A Modular Visual Model for Hybrid Systems

Radu Grosu, Thomas Stauner* and Manfred Broy

Institut für Informatik, TU München, D-80290 München
Email: {grosu,stauner,broy}@informatik.tu-muenchen.de

Abstract. Visual description techniques are particularly important for the design of hybrid systems because specifications of such systems must usually be discussed between engineers from a number of different disciplines. Modularity is vital for hybrid systems not only because it allows to handle large systems, but also because hybrid systems are naturally decomposed into the system itself and its environment.
Based on two different interpretations for hierarchic graphs and on a clear hybrid computation model, we develop *HyCharts*, two modular visual formalisms for the specification of the architecture and behavior of hybrid systems. The operators on hierarchic graphs enable us to give a surprisingly simple denotational semantics for many concepts known from statechart-like formalisms. Due to a very general composition operator, HyCharts can easily be composed with description techniques from other engineering disciplines. Such heterogeneous system specifications seem to be particularly appropriate for hybrid systems because of their interdisciplinary character.

1 Introduction

Hybrid systems have been a very active area of research over the past few years and a number of specification techniques have been developed for such systems. While they are all well suited for closed systems, the search for hybrid description techniques for open systems is relatively new.

For open systems – as well as for any large system – modularity is essential. It is not only a means for decomposing a specification into manageable small parts, but also a prerequisite for reasoning about the parts individually, without having to regard the interior of other parts. Thus, it greatly facilitates the design process and can help to push the limits of verification tools, like model-checkers, further.

With a collection of operators on hierarchic graphs as tool-set, we follow the ideas in [6] and define a simple and powerful computation model for hybrid systems. Based on this model HyCharts, namely HySCharts and HyACharts, are introduced as two different interpretations of hierarchic graphs. HySCharts are a visual representation of hybrid, hierarchic state transition diagrams. HyACharts are a visual representation of hybrid data-flow graphs (or architecture graphs) and allow to compose hybrid components in a modular way. The behavior of

* The second author was supported with funds of the DFG, within the priority program *Design and design methodology of embedded systems* (reference number Br 887/9-1).

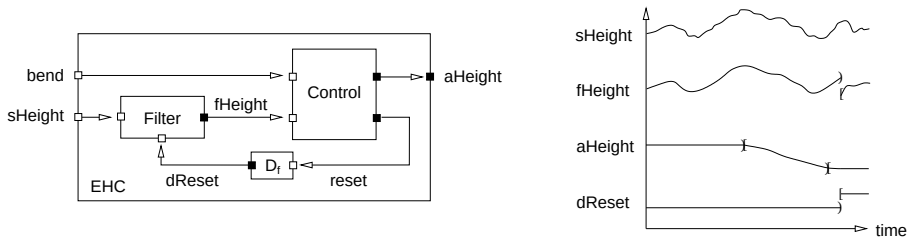

Fig. 1. The EHC: Architecture and a typical evolution.

these components can be described by using HySCharts or by any technique from system theory that can be given a semantics in terms of dense input/output relations. This includes differential equations. Dense input/output relations are a relational extension of hybrid FOCUS [10, 4].

Example 1 (An electronic height control system, EHC) The following example illustrates the kind of systems we want to regard. It will be used throughout the paper to demonstrate the use of HyCharts.

The purpose of the electronic height control system (EHC), which was originally proposed by BMW, is to control the chassis level of an automobile by a pneumatic suspension. The abstract model of this system which regards only one wheel was first presented in [12]. It basically works as follows: Whenever the chassis level is below a certain lower bound, a *compressor* is used to increase it. If the level is too high, air is blown off by opening an *escape valve*. The chassis level *sHeight* is measured by *sensors* and *filtered* to eliminate noise. The filtered value *fHeight* is read periodically by the *controller*. which operates the compressor and the escape valve and resets the filter when necessary. A further sensor *bend* tells the controller whether the car is going through a curve.

Here, we concentrate on the software part of the system. The environment is omitted. The basic components of the *system* are therefore the filter and the controller. The escape valve and the compressor are modeled within the controller. The diagrams in Figure 1 depict on the left the architecture of the EHC and on the right a typical evolution of the system over time. The architecture of the EHC is given by a HyAChart. Each component of this chart can be defined again by a HyAChart or by a HySChart or some other compatible formalism. The components only interact via clearly defined interfaces, or channels, in order to get modularity. The behavior of a component is characterized, as intuitively shown in Figure 1, right, by periods where the values on the channels change smoothly and by time instances at which there are discontinuities. In our approach the smooth periods result from the analog parts of the components. The discontinuities are caused by their combinational (or discrete) parts.

We specify the behavior of both the combinational and the analog part of a component by a HySChart, i.e., by a hybrid, hierarchic state transition diagram, with nodes marked by activities and transitions marked by actions. The transitions define the discontinuities, i.e., the instantaneous actions performed by the combinational part. The activities define the smooth periods, i.e., the time

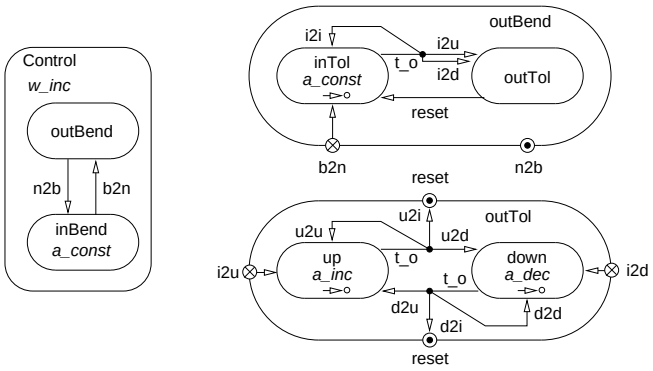

Fig. 2. The EHC's *Control* component.

