Proceedings of the 2006 IEEE
Conference on Computer Aided Control Systems Design
Munich, Germany, October 4-6, 2006

Deriving Simulators for Hybrid Chi Models

D.A. van Beek, K.L. Man, M.A. Reniers, J.E. Rooda, and R.R.H. Schiffelers

Abstract— **The hybrid Chi language is a formalism for modeling, simulation and verification of hybrid systems. The formal semantics of hybrid Chi allows the definition of provably correct implementations for simulation, verification and real-time control. This paper discusses the principles of deriving an implementation for simulation and verification directly from the semantics, and presents an implementation based on a symbolic solver. The simulator is illustrated by means of a case study.**

I. INTRODUCTION

Hybrid systems related research is based on two, originally different, world views: on the one hand the dynamics and control (DC) world view, and on the other hand the computer science (CS) world view. Clearly, hybrid systems represent a domain where the DC and CS world views meet, and we believe that a formalism that integrates the DC and CS world views is a valuable contribution towards integration of the DC and CS methods, techniques, and tools. The hybrid χ (hybrid Chi) formalism [1], [2] is such a formalism. On the one hand, it can deal with continuous-time systems, such as piecewise affine (PWA) systems, mixed logic dynamical (MLD) systems or linear complementarity (LC) systems, and hybrid systems based on sets of ordinary differential equations (ODEs) using discontinuous functions in combination with algebraic constraints (the DC approach). On the other hand, it can deal with discrete-event systems, without continuous variables or differential equations, and with hybrid systems in which discontinuities take place (mainly) by means of actions (the CS approach). The intended use of hybrid χ is for modeling, simulation, verification, and real-time control. Its application domain ranges from physical phenomena, such as dry friction, to large and complex manufacturing systems.

The history of the χ formalism dates back quite some time. It was originally designed as a modeling and simulation language for specification of discrete-event, continuous-time or combined discrete-event/continuous-time models. The first simulator [3], however, was suited to discrete-event models only. The simulator was successfully applied to a large number of industrial cases, such as an integrated circuit manufacturing plant, a brewery, and process industry plants [4]. For the purpose of verification, the discrete-event part of the language was mapped onto the process algebra χ_σ [5] by means of a syntactical translation. The semantics of χ_σ was defined using a structured operational semantics style (SOS), bisimulation relations were derived, and a model checker was built. In this way, verification of discrete-event χ models was made possible [6]. The hybrid language and simulator were developed in [7], [8], [9]. The hybrid χ formalism defined in [1], [2] integrates the modeling language and the verification formalism. It resulted in a process algebra with a relatively straightforward and elegant syntax and semantics that is also highly suited to electrical and mechanical engineers.

When comparing the χ language to other languages for the analysis of hybrid systems, without going into detail, a distinction can be made between simulation languages and verification formalisms. Simulation languages, such as Matlab / Simulink [10] and Modelica [11], focus on ease of use. The main purpose of these languages is to help understand the dynamic behavior of systems by means of modeling and simulation.

Verification formalisms on the other hand, are 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. These languages are often based on hybrid automata [12], but process algebras [13], [14] and Petri nets [15], [16] are also used.

Where simulation languages are usually large, with many different modeling elements, verification formalisms are usually quite small. Since simulation languages do not have a formal semantics, they cannot be used for formal verification of model properties. The χ language aims at combining the advantages of simulation languages and verification formalisms. Due to the orthogonal design of the χ language and careful selection of the language primitives, the core of the language is small. The additional syntactical extensions provide the modeler with expressive power, while still allowing verification. For a more detailed comparison of the χ language with verification formalisms, see [1], [2]. A more detailed comparison of the χ language with simulation languages can be found in [9].

The semantics of hybrid χ is defined using a structured operational semantics style (SOS) [17]. In this paper, we describe the derivation of an implementation from these deduction rules, called the stepper. This stepper is the basis for implementations for simulation and verification. On the one hand, the stepper is closely related to the structured operational semantics of χ, and on the other hand, the stepper can relatively easily be embedded in a simulation or verification implementation. So far, a χ simulator has been defined and implemented, based on the stepper in combination with a

Work partially done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368

D.A. van Beek, R.R.H. Schiffelers, and J.E. Rooda are with the Department of Mechanical Engineering, Eindhoven University of Technology, P.O. Box 513 5600 MB Eindhoven, The Netherlands, {d.a.v.beek, r.r.h.schiffelers, j.e. Rooda}@tue.nl

K.L. Man and M.A. Reniers are with the Department of Computer Science, Eindhoven University of Technology, P.O. Box 513 5600 MB Eindhoven, The Netherlands {k.l.man, m.a.reniers}@tue.nl

0-7803-9797-5/06/$20.00 ©2006 IEEE

symbolic solver. For this prototype implementation, the focus was to stay very close to the definition of the simulator, instead of focusing on efficiency/speed. We do not describe the actual (Python [18]) implementation of the simulator in detail, but confine ourselves to its theoretical foundations and its architecture.

The paper is organized as follows. First, in Section II the hybrid χ language is introduced. In Section III, the stepper is formally defined. The symbolic simulator is described in Section IV. In Section V, the integration between the χ tool set and third party tools is briefly discussed. In Section VI, the simulator is illustrated by means of a case study. Concluding remarks are given in Section VII.

II. The hybrid χ language

In this section, the syntax and semantics of (a representative subset of) the hybrid χ language, called χ_{sub}, are discussed. The main differences between the χ_{sub} language and the complete χ language is that the scoping operators and many syntactical extensions have been omitted in χ_{sub}. Scoping operators introduce local scopes for variables, channels and/or recursion definitions. The syntactical extensions introduce user friendly notations, such as the delay statement for time passing, that are defined by rewriting them into other, more primitive, syntax. A more detailed explanation of hybrid χ can be found in [1], [2]. In the rest of this paper, we simply speak of χ instead of hybrid χ or χ_{sub}.

