;[]\; \nabla y_1.x \leq 20 - y_1.l_\mathrm{s});\; s_1 := true$
      $;\; (y_1.\dot{x} = v_1 \;[]\; \nabla y_1.x \leq 20);\; p_2!y_1.box$
      $\rrbracket$
    $)$
  $\rrbracket$
$\| \; \llbracket \; \{box \mapsto \perp\}, \emptyset$
  $\mid *(p_2?box;\; \Delta(10 - l_\mathrm{s})/1;\; s_1 := false)$
  $\rrbracket)))$
$,\; \{\tau \mapsto 0,\; s_\mathrm{G} \mapsto \perp,\; s_0 \mapsto false,\; s_1 \mapsto false$
$,\; v_0 \mapsto \perp,\; v_1 \mapsto \perp,\; l_\mathrm{s} \mapsto \perp\}, \{\tau\}\rangle$
```

7. CONCLUSIONS

Using the newly defined χ modeling language, hybrid χ models can be simulated and model properties can be verified. A straightforward translation from χ models to χ_{σ_h} models has been defined for the purpose of verification. Using equational reasoning/theorem proving, transformation rules can be derived. From a χ_{σ_h} specification, a transition system can be generated, and properties specified in some logic can be proven using a model checker. Currently, new tools for simulation, translation and verification are developed.

REFERENCES

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

Barton, P. I. (1992). The Modelling and Simulation of Combined Discrete/Continuous Processes. PhD thesis. University of London.

Bos, V. and J.J.T. Kleijn (2002). Formal Specification and Analysis of Industrial Systems. PhD thesis. Eindhoven University of Technology.

David, R. and H. Alla (2001). On hybrid Petri nets. *Discrete Event Dynamic Systems: Theory & Applications* **11**(1-2), 9–40.

Fábián, G. (1999). A Language and Simulator for Hybrid Systems. PhD thesis. Eindhoven University of Technology.

Fábián, G., D. A. van Beek and J. E. Rooda (2001). Index reduction and discontinuity handling using substitute equations. *Mathematical and Computer Modelling of Dynamical Systems* **7**(2), 173–187.

IEEE (1999). *IEEE Standard VHDL Analog and Mixed-Signal Extensions (IEEE Std 1076.1-1999)*. IEEE. New York.

Modelica Association (2002). *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling*. http://www.modelica.org.

Mosterman, Pieter J. and John E. Ciolfi (2002). Embedded code generation for efficient reinitialization. In: *15th Triennial World Congress of the International Federation of Automatic Control*. CD-ROM.

Schiffelers, R.R.H., D.A. van Beek, K.L. Man, M.A. Reniers and J.E. Rooda (2003). Formal semantics of hybrid Chi. Eindhoven University of Technology. http://se.wtb.tue.nl.

van Beek, D. A., A. van den Ham and J. E. Rooda (2002). Modelling and control of process industry batch production systems. In: *15th Triennial World Congress of the International Federation of Automatic Control*. Barcelona. CD-ROM.

van Beek, D. A. and J. E. Rooda (2000). Languages and applications in hybrid modelling and simulation: Positioning of Chi. *Control Engineering Practice* **8**(1), 81–91.

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