consuming behavior of the analog part while the combinational part is idle. As an example, Figure 2 shows the HySChart for the EHC's *Control* component. It consists of three hierarchic levels. Figure 2, left, depicts the highest level. Figure 2, top right, refines the state *outBend* and Figure 2, bottom right, further refines the state *outTol*. The states, transitions and activities (written in italics in the figure) are explained in Section 5. □

In contrast to the well-known technique of hybrid automata [1], HyCharts are fully modular and suitable for open systems. The hybrid modules from Alur and Henzinger [2] are modular, but their utility suffers from the fact that it is not obvious how to model feedback loops. For theoretical reasons, loops pose a problem in our approach, too. Nevertheless we explicitly allow feedback loops, as long as they introduce a delay. Demanding a delay is not unrealistic, as signals cannot be transmitted at infinite speed. Another modular model, hybrid I/O automata, is presented in [9]. While this model is promising from the theoretical point of view, we think it has some deficits in practice. Namely, there is no graphical representation for hybrid I/O automata yet, there is no hierarchy concept for them and finally, there is no visual formalism for the specification of the architecture of a composed system. The same applies for the hybrid modules mentioned above. From the systems engineering point of view our approach is therefore more convenient. In contrast to the hybrid statecharts introduced in [8] HyCharts not only permit hierarchic states, but also hierarchic activities. HyCharts look largely similar to the description techniques used in the software engineering method for real-time object-oriented systems ROOM [11] and may therefore be seen as a hybrid extension of them.

The rest of the paper is organized as follows. In Section 2 we present an abstract interpretation of hierarchic graphs. This interpretation provides the infrastructure for defining an unusually simple denotational semantics for the key concepts of statecharts [7] offered in HyCharts, like hierarchy and preemption. It is also the foundation for the denotational semantics of our hybrid computation model, which is introduced in Section 3. Following the ideas developed in this model, HyCharts are defined in Sections 4 and 5 as a multiplicative and respectively an additive interpretation of hierarchic graphs. Both interpretations are

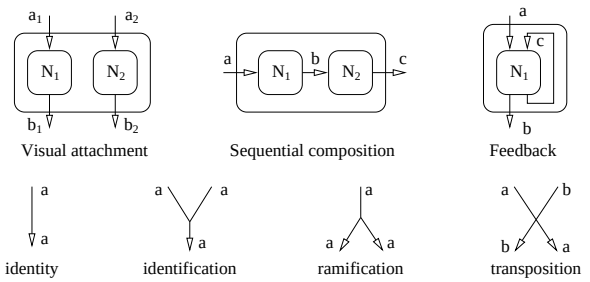

Fig. 3. The composition operators and the connectors

introduced in an intuitive way by using the example above. Finally, in Section 6 we summarize our results.

A version of this paper that goes into greater technical detail is available as a technical report [5].

2 Hierarchic Graphs as Relations

A *hierarchic graph* consists of a set of *nodes* connected by a set of *arcs*. For each node, the incoming and the outgoing arcs define the node's *interface*. In general, the arcs have associated some *type information*.

Suppose T is a set of *type names* and D is a *type function* mapping each name $t \in T$ to an associated *domain* of values D_t. Since we want to speak about incoming and outgoing arcs collectively we assume given a binary (monoidal) operation $\star$ with neutral element $\mathbf{e}$, both on type names and on the corresponding domains. For types, we obtain the set of terms given by the grammar $a := t \in T \mid \mathbf{e} \mid a \star a \mid (a)$. The $\star$ operation on type terms is assumed to be compatible with the $\star$ operation on domains, hence $D_{a\star b} = D_a \star D_b$ and $D_{a\star\mathbf{e}} = D_{\mathbf{e}\star a} = D_a$.

Now a node N with incoming arcs that collectively have type a and outgoing arcs that collectively have type b can be interpreted as a *relation* $N \subseteq D_a \times D_b$. Visually, this is represented by a box labeled by N, with an incoming arrow labeled by a and an outgoing arrow labeled by b. We shall also write $N : a \to b$.

Operators on nodes. In order to obtain graphs, we put nodes next to one another and interconnect them by using the following operators on relations: *visual attachment, sequential composition* and *feedback*. Their visual representation is given in Figure 3, top.

The *visual attachment* is achieved by extending $\star$ to an operation over nodes. Given $N_1 : a_1 \to b_1$ and $N_2 : a_2 \to b_2$ we define $N_1 \star N_2$ to be of type $a_1 \star a_2 \to b_1 \star b_2$. The *sequential composition* is the usual composition of relations. Given $N_1 : a \to b$ and $N_2 : b \to c$ we define $N_1 ; N_2$ to be of type $a \to c$. The *feedback* operation allows to connect the output of a node to the input of the same node, if both have the same type. Given $N : a \star c \to b \star c$ we define $N{\uparrow}^c_\star$ to be of type $a \to b$.

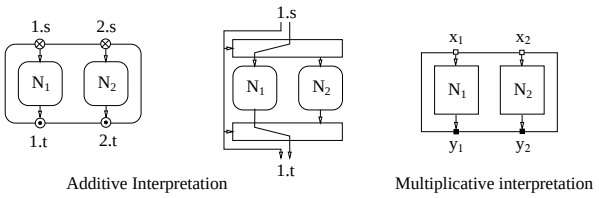

Fig. 4. The additive and multiplicative interpretations

Operators on arcs. Beside operators on nodes, we also have the following operators on arcs: *identity, identification, ramification* and *transposition.* The visual representation of these *connectors* is given in Figure 3, bottom.

The *identity* connector $\mathsf{I}_a : a \to a$ copies its input to the output. The binary *identification* connector $\vee_a : a \star a \to a$ joins two inputs together. This operator is naturally extended to k inputs. In this case it is written $\vee_a^k$. The binary *ramification* connector $\wedge^a : a \to a \star a$ copies the input information on two outputs. For its natural extension to k outputs we write $\wedge_k^a$. Finally the *transposition* connector ${}^a\mathsf{X}^b : a \star b \to b \star a$ exchanges the inputs.