A χ process is a triple $\langle p, \sigma, E\rangle$, where p denotes a process term (statement), σ denotes a valuation, and E denotes an environment. A valuation is a partial function from variables to values. Syntactically, a valuation is denoted by a set of pairs $\{x_0 \mapsto c_0, \ldots, x_n \mapsto c_n\}$, where x_i denotes a variable and c_i its value. The environment E is a tuple (C, J, L, H, R), where C, J, L denote the set of continuous variables, the set of jumping continuous variables (i.e., variables that are allowed to jump in discrete transitions), and the set of algebraic variables, respectively, H is a set of channels, and R denotes a recursive process definition (partial function from recursion variables to process terms). The valuation σ and the environment E, together define the variables that exist in the χ process and the variable classes to which they belong. In χ, there is the predefined variable $\mathsf{time} \in \text{dom}(\sigma)$, that denotes the current (model) time.

A. Syntax

The subset χ_{sub} of the χ language for which the stepper functions are defined consists of processes $\langle p, \sigma, E\rangle$, where $p \in P_{\text{T}}$. The process terms from the set P_{T} are defined by the following grammar:

$$\begin{aligned} p_{\text{T}} \quad ::= \quad & W : r \gg l_{\text{a}} \quad | \quad u \quad | \quad [p_{\text{T}}] \quad | \quad p_{\text{T}};\ p_{\text{T}} \\ | \quad & b \rightarrow p_{\text{T}} \quad | \quad p_{\text{T}}\ []\ p_{\text{T}} \quad | \quad p_{\text{T}} \parallel p_{\text{T}} \\ | \quad & h\,!!\,\mathbf{e}_n \quad | \quad h\,??\,\mathbf{x}_n \quad | \quad X \end{aligned}$$

Here, W is a set of (non-dotted) variables such that $\mathsf{time} \notin W$, r is a predicate over variables (including the variable time), dotted continuous variables, and '$^-$' superscripted variables (including the dotted variables, e.g. $\dot{x}$). The action label l_{a} is taken from a given set A_{label}. Furthermore, u and b are both predicates over variables (including the variable time) and dotted continuous variables, h denotes a channel, $\mathbf{e}_n$ denotes the expressions $e_1, \ldots, e_n$, and $\mathbf{x}_n$ denotes the (non-dotted) variables $x_1, \ldots, x_n$ such that $\mathsf{time} \notin \{\mathbf{x}_n\}$. Finally, X is a recursion variable. The operators are listed in descending order of their binding strength as follows $\rightarrow$, ;, $\{\parallel, []\}$. The operators inside the braces have equal binding strength and parentheses may be used to group expressions.

An *action predicate* $W : r \gg l_{\text{a}}$ denotes instantaneous changes to the variables from set W, by means of an action labeled l_{a}, such that predicate r is satisfied.

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

Arbitrary delay behavior can be added to the behavior of process term p by means of the *any delay operator* $[p]$. The action behavior remains unchanged.

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

The *guarded process term* $b \rightarrow p$ can do whatever actions p can do under the condition that the guard b evaluates to true using the current extended valuation (see Section II-B). The guarded process term can delay according to p under the condition that for the intermediate extended valuations during the delay, the guard b holds. The guarded process term can perform arbitrary delays under the condition that for the intermediate valuations during the delay, possibly excluding the first and last valuation, the guard b does not hold.

The *alternative composition operator* $p\ []\ q$ models a nondeterministic choice between different actions of a process. With respect to time behavior, the participants in the alternative composition have to synchronize (i.e. a strong time-deterministic choice).

Parallelism can be specified by means of the *parallel composition process term* $p \parallel q$. Parallel processes interact by means of shared variables or by means of synchronous point-to-point communication/synchronization via a channel. The parallel composition $p \parallel q$ synchronizes the time behavior of p and q, interleaves the action behavior (including the instantaneous changes of variables) of p and q, and synchronizes matching send and receive actions. The synchronization of time behavior means that only the time behaviors that are allowed by both p and q are allowed by their parallel composition. The consistent equation semantics of χ enforces that actions by p (or q) are allowed only if the values of the variables before and after the actions are consistent with the other process term q (or p). This means, among others, that the delay predicates of q must hold before and after execution of an action by p.

By means of the *send process term* $h\,!!\,e_1, \ldots, e_n$, for $n \geq 1$, the values of expressions $e_1, \ldots, e_n$ (evaluated w.r.t. the extended valuation) are sent via channel h. For $n = 0$, this reduces to $h\,!!$ and nothing is sent via the channel.

By means of the *receive process term* $h\,??\,x_1, \ldots, x_n$, for

$n \geq 1$, values for $x_1, \ldots, x_n$ are received from channel h. We assume that all variables in $\mathbf{x}_n$ are different: $x_i = x_j \implies i = j$. For $n = 0$, this reduces to h ??, and nothing is received via the channel. Communication in χ is the sending of values by one parallel process via a channel to another parallel process, where the received values (if any) are stored in variables. For communication, the acts of sending and receiving (values) have to take place in different parallel processes at the same moment in time. In case no values are sent and received, we refer to synchronization instead of communication.

Process term X denotes a recursion variable (identifier) that is defined in the environment of the process. Among others, it is used to model repetition. Recursion variable X can do whatever the process term of its definition can do. The use of recursion is restricted to so called "guarded recursion" [19], [20], which means that the occurrence of each recursion variable in a process term must be preceded by some other process term in a sequential composition.

Besides process terms and operators introduced before, there is an additional, more user-friendly syntax available, the so-called syntactic extensions. These syntactic extensions are expressed in terms of the syntax introduced before. For details of these syntactic extensions, we refer to [1], [2]. The notation

$$
\begin{array}{l}
\langle\ \mathsf{disc}\ s_1, \ldots, s_k \\
,\ \mathsf{cont}\ x_1, \ldots, x_n \\
,\ i \\
,\ X_1 \mapsto p_1, \ldots, X_r \mapsto p_r \\
|\ p \\
\rangle
\end{array}
$$

is an abbreviation for the process term p in the environment $E = (\{x_1, \ldots, x_n\}, \emptyset, \emptyset, \emptyset, \{X_1 \mapsto p_1, \ldots, X_r \mapsto p_r\})$ and with a valuation satisfying the initialization predicate i. Here, $s_1, \ldots, s_k$ denote discrete variables, $x_1, \ldots, x_n$ denote non-jumping continuous variables, i denotes an initialization predicate that restricts the allowed values of the variables initially, $X_1 \mapsto p_1, \ldots, X_r \mapsto p_r$ denote the recursion definitions, and p is a process term. Often, if there are no discrete variables, these are omitted from the notation. This also applies to other parts of the notation.