To be a precise formalization of graphs, the above abstract operators and connectors have to satisfy a set of laws, which intuitively express our visual understanding of graphs. These laws correspond to *symmetric monoidal categories with feedback* enriched with branching constants, see e.g. [13]. Such a category also contains *associativity isomorphisms* for $\star$. To simplify notation, they are never written explicitly and assumed present, when necessary.

[6] shows that the *additive* and the *multiplicative interpretations* of the operators and connectors are particularly relevant for computer science.

The additive interpretation. Here $\star$ is interpreted as the *disjoint sum* operation $+$ over the variable state space $\mathcal{S}$, i.e., $\mathcal{S}+\mathcal{S} = \{1\}\times\mathcal{S} \cup \{2\}\times\mathcal{S}$. In the following, we write the tuples $(1, s)$ and $(2, s)$ concisely as $1.s$ and $2.s$. Extending $+$ to an operation over nodes, we obtain $(N_1 + N_2)(i.s) = \{i.t \mid t \in N_i(s)\}$.

Its visual notation is given in Figure 4, left. The meaning of $(N_1 + N_2)(1.s)$ is intuitively shown in Figure 4, middle. Receiving the tuple $1.s$, the sum uses the control information 1 to "demultiplex" the input and select the corresponding relation N_1; this relation is then applied to the state s to obtain the next state t; finally, the output of the relation is "multiplexed" to $1.t$.

The effect of using the additive interpretation is that (composed) nodes closely correspond to *control states* and arcs to *transitions* of automata. A node receives control *on one* of its *entry points*, i.e., its incoming arcs, and passes control on *on one* of its *exit points*, i.e., its outgoing arcs. The whole graph models the control-flow in the automaton.

The other operators and connectors are defined consistently with $+$. Feedback in the additive interpretation allows loops. Hence, it has to be used with care in order to avoid non-termination, as in usual programming. For the additive connectors we write the symbols I_a, $_k\!\!\succ\!\!\bullet_a$, $_a\!\!\bullet\!\!\prec_k$ and ${}^b_a\!\!\diagdown\!\!\!\diagup$.

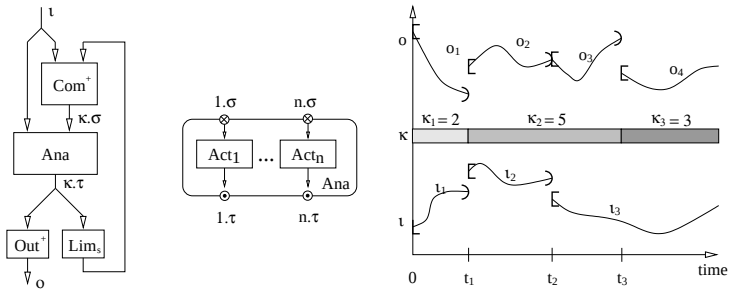

Fig. 5. The hybrid-machine computation model

The multiplicative interpretation. Here $\star$ is interpreted as the *product* operation $\times$ over sets of communication histories.[1] Extending $\times$ to an operation over nodes, we obtain $(N_1 \times N_2)(x_1, x_2) = \{(y_1, y_2) \mid y_i \in N_i(x_i)\}$.

Its visual notation is given in Figure 4, right. When we think of a system as consisting of interconnected components running in parallel, the effect of this interpretation is that arcs closely correspond to *data flow* and nodes to *system components*. A component receives data *on all* of its *input channels* and sends data along *all* of its *output channels*. Thus the graph models the data-flow in the system.

The other operators and connectors are defined consistently with $\times$. In the multiplicative interpretation, feedback allows to map the output of a component back to its input. It is defined as the greatest solution of a fixed-point equation. A unique solution is guaranteed to exist, if the output on the feedback channel c is delayed before it is fed back to the input. The multiplicative connectors are written as I_a, $\bigvee_a^k$, $\bigwedge_k^a$ and ${}^a\mathsf{X}^b$.

3 The Hybrid Computation Model

We start this section by informally explaining how our hybrid computation model works. After that the model's constituents are introduced formally.

General idea. We model a *hybrid system* by a network of autonomous *components* that communicate in a *time synchronous* way. Time synchrony is achieved by letting time flow uniformly for all components.

Each component is modeled by a *hybrid machine*, as shown in Figure 5, left. This machine consists of three basic parts: a *combinational* (or discrete) part (*Com*), an *analog* (or continuous) part (*Ana*) and a *feedback* loop.[2] The feedback models the *state* of the machine. It allows the component to remember at each moment of time t the input received and the output produced "just before" t.

[1] Communication histories basically are functions from the time domain to some data domain M, a detailed definition follows in the next section.

[2] Note the similarity of this machine model with models from control theory [3].

The combinational part is concerned with the control of the analog part and has *no memory*. It instantaneously and nondeterministically maps the current input and the fed back state to the next state. The next state is used by the analog part to select an *activity* among a set of activities (or execution modes) and it is the starting state for this activity. If the combinational part passes the fed back state without modification, we say that it is *idle*. The combinational part can only select a new next state (different from the fed back state) at distinct points in time. During the intervals between these time instances it is idle and the selection of the corresponding activity is stable for that interval, provided the input does not change discretely during the interval. The analog part describes the input/output behavior of the component whenever the combinational part is idle. Hence, it adds to the component the temporal dimension. It may select a new activity whenever there is a discrete change in the input it receives from the environment or the combinational part.