B. Semantics

The valuation σ from a χ process $\langle p, \sigma, E\rangle$ captures the values of those variables that are relevant for determining the future behaviors of a process. The dotted continuous variables and the algebraic variables are not included in the domain of σ, because their values depend only on the process term p, possibly together with the values of the other variables.

The values of the dotted continuous and algebraic variables are included in the so-called 'extended valuation'. This extended valuation is required, among others, to ensure consistency of χ processes. Consistency is related to extended valuations in the following way: a χ process $\langle p, \sigma, E\rangle$ is consistent with extended valuation ξ, where ξ is the valuation σ extended with the (valuation for the) algebraic and dotted variables as defined in the environment E, if the 'active' delay predicates u in p hold when evaluated in extended valuation ξ.

The semantics of χ is defined by means of deduction rules in SOS style [17] that associate a hybrid transition system with a χ process as defined in [2]. Such a hybrid transition system has the following relations and predicates:

Action transitions: The intuition of an action transition $\langle p, \sigma, E\rangle \xrightarrow{\xi, a, \xi'} \langle p', \sigma', E'\rangle$ is that the process $\langle p, \sigma, E\rangle$ executes the discrete action a with extended valuations ξ and ξ' and thereby transforms into the process $\langle p', \sigma', E'\rangle$, where σ' and E' denote the accompanying valuation and environment of the process term p', respectively, after the discrete action a is executed. The extended valuations ξ and ξ' denote the extended valuations before and after the transition, respectively.

Termination transitions: The intuition of a termination transition $\langle p, \sigma, E\rangle \xrightarrow{\xi, a, \xi'} \langle \checkmark, \sigma', E'\rangle$ is that the process $\langle p, \sigma, E\rangle$ executes the discrete action a with extended valuations ξ and ξ' and thereby transforms into the terminated process $\langle \checkmark, \sigma', E'\rangle$.

Time transitions: The intuition of a time transition $\langle p, \sigma, E\rangle \overset{t, \rho}{\longmapsto} \langle p', \sigma', E'\rangle$ is that during the time transition, the extended valuation at each time point $s \in [0, t]$ is given by $\rho(s)$. At the end-point t, the resulting process is $\langle p', \sigma', E'\rangle$.

Consistency predicates: The intuition of a consistency predicate $\langle p, \sigma, E\rangle \overset{\xi}{\rightsquigarrow}$ is that the active delay predicates in the process $\langle p, \sigma, E\rangle$ are satisfied w.r.t. extended valuation ξ.

In χ, only consistent processes can do action or delay transitions, and the result of an action or delay transition is always a consistent process. This is why the semantics of hybrid χ is also called a 'consistent equation semantics'.

The relations and predicates mentioned above are defined through so-called deduction rules. A deduction rule is of the form $\frac{H}{r}$, where H is a number of hypotheses separated by commas and r is the result of the rule. The result of a deduction rule can be derived if all of its hypotheses are derived. In case the set of hypotheses is empty, the deduction rule is called an axiom (see Rule 2).

Next, the deduction rules for the action predicate, delay predicate and sequential composition operator are given to illustrate the definition of the semantics of hybrid χ. Deduction rules for the other language elements can be found in [1], [2].

Action predicate: The action predicate process term $W : r \gg l_{\mathrm{a}}$ denotes instantaneous changes to the variables from set W and the globally defined set of jumping continuous variables, by means of an action labeled l_{a}, such that predicate r over variables from the domains of the extended valuations before and after the transition is satisfied, see Rule 1.

The values of the variables from $\mathrm{dom}(\sigma)$ in ξ are given by σ. The dotted variables $\dot{C}$ and the algebraic variables L in ξ can in principle take any value ($\xi = \sigma \cup \xi^{\dot{C}L}$, where $\xi^{\dot{C}L}$ is an arbitrary valuation (function) with domain $\dot{C} \cup L$

and $\cup$ combines two functions with disjoint domains into one function) as long as the action predicate r is satisfied ($\xi^- \cup \xi' \models r$). Variables occurring with a '$^-$' superscript in r are evaluated in ξ^-, which denotes the extended valuation with the values of variables before the discrete change: $\mathrm{dom}(\xi^-) = \{x^- \mid x \in \mathrm{dom}(\xi)\}$ and $\xi^-(x^-) = \xi(x)$. For extended valuation ξ', the values of the discrete and the non-jumping continuous variables ($\mathrm{dom}(\sigma) \setminus (J \cup W)$) are given by σ. The jumping continuous variables J, the variables from set W, the dotted variables $\dot{C}$ and the algebraic variables L are allowed to change such that the predicate r is satisfied. Function Ξ returns a set of extended valuations, given a valuation, a set of continuous variables, a set of jumping continuous variables, and a set of algebraic variables. Formally, function Ξ is defined as:

$$\Xi(\sigma, C, J, L) \quad = \quad \{\, \xi \mid \mathrm{dom}(\xi) = \mathrm{dom}(\sigma) \cup \dot{C} \cup L, \; \forall_{x \in \mathrm{dom}(\sigma) \setminus J}\, \xi(x) = \sigma(x) \,\}.$$

Since there are no time transition rules defined for action predicates, this means that action predicates cannot perform any time transitions.

Rule 2 states that action predicates are always consistent with any extended valuation $\sigma \cup \xi^{\dot{C}L}$ with respect to σ in any environment E. In the deduction rules given below, it is assumed that $E = (C, J, L, H, R)$.

$$\frac{\xi = \sigma \cup \xi^{\dot{C}L},\ \xi' \in \Xi(\sigma, C, J \cup W, L),\ \xi^- \cup \xi' \models r}{\langle W : r \gg l_{\mathrm{a}}, \sigma, E\rangle \xrightarrow{\ \xi,\ l_{\mathrm{a}},\ \xi'\ } \langle \checkmark, \xi' \upharpoonright \mathrm{dom}(\sigma), E\rangle}\ 1$$

$$\frac{}{\langle W : r \gg l_{\mathrm{a}}, \sigma, E\rangle \overset{\sigma \cup \xi^{\dot{C}L}}{\rightsquigarrow}}\ 2$$

Delay predicate: Delay predicate u is a predicate over variables and dotted continuous variables. In the deduction rules given below it is assumed that $E = (C, J, L, H, R)$.

$$\frac{\rho \in \Omega_{FG}(\sigma, C, L, u, t)}{\langle u, \sigma, E\rangle \overset{t,\rho}{\longmapsto} \langle u, \rho(t) \upharpoonright \mathrm{dom}(\sigma), E\rangle}\ 3 \qquad \frac{\sigma \cup \xi^{\dot{C}L} \models u}{\langle u, \sigma, E\rangle \overset{\sigma \cup \xi^{\dot{C}L}}{\rightsquigarrow}}\ 4$$

Function Ω_{FG} returns a set of trajectories, given a valuation representing the current values of the discrete and continuous variables, the set of continuous variables, the set of algebraic variables, a delay predicate and a time point that denotes the duration of the trajectory.

A trajectory ρ is a function from the time interval $[0, t]$, where $t \geq 0$, to an extended valuation, where the domain of each valuation consists of all variables and dotted continuous variables. The trajectory ρ satisfies the predicate u for all time points of its domain ($\forall_{s \in [0,t]}\, \rho(s) \models u$). The trajectory of each discrete variable $x \in \mathrm{dom}(\sigma) \setminus (\{\mathsf{time}\} \cup C)$ is restricted to a constant function. The initial value (starting-point) of the trajectory of each discrete and continuous variable equals the value of that variable in σ ($\forall_{x \in \mathrm{dom}(\sigma)}\, (\rho \downarrow x)(0) = \sigma(x)$). Here $\rho \downarrow x$ denotes a function from a time interval to a value and it is defined by $(\rho \downarrow x)(t) = \rho(t)(x)$.

The trajectories of the algebraic variables ($\rho \downarrow x$ for $x \in L$) are required to be functions of type F. This set of functions is a parameter of the solution concept of χ. Having the set F as a parameter of the solution concept allows us to restrict F to, for instance, the set of piecewise constant functions, if this would be required for certain properties to hold.

The trajectories of the dotted variables are required to be integrable. The relation between the trajectory of a continuous variable x and the trajectory of its 'derivative' $\dot{x}$ is given by the Caratheodory solution concept [21]: $(\rho \downarrow x)(s) = (\rho \downarrow x)(0) + \int_0^s (\rho \downarrow \dot{x})(s')ds'$. Note that this integral relation can hold only for those continuous variables for which $\rho \downarrow x$ is an absolutely continuous function. Thus the solution function Ω_{FG} restricts the trajectory $\rho \downarrow x$ of every continuous variable x to an absolutely continuous function, but it does allow a non-smooth trajectory for a continuous variable in the case that the trajectory of its 'derivative' $\rho \downarrow \dot{x}$ is non-smooth or even discontinuous, as in, for example, $\langle\ \mathsf{cont}\ y,\ y = 0 \mid \dot{y} = \mathrm{step}(\mathsf{time} - 1)\ \rangle$, where $\mathrm{step}(x)$ equals 0 for $x \leq 0$ and 1 for $x > 0$. For a complete and formal definition of the solution function Ω_{FG} see [1], [2].

Sequential composition: The sequential composition of process terms p and q behaves as process term p until p terminates, and then continues to behave as process term q. When p terminates, its resulting extended valuation ξ' must be consistent with q (see Rule 5).

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{\ \xi,a,\xi'\ } \langle \checkmark, \sigma', E\rangle,\ \langle q, \sigma', E\rangle \overset{\xi'}{\rightsquigarrow}}{\langle p;\ q, \sigma, E\rangle \xrightarrow{\ \xi,a,\xi'\ } \langle q, \sigma', E\rangle}\ 5$$

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{\ \xi,a,\xi'\ } \langle p', \sigma', E\rangle}{\langle p;\ q, \sigma, E\rangle \xrightarrow{\ \xi,a,\xi'\ } \langle p';\ q, \sigma', E\rangle}\ 6$$

$$\frac{\langle p, \sigma, E\rangle \overset{t,\rho}{\longmapsto} \langle p', \sigma', E\rangle}{\langle p;\ q, \sigma, E\rangle \overset{t,\rho}{\longmapsto} \langle p';\ q, \sigma', E\rangle}\ 7 \qquad \frac{\langle p, \sigma, E\rangle \overset{\xi}{\rightsquigarrow}}{\langle p;\ q, \sigma, E\rangle \overset{\xi}{\rightsquigarrow}}\ 8$$

III. FORMAL DEFINITION OF THE STEPPER

The stepper computes the set of possible transitions given a χ process. The stepper consists of four main functions: function $\mathcal{S}_{\mathrm{a}}$ which returns the set of action steps given a χ process, function $\mathcal{S}_{\mathrm{d}}$ which returns the set of time steps given a χ process, function Tr which returns the set of transitions given a step, and function Tr$'$ which returns the reduced set of transitions. In the semantics of χ, transitions are derived locally. For instance, in order to determine the transitions of the process $\langle \mathsf{cont}\ x, y, x = y = 0 \mid \dot{x} = 1 \parallel \dot{y} = 2 \rangle$, the transitions of the process terms $\dot{x} = 1$ and $\dot{y} = 2$ are determined in isolation from each other. For both cases, this results in an infinite number of possible transitions: Solving $\dot{x} = 1$ in isolation returns an unique solution for x, and an infinite number of solutions for y, and on the other hand, solving $\dot{y} = 2$ in isolation returns an unique solution for y, and an infinite number of solutions for x. The solutions of the process term $\dot{x} = 1 \parallel \dot{y} = 2$ are defined as the intersection of both sets of solutions. For an implementation, it is better to solve the equations $\dot{x} = 1$ and $\dot{y} = 2$ simultaneously, which in this case result in unique solutions for both x and y. In the stepper, this is achieved by means of action and time steps. Action steps and time steps can be seen as symbolic transitions. They consist of