Example 2 Figure 5, right, shows the exemplary behavior of a component. The shaded box κ indicates the time periods where the combinational part idles in node i. (κ can be regarded as the control state.) At time t_1 the discrete move of the environment triggers a discrete move of the combinational part. According to the new next state received from the combinational part, the analog part selects a new activity. The activity's start value at time t_1 is as determined by the combinational part. At time t_2 there is a discrete move of the environment, but the combinational part remains idle. The analog part chooses a new flow whose start value is the analog part's output just before t_2, because this is what it receives from the combinational part at time t_2. Thus, the output has a higher order discontinuity here. At time t_3 the environment does not perform a discrete move, but the combinational part does, e.g. because some threshold is reached. Again the analog part selects a new activity, which begins with the start value determined by the combinational part. During the intervals $(0, t_1), (t_1, t_3)$ and (t_3, ∞) the combinational part is idle. □

Feedback and state. Since the input received and the output produced may change abruptly at any time t, as shown in Figure 5, right, we consider that the state of the component at moment t is the limit $lim_{x \nearrow t} \psi(x)$ of all the outputs $\psi(x)$ produced by the analog part when x approaches t. In other words, the feedback loop reproduces the analog part's output with an infinitesimal *inertia*. We say that the output is *latched*. The infinitesimal *inertia* is realized by the Lim_s part of the hybrid machine (Fig. 5, left). Its definition is $Lim_s(\psi)(t) = s$ if $t = 0$ and $Lim_s(\psi)(t) = lim_{x \nearrow t} \psi(x)$ for $t > 0$, where s is the initial state of the hybrid machine.

The *state* of the machine consists of a mapping of *latched* (or *controlled*) variable names to values of corresponding type. Let S denote the set of controlled variable names with associated domains $\{\sigma_v \mid v \in S\}$. Then the set of all possible states, i.e. the variable state space, is given by $\mathcal{S} = \prod_{v \in S} \sigma_v$. The set of controlled variable names can be split in two disjoint sets: a set P of *private* variable names and a set O of *output* (or *interface*) variable names. We write $\mathcal{S}_P$ for $\prod_{v \in P} \sigma_v$

and $\mathcal{S}_O$ for $\prod_{v \in O} \sigma_v$. Clearly, $\mathcal{S} = \mathcal{S}_P \times \mathcal{S}_O$. The latched inputs are a subset of P.

The input is a mapping of *input* variable names to values of corresponding type. Let I denote the set of input variable names with associated domains $\{\sigma_v \mid v \in I\}$. Then the set of all possible inputs is given by $\mathcal{I} = \prod_{v \in I} \sigma_v$.

The combinational part. The combinational part is a relation from the current inputs and the latched state to the next state, formally:

$$Com \in (\mathcal{I} \times \mathcal{S}_a) \to \mathcal{P}(\mathcal{S}_a)$$

where a is a sum term and $\mathcal{P}(X) = \{Y \subseteq X \mid Y \neq \{\}\}$. The sum term is due to the additive interpretation of hierarchic graphs which defines Com and gives the number of leaf nodes in Com (see Section 5.1). The computation of Com takes no time.

An important property of the relation defining the combinational part is that it is defined for all states and inputs, i.e., it is *total*. To emphasize totality, we wrote it in a functional style. Furthermore, we want that the combinational part passes the next state to the analog part only if it cannot further proceed. In other words, if $s' \in Com(i, s)$ is the next state, then $Com(i, s') = \{s'\}$, i.e., no new state $s'' \neq s'$ can be computed starting in s' with input i. We say that Com is idle for i and s'. Finally, the set $E \subseteq \mathcal{I} \times \mathcal{S}_a$ of inputs and states for which Com is not idle must be topologically closed. Together with the preceding property this guarantees that the extension of Com over time can only make discrete moves at distinct points in time. It is needed to ensure that the semantics of a hybrid machine is well-defined.

The analog part. Whenever the combinational part idles, the analog part performs an *activity*. We describe an activity by a relation Act with type

$$Act \in (\mathcal{I} \times \mathcal{S})^{\mathbb{R}_{c+}} \to \mathcal{P}(\mathcal{S}^{\mathbb{R}_{c+}})$$

where $\mathbb{R}_+$ stands for the non-negative real line. For any set M, the set $M^{\mathbb{R}_{c+}}$ stands for the set of functions $\mathbb{R}_+ \to M$ that are *continuous* and *piecewise smooth*. We say that a function $f \in \mathbb{R}_+ \to M$ is piecewise smooth iff every finite interval on the nonnegative real line $\mathbb{R}_+$ can be partitioned into *finitely* many left closed and right open intervals such that on each interval f is infinitely differentiable (i.e., f is in C^∞) for $M = \mathbb{R}$ or f is constant for $M \neq \mathbb{R}$. Infinite differentiability is required for convenience. It allows us to assume that all differentials of f are well defined. A tuple of functions is infinitely smooth iff all its components are. We also call $M^{\mathbb{R}_{c+}}$ the set of *flows* over M.

To model analog behavior in a "well behaved" way, activities must be total and *time guarded*, that is at any moment of time t, the output at t should be completely determined by the input and the state received until that moment. Formally, for all $\varphi_1, \varphi_2, \sigma_1, \sigma_2$ and t:

$$(\varphi_1, \sigma_1)\downarrow_{[0,t]} = (\varphi_2, \sigma_2)\downarrow_{[0,t]} \quad \Rightarrow \quad Act(\varphi_1, \sigma_1)\downarrow_{[0,t]} = Act(\varphi_2, \sigma_2)\downarrow_{[0,t]}$$

where by $(\sigma, \varphi)\downarrow_\delta$ we denote the restriction of σ and φ to the time interval δ.

The complete behavior of the analog part is described by a relation Ana with type:

$$Ana \in (\mathcal{I} \times \mathcal{S}_a)^{\mathbb{R}+} \to \mathcal{P}(\mathcal{S}_a^{\mathbb{R}+})$$

where $\mathcal{S}_a$ is the output type of Com and for any set M, $M^{\mathbb{R}+}$ denotes the set of *piecewise smooth* functions $\mathbb{R}_+ \to M$. Hence, the input and output of the analog part is not necessarily continuous. We call $M^{\mathbb{R}+}$ the set of *dense communication histories*.

The relation Ana is obtained by pasting together the flows of the activities associated to the nodes where the combinational part Com idles. Pasting is realized as shown in Figure 5, middle, by extending the sum operation to activities. Given a set of activities $ACT = \{Act_j \mid j \leq n\}$, their sum is defined as below[3]:

$$+_{j=1}^{n} Act_j \stackrel{\text{def}}{=} \{\ (\iota, \kappa.\sigma, \kappa.\tau) \mid \forall \delta, m.\ \kappa|_\delta = m^\dagger \Rightarrow m \leq n \wedge (\iota, \sigma, \tau)|_\delta \in (Act_m)|_\delta\}$$

where δ is a left closed right open interval, $m^\dagger$ is the extension of m to a constant function over δ, $\iota \in \mathcal{I}^{\mathbb{R}+}$ and $\kappa.\sigma, \kappa.\tau \in \mathcal{S}_a^{\mathbb{R}+}$. The tuple $\kappa.\sigma$ consists of the control flow κ which gives at each moment of time the node where the combinational part idles (see Figure 5, right) and the state flow σ which gives at each moment of time the state passed by the combinational part. The tuple $\kappa.\tau$ consists of the same control flow κ and of the state flow τ computed by the sum. For each interval δ in which the combinational part idles, the sum uses the control information $\kappa|_\delta$ to demultiplex the input $\kappa.\sigma|_\delta$ to the appropriate activity and to multiplex the output $\tau|_\delta$ to $\kappa.\tau|_\delta$. Section 5.2 will show how Ana is constructed from the activities in a HySChart by using the $+$ operator.

Note that the type of Ana assures that $(\iota, \kappa.\sigma)$ is partitioned into pieces, where ι, κ and σ are simultaneously piecewise smooth. The output histories $\kappa.\tau$ of Ana are again piecewise smooth, by the definition of Ana.

As we demand that every activity is total and time guarded, the analog part also is total and time-guarded. Furthermore, for the analog part we demand that it is *resolvable*, which means that it must have a fixed point for every state $s_0 \in \mathcal{S}_a$ and every input stream $i \in \mathcal{I}^{\mathbb{R}_{c+}}$, i.e.,

$$\exists \sigma \in \mathcal{S}_a^{\mathbb{R}_{c+}}.\sigma(0) = s_0 \ \wedge\ \sigma \in Ana(\iota, \sigma)$$

Resolvability of the analog part is needed to prove that the semantics of a hybrid machine is well-defined [5].

The component. Given an initial state s_0, the behavior of the hybrid machine is a relation Cmp between its input and output communication histories. Writing the graph in Figure 5, left, as a relational expression with the multiplicative operators results in the denotational semantics of Cmp:

$$\begin{aligned} Cmp \quad &\in \quad \mathcal{S}_a \to \mathcal{I}^{\mathbb{R}+} \to \mathcal{P}(\mathcal{O}^{\mathbb{R}+}) \\ Cmp(s) &= ((\aleph_2 \times \mathsf{I})\ ;\ (\mathsf{I} \times Com^\dagger)\ ;\ Ana\ ;\ \aleph_2\ ;\ (Out^\dagger \times Lim_s))\uparrow_\times \end{aligned}$$

where $R^\dagger$ trivially extends the combinational relation R in time, i.e., $R^\dagger(\iota) \stackrel{\text{def}}{=} \{o \mid o(t) \in R(\iota(t))\}$ for any $t \geq 0$. Out selects the output variables from the state stream.

[3] Here we use for convenience the relational notation $Act \subseteq \mathcal{I}^{\mathbb{R}_{c+}} \times \mathcal{S}^{\mathbb{R}_{c+}} \times \mathcal{S}^{\mathbb{R}_{c+}}$.

By definition, Cmp is a *time guarded* relation, because $Com^{\dagger}$, Ana, $Out^{\dagger}$, Lim_s, I and $\wr_2$ are time guarded. [5] proves that Cmp is total if Com and Ana satisfy the properties required above and if Com in connection with Ana never performs infinitely many discrete actions within a finite interval. The central part of the proof is that, due to the properties of Com and Ana, some time $\delta > 0$ passes between any two discrete moves of Com.

4 System Architecture Specification - HyACharts

The system architecture specification determines the interconnection of a system's components.

Graphical syntax. The architecture specification is a hierarchic graph, a so called HyAChart (Hybrid Architecture Chart), whose nodes are labeled with component names and whose arcs are labeled with channel names. We use a graphical representation that is analogous to the structure specifications in ROOM [11].

Semantics. As a HyAChart is a hierarchic graph, it is constructed with the operators of Section 2. Writing the graph as the equivalent relational formula and interpreting the operators in it multiplicatively directly gives the HyAChart's semantics. As $\star$ is interpreted as the product operation for sets in this interpretation, visual attachment here corresponds to parallel composition. Hence, each node in the graph is a component acting in parallel with the other components and each arc in the graph is a channel describing the data-flow from the source component to the destination component, as explained in Section 2.

The component names in the graph refer to input/output behaviors specified in other HyACharts, in HySCharts (Section 5) or with other formalisms. The channel names are the input and output variable names used in the specification of the components. The variables' types must be specified separately.

Example 3 (The HyAChart of the EHC) We now return to the HyAChart of our example system given in the introduction in Figure 1, left, and develop its semantics.

The boolean-valued channel *bend* in the figure tells the controller whether the car is in a curve. The real-valued channel $sHeight$ carries the chassis level measured by the sensors. The real-valued channel $fHeight$ carries the filtered chassis level. The real-valued channel $aHeight$ carries the chassis level as proposed by the actuators, compressor and escape valve, without environmental disturbances. The boolean-valued channels $reset$ and $dReset$ ("delayed reset") transfer the boolean reset signal to the filter. The delay component D_f ensures that the feedback is well-defined (see e.g. [10]).