all information that is needed to determine the transitions that the process from which they are derived can perform. An action step represents zero or more action transitions and a time step represents zero or more time transitions. An action step $(c, W, r, l_a, C^b, C^a, p')$ consists of the condition (guards) c that should hold, the set of variables W that may change, the predicate r describing the discrete updates, the performed action label l_a, the consistency requirements before the action C^b, the consistency requirements after the action C^a, and the resulting process term p'. A time step $(c^{[0]}, c^{(0,t)}, c^{[t]}, c^{[0,t]}, c, p')$ consists of the predicate $c^{[0]}$ that should hold at the start point of the time transitions, the predicate $c^{(0,t)}$ that should hold at all time points between the start- and endpoint of the time transitions, the predicate $c^{[t]}$ that should hold at the endpoint of the time transitions, the predicate $c^{[0,t]}$ that should hold at all time points (including the start- and endpoint) of the time transitions, the predicate c that should hold at least at one time point of the time transition, and the resulting process term p'.

The type of information that is stored in the action and time steps is derived from the SOS by looking very carefully at the premises of the deduction rules. For example, the predicate C^a as a part of the action steps results from the premise about the consistency of the process term q in Rule 5.

A. Derivation of Functions $\mathcal{S}_a$ and $\mathcal{S}_d$

Given a χ process, functions $\mathcal{S}_a$ and $\mathcal{S}_d$ return the set of action steps and the set of time steps, respectively. For the $\mathcal{S}_a$ and $\mathcal{S}_d$ functions, no computations other than syntactic rewriting have to be performed.

Function $\mathcal{S}_a$ returns a set of action steps of a χ_{sub} process. For the action predicate, the delay predicate and the sequential composition operator, function $\mathcal{S}_a$ is defined as follows:

$$
\begin{aligned}
&\mathcal{S}_a(W : r \gg l_a, E) = \{(\text{true}, W, r, l_a, \text{true}, \text{true}, \checkmark)\}\\
&\mathcal{S}_a(u, E) = \emptyset\\
&\mathcal{S}_a(p;\, q, E) \;= \{(c, W, r, l_a, C^b, C^a \wedge \mathcal{C}_c(q, E), q)\\
&\qquad\qquad |(c, W, r, l_a, C^b, C^a, \checkmark) \in \mathcal{S}_a(p, E)\}\\
&\qquad\qquad \cup\; \{(c, W, r, l_a, C^b, C^a, p';\, q)\\
&\qquad\qquad\; |(c, W, r, l_a, C^b, C^a, p') \in \mathcal{S}_a(p, E)\\
&\qquad\qquad\; , p' \neq \checkmark\}
\end{aligned}
$$

Function $\mathcal{C}_c$ returns the consistency predicate which has to be satisfied in order for p to be consistent. For the action predicate, delay predicate and sequential composition operator, function $\mathcal{C}_c$ is defined as follows:

$$
\begin{aligned}
&\mathcal{C}_c(W : r \gg l_a, E) = \text{true}\\
&\mathcal{C}_c(u, E) = u\\
&\mathcal{C}_c(p;\, q, E) = \mathcal{C}_c(p, E)
\end{aligned}
$$

Observe that these definitions are closely related to the deduction rules presented before. For example, for a delay predicate there are no action steps since there is no deduction rule that allows the derivation of an action transition for the delay predicate. As another example, the first contributing set in the definition of the action steps of a sequential composition is derived from Rule 5. The part $(c, W, r, l_a, C^b, C^a, \checkmark) \in \mathcal{S}_a(p, E)$ refers to the first premise and the conjunct $\mathcal{C}_c(q, E)$ refers to the second premise.

Function $\mathcal{S}_d$ returns a set of time steps of a χ_{sub} process. For the action predicate, delay predicate and sequential composition operator, function $\mathcal{S}_d$ is defined as follows:

$$
\begin{aligned}
&\mathcal{S}_d(W : r \gg l_a, E) = \emptyset\\
&\mathcal{S}_d(u, E) = \{(u, u, u, u, \text{true}, u)\}\\
&\mathcal{S}_d(p;\, q, E) \;= \{(c^{[0]}, c^{(0,t)}, c^{[t]}, c^{[0,t]}, c, p';\, q)\\
&\qquad\qquad |(c^{[0]}, c^{(0,t)}, c^{[t]}, c^{[0,t]}, c, p') \in \mathcal{S}_d(p, E)\}
\end{aligned}
$$

Also for this function the definitions are derived from the deduction rules for time transitions, although this is less obvious here. Observe that for the given definitions of these functions the environment E is not used. It is used in the definitions for $\|$ (see [2]).

B. Transition functions

Function Tr returns a set of transitions given a χ_{sub} process. It is defined as $\text{Tr} = \text{Tr}_a \cup \text{Tr}_d$. Function Tr_a is defined as follows.

$$
\begin{aligned}
&\text{Tr}_a(p, \sigma, (C, J, L, H, R)) =\\
&\quad \{\langle p, \sigma, (C, J, L, H, R)\rangle \xrightarrow{\xi, a, \xi'} \langle p', \xi'_\sigma, (C, J, L, H, R)\rangle\\
&\quad |(c_p, W_p, r_p, l_{ap}, C^b_p, C^a_p, p') \in \mathcal{S}_a(\langle p, (C, J, L, H, R)\rangle)\\
&\quad , \xi = \sigma \cup \xi^{\dot{C}L}, \xi \models c_p, \xi' \in \Xi(\sigma, C, J \cup W_p, L)\\
&\quad , \xi^- \cup \xi' \models r_p, \xi \models C^b_p, \xi' \models C^a_p\},
\end{aligned}
$$

where a denotes $\mathcal{M}(\xi, l_{ap}, \xi')$, and function $\mathcal{M}$ is defined as follows, where $[\mathbf{e}_n]$ denotes a list of expressions, $[\mathbf{x}_n]$ denotes a list of variables, and h denotes a channel:

$$
\mathcal{M}(\xi, l_a, \xi') = \begin{cases} \text{isa}(h, [\xi(\mathbf{e}_n)]) & \text{if } l_a \equiv \text{isa}(h, [\mathbf{e}_n])\\ \text{ira}(h, [\xi'(\mathbf{x}_n)], \{\mathbf{x}_n\}) & \text{if } l_a \equiv \text{ira}(h, [\mathbf{x}_n])\\ \text{ca}(h, [\xi(\mathbf{e}_n)]) & \text{if } l_a \equiv \text{ca}(h, [\mathbf{e}_n])\\ l_a & \text{otherwise.} \end{cases}
$$

The information that has been collected in the predicates of the action steps is now interpreted, i.e., predicates are solved. For example $\xi^- \cup \xi' \models r_p$ now means that the extended valuations ξ and ξ' together have to satisfy the collected predicates that describe the discrete changes to the discrete, algebraic and jumping continuous variables.

Function Tr_d is defined as follows:

$$
\begin{aligned}
&\text{Tr}_d(p, \sigma, (C, J, L, H, R)) =\\
&\quad \{\langle p, \sigma, (C, J, L, H, R)\rangle \overset{t, \rho}{\longmapsto} \langle p', \rho_\sigma, (C, J, L, H, R)\rangle\\
&\quad |(c^{[0]}_p, c^{(0,t)}_p, c^{[t]}_p, c^{[0,t]}_p, c_p, p') \in \mathcal{S}_d(\langle p, (C, J, L, H, R)\rangle)\\
&\quad , \rho \in \Omega_{FG}(\sigma, C, L, c^{[0,t]}_p, t), \rho(0) \models c^{[0]}_p,\\
&\quad \forall_{s \in (0,t)} \rho(s) \models c^{(0,t)}_p, \rho(t) \models c^{[t]}_p, \exists_{s \in [0,t]} \rho(s) \models c_p\}
\end{aligned}
$$

Similarly, for this function trajectories ρ are defined that satisfy the predicates that have been collected using the solution function Ω_{FG} of χ.

In [2], it is proven that all action and termination transitions in the range of the transition function can also be

derived from the semantics of χ. As conjectures we have that all time transitions in the range of the transition function can also be derived from the semantics of χ, and that all transitions that can be derived from the semantics of χ are also in the range of the transition function.

In general, the set of transitions of a χ specification is infinite. For instance, action predicate $\{x\} : \text{true} \gg \tau$ has an infinite number of solutions for variable x (assuming that the type of x contains an infinite number of elements). The same holds for delay predicates that can have an infinite number of solutions. The number of time transitions a χ process can have may also be infinite: if a process can delay for t time units, then, for every $0 \leq t' \leq t$, it can also delay for t' time units. Since the time domain of χ is the set of real numbers, there frequently are infinitely many time transitions.

As an attempt to get rid of this infinity as often as possible, we can define a function Tr' which returns a reduced set of transitions (see [2] for its definition). Instead of returning all time transitions of a time step, for each trajectory only the time transition with longest duration is returned. Although this reduced set of transitions can still be infinite, in practice, this is rarely the case.

IV. SIMULATOR

Simulation is a powerful method to analyze the dynamic behavior of a model. In this section, a simulator is defined which is mainly based on the stepper functions from Section III. Using pseudo-code, the simulator is defined as follows.

```
Simulate(⟨p, σ, E⟩) =
    while p ≠ ✓ do
        transitions := Tr'(⟨p, σ, E⟩)
        if transitions = ∅ then
            return deadlock
        else
            transition := pick(transitions)
            Visualize(transition)
            ⟨p, σ, E⟩ := GetState(transition)
        endif
    endwhile
    return simulation_ended
```

First the function Tr', that is defined in the previous section, returns the set of possible transitions *transitions*. The process has deadlock behavior (**return** *deadlock*) if the set of transitions is empty. Otherwise, a transition (*transition*) is selected (non-deterministically) from the set of transitions. The transition is then visualized. This may be done while the simulation is running, as indicated in the algorithm shown above, or at the end of the simulation. The resulting state of the transition is obtained by means of function GetState. If the process is terminated ($p = \checkmark$), the simulation is successfully terminated (**return** *simulation_ended*).

Note that an implementation of the stepper functions may impose additional restrictions on the χ_{sub} syntax. For instance, for an implementation of function Tr' a (symbolic) solver is needed to compute the solutions of action predicates, the solution of delay predicates, and the maximum duration of a time transition. Depending on the solver that is used, additional restrictions may be required.

The stepper functions are defined in such a way that it is easy to define different implementations. For instance, in [22], we defined a so-called DE$^+$ simulator. This simulator interacts with Matlab Simulink [10] using an S-function as follows. The DE$^+$ simulator performs action transitions until the stepper returns a time step. This time step is returned to Simulink. During the time transition, Simulink solves the delay predicates as specified in the time step. At the end of the time transition, the DE$^+$ simulator is called again.

The DE$^+$ simulator consists of a function Simulate, which is defined as follows:

```
Simulate(⟨p, σ, E⟩) =
    while p ≠ ✓ do
        steps := S_a(⟨p, σ, E⟩) ∪ S_d(⟨p, σ, E⟩)
        step := pick(steps)
        if step = ActionStep then
            ⟨p, σ, E⟩ := DoActionStep(step)
        else
            return step
        endif
    endwhile
    return stop_simulation
```

Given a χ_{sub} process $\langle p, \sigma, E\rangle$, functions $\mathcal{S}_{\text{a}}$ and $\mathcal{S}_{\text{d}}$ return the set of possible steps (*steps*) that the process can perform. From this set of steps, one step (*step*) is selected (non-deterministically) (*step* := pick(*steps*)). If the selected step is an action step, then this step is performed (DoActionStep(*step*)) (or more precisely, one of the possible transitions of this step is performed). If the selected step is a time step, then this step is returned (**return** *step*). If the process is terminated ($p = \checkmark$), a request is sent to the simulation coordinator to stop the simulation (*stop_simulation*).