The types of the filter, the control component and the delay component follow from the channels' types. The filter has type $Filter \in (\mathbb{R} \times \mathbb{B})^{\mathbb{R}+} \to \mathcal{P}(\mathbb{R}^{\mathbb{R}+})$, the controller's type is $Control \in (\mathbb{B} \times \mathbb{R})^{\mathbb{R}+} \to \mathcal{P}((\mathbb{R} \times \mathbb{B})^{\mathbb{R}+})$ and the delay has type $D_f \in \mathbb{B}^{\mathbb{R}+} \to \mathcal{P}(\mathbb{B}^{\mathbb{R}+})$. The semantics of the whole system EHC is defined as below. It is the relational algebra term corresponding to the HyAChart of

Figure 1, left.

$$EHC \in (\mathbb{B} \times \mathbb{R})^{\mathbb{R}+} \to \mathcal{P}(\mathbb{R}^{\mathbb{R}+})$$
$$EHC = ((\mathsf{I} \times Filter)\,;\, Control\,;\, (\mathsf{I} \times D_f))\uparrow_{\times}$$

Note that the user only has to draw the HyAChart and define the types of the channels. □

5 Component Specification - HySCharts

A HySChart (Hybrid StateChart) defines the combinational and the analog part of a hybrid machine. The input/output behavior of the resulting component follows from these parts as explained in Section 3.

The Graphical Syntax of HySCharts. A HySChart is a hierarchic graph, where each node is of the form depicted in Figure 6, left. Each node may have sub-nodes. It is labeled with a node name, which only serves for reference, an activity name and possibly the symbols $\to\circ$ and $\circ\to$ to indicate the existence of an entry or exit action, which is executed when the node is entered or left. The outgoing edges of a node are labeled with action names. The action names stand for predicates on the input, the latched state and the next state. They are structured into a *guard* and a *body*. The activity names refer to systems of ordinary differential (in)equations. The specification of actions and activities and their semantics is explained in detail in the following. Transitions from composed nodes express preemption. Except for activities, HySCharts look similar to ROOM-charts [11].

The Semantics of HySCharts. The semantics of a HySChart is divided into a combinational and an analog part. The combinational part follows almost directly from the diagram. The analog part is constructed from the chart with little effort. In the following we will first explain how the combinational part is derived from a HySChart and how actions are specified. Then, the analog part and continuous activities are covered.

5.1 The Combinational Part

A HySChart is a hierarchic graph and therefore constructed from the operators in Section 2. As mentioned there, interpreting the graph additively leads to a close correspondence to automata diagrams.

We may view the graph as a network of autonomous *computation units* (the nodes) that communicate with each other over directed *control paths* (the arcs). Due to the additive interpretation, each time control resides in only one (primitive) computation unit.

In order to derive the combinational part from the HySChart we now give a semantics to its nodes, i.e., to its computation units. The semantics for hierarchy and actions follows.

Computation units. Each primitive node of the HySChart represents the graph given in Fig. 6, top right. It corresponds to the relational expression:

$$\mathtt{CompUnit} \stackrel{\text{def}}{=} (+_{i=1}^{m} \mathtt{entry} + \mathsf{I})\,;\, {}_{m+1}{\succ}\bullet\,;\bullet{\prec}_{n+1}\,;\, ((+_{i=1}^{n} (\mathtt{guard}_i\,;\, \mathtt{exit}\,;\, \mathtt{body}_i)) + \mathtt{wait})$$

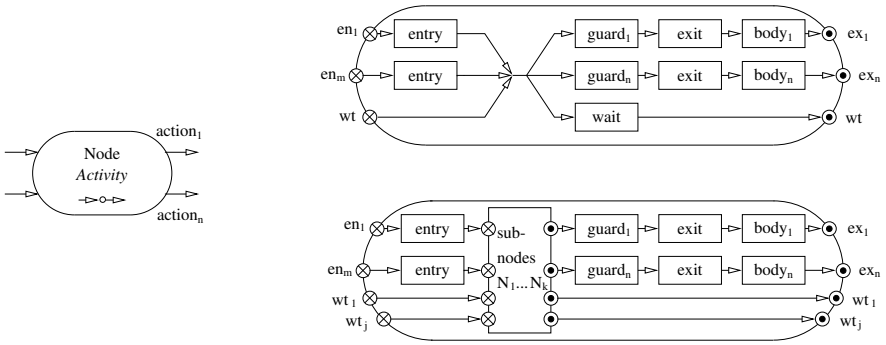

Fig. 6. Syntax and semantics of a computation unit.

According to the additive operators, it has the following intuitive meaning. A computation unit gets the *control* along one of its *entry points* $\texttt{en}_i$ and gives the control back along one of its *exit points* $\texttt{ex}_j$.

After getting control along a regular entry point, i.e., an entry point different from *wait* `wt`, a computation unit may first execute an *entry action* `entry`. Then it evaluates a set of *action guards* $\texttt{guard}_k$. If one of the guards is true, then the corresponding *action* is said to be *enabled*, the *exit action* `exit` is executed, if present, and then the action's *body* $\texttt{body}_k$ is executed. After that, the computation unit passes control to another computation unit along the exit point corresponding to the executed action.

If more than one guard is true, then the computation unit *nondeterministically* chooses one of them. Guard `wait` in the diagram stands for the negation of the disjunction of the actions' guards $\texttt{guard}_k$. Hence, if none of the guards is true, then the discrete computation is completed, and the control leaves the combinational part along the designated wait exit point `wt`. The next section shows that the analog part takes advantage of the information about the exit point to determine the activity to be executed and gives control back along the corresponding wait entry point.

Hierarchy. A composed or *hierarchic* node in the HySChart stands for the graph in Figure 6, bottom right. A principal difference to primitive nodes is that the entry points are not identified, instead they are connected to the corresponding entry points of the sub-nodes. Similarly, the exit points of the sub-nodes are connected to the corresponding exit points of their enclosing hierarchic node. Furthermore, the hierarchic node has a wait entry and wait exit point for every wait entry/exit point of the sub-nodes. When it receives control on one of them, it is directly passed on to the wait entry point of the corresponding sub-node. Thus, the wait entry point identifies a sub-node. The hierarchic node is left along a wait exit point, if a sub-node is left along its corresponding wait exit point.