V. THIRD PARTY TOOLS

The χ tool set is designed in such a way that it can be integrated with third party tools. The main reason for this is to reuse existing applications and libraries. For the implementation of the stepper and the symbolic simulator, the Python [18] programming language has been used.

Currently, the χ tools are integrated with four different tools: Maple, Gnuplot, Graphviz, and PHAVer.

For the computation of solutions of action predicates and delay predicates, and the maximum duration of time transitions, the symbolic solving capabilities of the mathematical package Maple [23] are used.

For the visualization of simulation results, in particular the trajectories of time transitions, the portable command-line driven interactive data and function plotting utility Gnuplot [24] is used.

For visualization of the hybrid transition systems obtained by means of the simulator the Graphviz tools are used. Graphviz [25] is an open toolkit for graph visualization. It is

developed at AT&T Labs-Research. The Graphviz tools use a common language to specify attributed graphs. This language is called *Libgraph*, but is probably better known as the *dot* format, after its best-known application. Graphviz provides tools for graph filtering and graph rendering. The filtering tools can be batch-oriented as well as interactive. For our application, visualization of hybrid automata, we only need a small part of the functionality offered by Graphviz. For instance, in the hybrid transition system there are only three types of states/locations (initial, termination and normal) and two different types of edges (action transitions and time transitions) in our graphs.

PHAVer (Polyhedral Hybrid Automaton Verifyer) [26] is a tool for analyzing linear hybrid automata. Currently, PHAVer is used as a verification engine for the hybrid automata obtained by the formal translation from χ specifications to hybrid automata (see [2] for more details about this translation).

VI. CASE STUDY USING THE SIMULATOR

Figure 1 shows a bottle filling line taken from [27]. It consists of a storage tank that is continuously filled with a flow Q_{in}, a conveyor belt that supplies empty bottles, and a valve that is opened when an empty bottle is below the filling nozzle, and is closed when the bottle is full. When a bottle has been filled, the conveyor starts moving to put the next bottle under the filling nozzle, which takes one unit of time. When the storage tank is not empty, the bottle filling flow Q equals Q_{set}. When the storage tank is empty, the bottle filling flow equals the flow Q_{in}. We assume $Q_{\text{in}} < Q_{\text{set}}$.

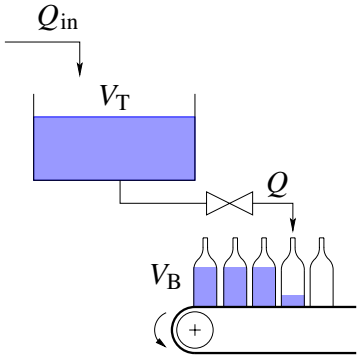

Fig. 1. Filling Line.

The model is defined below. The constants V_{T0}, V_{Tmax}, and V_{Bmax} define the initial volume of the storage tank, the maximum volume of the storage tank, and the filling volume of the bottles, respectively. The constants Q_{in} and Q_{set} define the value of the flow that is used to fill the storage tank, and the maximum value of the bottle filling flow Q.

$$
\begin{array}{lll}
\langle\ \mathsf{disc}\ x,\ Q & & \\
,\ \mathsf{cont}\ V_{\text{T}},\ V_{\text{B}} & & \\
,\ x = Q = V_{\text{B}} = 0,\ V_{\text{T}} = V_{\text{T0}} & & \\
,\ \mathit{closed} & \mapsto & \dot{V}_{\text{T}} = Q_{\text{in}},\ V_{\text{T}} \le V_{\text{Tmax}} \\
 & & []\,[\mathit{open}\,??\,Q];\ \mathit{opened} \\
,\ \mathit{opened} & \mapsto & ((\dot{V}_{\text{T}} = Q_{\text{in}} - Q,\ 0 \le V_{\text{T}} \le V_{\text{Tmax}}) \\
 & & []\,[\mathit{close}\,??\,Q];\ \mathit{closed} \\
 & & []\,[\{Q\} : Q = Q_{\text{in}} \gg c];\ \mathit{empty} \\
 & &) \\
,\ \mathit{empty} & \mapsto & (V_{\text{T}} = 0 \\
 & & []\,[\mathit{close}\,??\,Q];\ \mathit{closed} \\
 & &) \\
,\ \mathit{moving} & \mapsto & (\{V_{\text{B}}, x\} : V_{\text{B}} = 0,\ x = \mathsf{time} + 1 \gg a \\
 & & ;\ \mathsf{time} \ge x \rightarrow \mathit{open}\,!!\,Q_{\text{set}};\ \mathit{filling} \\
 & &) \\
,\ \mathit{filling} & \mapsto & (V_{\text{B}} \ge V_{\text{Bmax}} \rightarrow \mathit{close}\,!!\,0;\ \mathit{moving}) \\
\mid \mathit{closed} \parallel \mathit{moving} \parallel \dot{V}_{\text{B}} = Q & & \\
\rangle & &
\end{array}
$$

The storage tank is modeled by means of recursion variables / modes: *closed*, *opened*, and *empty* that correspond to the valve being closed, the valve being opened, and the valve being opened while the storage tank is empty.

In the mode *opened*, the storage tank is usually not empty. When the storage tank is empty in mode *opened*, the delayable action predicate $[\{Q\} : Q = Q_{\text{in}} \gg c]$ may be executed causing the next mode to be *empty*. Due to the consistent equation semantics, the action predicate can be executed only if the delay predicate $V_{\text{T}} = 0$ in the next mode *empty* holds. Therefore, the transition to mode *empty* can be taken only when the storage tank is empty. The initial mode is *closed*.

The conveyor is modeled by means of recursion variables / modes *moving* and *filling*. In mode *moving*, the conveyor supplies an empty bottle in 1 second ($\{V_{\text{B}}, x\} : V_{\text{B}} = 0, x = \mathsf{time} + 1 \gg a$; $\mathsf{time} \ge x \rightarrow \mathit{open}\,!!\,Q_{\text{set}}$. Then it synchronizes with the storage tank process by means of the send statement $\mathit{open}\,!!\,Q_{\text{set}}$, and it proceeds in mode *filling*. When the bottle is filled in mode filling ($V_{\text{B}} \ge V_{\text{Bmax}}$), the process synchronizes with the storage tank to close the valve and returns to mode moving. The initial mode is *moving*.

Figure 2 shows the first 12 seconds of a simulation run of the model using the following values for the constants: $V_{\text{T0}} = 5$ liter, $V_{\text{Bmax}} = 10$ liter, $V_{\text{Tmax}} = 20$ liter, $Q_{\text{in}} = 1.5\ \frac{\text{liter}}{\text{sec}}$, and $Q_{\text{set}} = 3\ \frac{\text{liter}}{\text{sec}}$.

VII. CONCLUSIONS

In this paper, we showed how one can derive implementations of simulators for hybrid χ models. In this approach, first symbolic action and time transitions are computed. We explained how the corresponding definitions are obtained from the deduction rules defining the SOS of the constructions of the hybrid χ language.

Based on these stepper functions, one can for example define a simulator which uses a symbolic solver for solving the predicates that are collected in the stepper functions.

A simulator that has been built based on the principles described in this paper has been used for simulating a hybrid χ model of a bottle filling line.

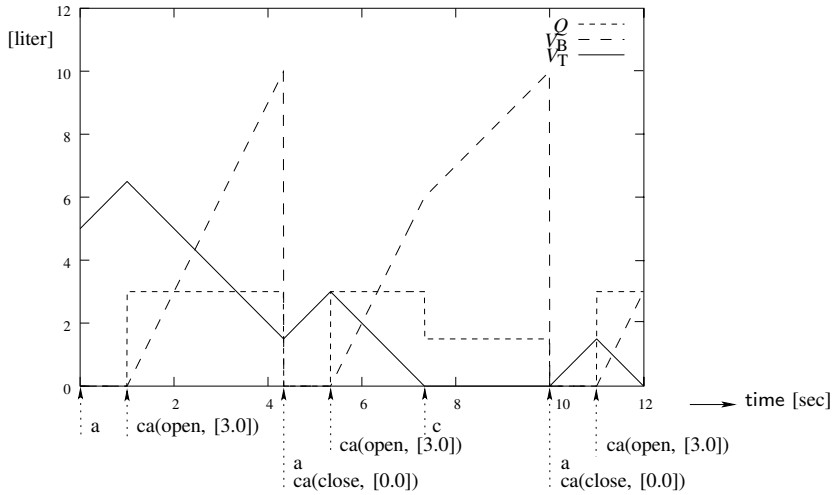

Fig. 2. Simulation results of model *FillingLine*.

VIII. ACKNOWLEDGMENTS

The authors gratefully acknowledge Jos Baeten, Albert Hofkamp, and Rolf Theunissen for their support and useful comments on preliminary versions of this paper.

REFERENCES

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

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

[3] G. Naumoski and W. Alberts, "A discrete-event simulator for systems engineering," Ph.D. dissertation, Eindhoven University of Technology, 1998.

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

[5] V. Bos and J. J. T. Kleijn, "Formal specification and analysis of industrial systems," Ph.D. dissertation, Eindhoven University of Technology, 2002.

[6] V. Bos and J. J. T. Kleijn, "Automatic verification of a manufacturing system," *Robotics and Computer Integrated Manufacturing*, vol. 17, no. 3, pp. 185–198, 2000.

[7] N. W. A. Arends, "A systems engineering specification formalism," Ph.D. dissertation, Eindhoven University of Technology, 1996.

[8] G. Fábián, "A language and simulator for hybrid systems," Ph.D. dissertation, Eindhoven University of Technology, 1999.

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

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

[11] Modelica Association, *Modelica - A Unifi ed Object-Oriented Language for Physical Systems Modeling*, http://www.modelica.org, 2002.

[12] 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," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3–34, 1995.

[13] P. J. L. Cuijpers and M. A. Reniers, "Hybrid process algebra," *Journal of Logic and Algebraic Programming*, vol. 62, no. 2, pp. 191–245, 2005.

[14] J. A. Bergstra and C. A. Middelburg, "Process algebra for hybrid systems," Eindhoven University of Technology, Department of Computer Science, The Netherlands, Tech. Rep. CS-Report 03-06, 2003.

[15] A. D. Febbraro, A. Giua, and G. Menga, Eds., *Special Issue on Hybrid Petri Nets*, ser. Journal of Discrete Event Dynamic Systems, vol. 11, no. 1 and 2, 2001.

[16] R. David and H. Alla, "On hybrid Petri nets," *Discrete Event Dynamic Systems: Theory & Applications*, vol. 11, no. 1-2, pp. 9–40, 2001.

[17] G. D. Plotkin, "A structural approach to operational semantics," Computer Science Department, Aarhus University, Tech. Rep. DIAMI FN-19, 1981.

[18] Python website, 2005, http://www.python.org.

[19] R. Milner, *A Calculus of Communicating Systems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 1980, vol. 92.

[20] G. J. Milne, "Abstraction and nondeterminism in concurrent systems," in *3rd Int. Conference on Distr. Systems*, 1982, pp. 358–364.

[21] A. F. Filippov, *Differential Equations with Discontinuous Right Hand Sides*. Dordrecht: Kluwer Academic Publishers, 1988.

[22] D. A. van Beek, R. R. H. Schiffelers, M. Kvasnica, and M. A. P. Remelhe, "Specification of the simulation interface," HYCON, Tech. Rep. D 3.4.1, 2005.

[23] MapleSoft, "Maple," http://www.maplesoft.com.

[24] Gnuplot website, "Gnuplot," http://www.gnuplot.info.

[25] R. G. Emden and S. C. North, "An open graph visualization system and its applications to software engineering," *Software–Pratice and Experience*, vol. 30, no. 11, pp. 1203–1233, 2000.

[26] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Hybrid Systems: Computation and Control, 8th International Workshop*, ser. Lecture Notes in Computer Science, M. Morari and L. Thiele, Eds. Springer-Verlag, 2005, vol. 3414, pp. 258–273.

[27] J. F. Groote and J. J. van Wamel, "Analysis of three hybrid systems in timed μCRL," *Science of Computer Programming*, vol. 39, pp. 215–247, 2001.