Actions. An *action* a is a relation between the current input, the latched state and the next state:

$$a \subseteq (\mathcal{I} \times \mathcal{S}) \times \mathcal{S}$$

For HySCharts, actions are specified by their characteristic predicate. They are the conjunction of a precondition (the *action guard*) on the latched state and the current input and a postcondition (the *action body*) that determines the next state. The precondition implies that the postcondition is satisfiable, hence the action is enabled iff the precondition is true. We use *left-quoted variables* v‘ to denote the current input, *right-quoted variables* v' to denote the next state and *plain variables* to denote the latched state. Moreover, we mention only the changed variables and always assume the necessary equalities stating that the other variables did not change. To simplify notation further, we associate a variable c with each channel c.

For example, the action resetting the filter is defined as $dReset‘ \neq dReset \wedge dReset' = dReset‘ \wedge fHeight' = 0$. It says that each time $dReset$ is toggled, $fHeight$ should be reset to 0. We abbreviate this by $dReset? \wedge fHeight' = 0$, where $e?$ for boolean variables e stands for $e‘ \neq e \wedge e' = e‘$ and indicates that, toggling e is an event. Similarly, we define $e!$ for boolean variables e, to indicate the sending of an event. $e!$ stands for $e‘ \neq e$.

As mentioned in Section 3, the combinational part may only perform discrete state changes, on a topologically closed subset $\mathcal{I} \times \mathcal{S}$. This condition is satisfied by a HySChart defining the combinational part, if the precondition of every action in the chart identifies a topologically closed subset of $\mathcal{I} \times \mathcal{S}$. Note that in conjunction with hierarchy the action guards must be chosen with care in order to guarantee that the combinational part specified by the HySChart is total.

The additive interpretation of graphs also provides the infrastructure to easily model preemption, history variables and other concepts known from statecharts-like formalisms [6].

Semantics. If each node in the HySChart is replaced by the corresponding graph of Figure 6, right, we obtain a hierarchic graph whose nodes merely are relations. Writing the graph as the corresponding relational expression with the additive operators gives the denotational semantics of the HySChart's discrete part, i.e., the combinational part of a hybrid machine.

At the highest level of hierarchy, the hierarchic graph resulting from the HySChart has one wait entry/exit point pair for every primitive (or leaf) node in the chart. On the semantic level there is exactly one summand in the sum term a of the combinational part's type $(\mathcal{I} \times \mathcal{S}_a) \to \mathcal{P}(\mathcal{S}_a)$ for every entry/exit point pair. The analog part uses the entry/exit point information encoded in this disjoint sum to select the right activity for every node in the HySChart.

Example 4 (The EHC's Control component) To outline the utility of this approach for hybrid systems we now return to the HySChart for the controller given in the introduction. We describe the states and transitions in Figure 2 in a top-down manner. The activities, written in italics in the figure, are explained in the next section.

The computation unit Control. On the top level of the component *Control* we have two computation units, *outBend* and *inBend*. When the controller realizes that the car is in a curve, the computation unit *inBend* is entered. It is

left again when the controller senses that the car no longer is in a curve. Sensing a curve is event driven. We use the boolean variable *bend* for this purpose. The actions $n2b$ and $b2n$ are identical and very simple: $n2b \equiv b2n \equiv bend?$

The computation unit outBend. The computation unit *outBend* is refined to *inTol* and *outTol* as shown in Figure 2, top right. Control is in *inTol* as long as the filtered chassis level is within a certain tolerance interval. The compressor and the escape valve are off then. If $fHeight$ is outside this interval at a sampling point, one of the sub-nodes of *outTol* is entered. These sub-nodes are left again, when $fHeight$ is inside the desired tolerance again and the filter is reset. The actions originating from *inTol* are defined as follows:

$$t\_o \equiv w = t_s, \qquad i2i \equiv lb \leq fHeight \leq ub$$
$$i2u \equiv fHeight \leq lb, \qquad i2d \equiv fHeight \geq ub$$

An interesting aspect of *inTol* is the specification of the composed action started by the timeout t_o, which semantically corresponds to the ramification operator for hierarchic graphs. Of course, one could have used three separate transitions instead. However, in this case the visual representation would have failed to highlight the common enabling condition t_o.

Leaving the computation unit *outTol* along its exit point *reset* causes the execution of the *reset* action. This action is always enabled and defined by $reset \equiv reset!$. Note that we used here the same name for the action and its associated event. The transition $n2b$ originates from the composed node *outBend* (and from none of its sub-states). This expresses weak preemption, i.e., this transition can be taken from any sub-node of *outBend*, as long as it is not overwritten (see [5] for details).

The computation unit outTol. As shown in Figure 2, bottom right, the computation unit *outTol* consists of the computation units *up* and *down*. When the filtered chassis level is too low at a sampling point, node *up* is entered, where the compressor is on. When the level is too high, *down* is entered, where the escape valve is open. Control remains in these nodes until $fHeight$ is inside the desired tolerance again (actions $u2i, d2i$). These actions cause *outTol* to be left along the same exit point, *reset*. The actions originating from *up* and *down* are very similar to those of *inTol*, so we do not give them explicitly here.

As indicated by the symbol $\rightarrow\!\!\circ$ the nodes *inTol*, *up* and *down* have an entry action. It is defined as $entry \equiv w' = 0$ and resets w. Together with action t_o and the activity w_inc it models sampling in these nodes.

Semantics. As explained, the combinational part follows directly from the HySChart by replacing the nodes by their corresponding graphs of Figure 6, right. As every wait entry/exit point pair at the highest hierarchic level of the resulting graph corresponds to an operand in the type of the combinational part, we get that the combinational part of *Control* has type $Com \in (\mathcal{I} \times \mathcal{S}_a) \rightarrow \mathcal{P}(\mathcal{S}_a)$, where $\mathcal{S}_a = \mathcal{S} + (\mathcal{S} + (\mathcal{S} + \mathcal{S}))$.

Note that the user only has to draw the HySChart and give the definitions of the actions. The corresponding combinational part can be constructed automatically. □

5.2 The Analog Part

The second part of a HySChart's semantics is the analog part it defines. In the following we explain how this analog part is derived from the chart.

Activities. Each activity name in the HySChart refers to a system of ordinary differential (in)equations over the variables of the component. We demand that for any tuple of initial values $s \in \mathcal{S}$ and any continuous, piecewise smooth input stream $i \in \mathcal{I}^{\mathbb{R}_{c+}}$, the resulting initial value problem is solvable. This ensures that the analog part that is constructed in the following is *resolvable* as required in Section 3.

Example 5 (The activities of Control) In our example from Figure 2 the activity names written in italics stand for the following differential (in)equations:

$$w\_inc \equiv \tfrac{d}{dt}w = 1 \qquad a\_inc \equiv \tfrac{d}{dt}aHeight \in [cp_-, cp_+]$$
$$a\_const \equiv \tfrac{d}{dt}aHeight = 0 \qquad a\_dec \equiv \tfrac{d}{dt}aHeight \in [ev_-, ev_+]$$

where $cp_-, cp_+ > 0$ and $ev_-, ev_+ < 0$ are constants. For w this means that it evolves in pace with physical time. Variable $aHeight$ either increases with a rate in $[cp_-, cp_+]$ (activity *a_inc*), it decreases (*a_dec*) or remains constant (*a_const*). Note that this is all the user has to provide to specify the analog part. □

The activity $Act \in (\mathcal{I} \times \mathcal{S})^{\mathbb{R}_{c+}} \to \mathcal{P}(\mathcal{S}^{\mathbb{R}_{c+}})$ in every node is derived from the differential (in)equations in the following way: For the input stream i and the state stream s we take $s(0)$ as the initial value for the system of differential (in)equations. The activity's set of output streams then consists of the solutions of the resulting initial value problem for input stream i. For those controlled variables v, whose evolution is not determined by the initial value problem, the activity's output is equal to $s.v$, i.e., to the v component of the state stream the activity received. Hence, it remains unmodified.

Composition of Activities. To reflect the hierarchy in the HySChart the activities specified in the nodes are composed appropriately. Therefore, we extend the sequential composition operator ; to (disjoint sums of) activities:

$$Act_1 \,;\, Act_2 = \{(i, \sigma, \sigma') \mid \exists \tau.\ (i, \sigma, \tau) \in Act_1 \wedge (i, \tau, \sigma') \in Act_2\}$$

A HySChart can be seen as a tree with the primitive nodes as its leaves. The HySCharts in Figure 2, for example, has node *Control* as its root and the nodes *inBend*, *inTol*, *up* and *down* as leaves. Starting from the tree's root we derive the composed activity defined by the HySChart as follows: (We write Act_N for the (primitive) activity of node N and $CAct_N$ for the composed activity of node N, here.)

- if N has sub-nodes $M_1, \ldots, M_n$, $CAct_N \stackrel{\text{def}}{=} +_{i=1}^{n} (Act_N; CAct_{M_i})$
- if N is a primitive node, $CAct_N \stackrel{\text{def}}{=} Act_N$

The analog part is the composed activity of the HySChart's root node, i.e. $Ana = CAct_{root}$. Figure 7 and the following example explain this definition.

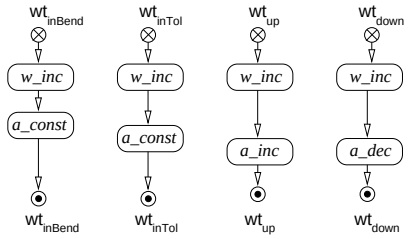

Fig. 7. The *Control* component's analog part.

Example 6 (The analog part of Control) The HySChart in Figure 2 has the analog part:

$$Ana \equiv (w\_inc\,;\, a\_const) + (w\_inc\,;(a\_const + (a\_inc + a\_dec)))$$

where we use the activity names to refer to the semantics of each activity, here. Figure 7 depicts the different paths in the associated tree. □

The entry and exit point symbols in the figure highlight that the analog part has one path for every primitive node in the HySChart. When we construct the combinational part from the HySChart, we also get one wait entry and wait exit point at its highest level of hierarchy for each primitive node. This allows to sequentially compose the combinational part with the analog part as in the semantics of a hybrid machine in Section 3. The distinct wait points allow both the combinational part and the analog part to know which node in the HySChart currently has control and to behave accordingly.

6 Conclusion

Based on a clear hybrid computation model, we were able to show that the ideas presented in [6] can smoothly be carried over to hybrid systems and yield modular, visual description techniques for such systems. Namely, the resulting techniques are HyACharts and HySCharts for the specification of hybrid system architecture and hybrid component behavior, respectively.

With an example we demonstrated the use of HyCharts and their features. Apart from many features known from statecharts-like formalisms, this in particular includes the ability to compose HySCharts with components specified with other formalisms. In our opinion such heterogeneous specifications are a key property for designing hybrid systems, as it allows to integrate techniques from different engineering disciplines.

Methodically we conceive a HySChart as a very abstract and precise mathematical model of a hybrid system. Knowing exactly the behavior of the analog part as given by a system of differential (in)equations allows us to develop more concrete models that can easily be implemented on discrete computers. For such models it is essential to choose a discretization which preserves the main properties of the abstract description.

Although this paper mainly aims at hybrid systems appearing in the context of disciplines like electrical and mechanical engineering, we think that the

continuous activities in HySCharts also make them well suited for specifying multimedia systems, such as video on demand systems. Basically HyCharts seem to be appropriate for any mixed analog/digital system where the use of continuous time is more natural than a discrete time model.

In the future we intend to develop tool support and a requirement specification language for HyCharts. For the verification of HySCharts we believe that the techniques known for linear hybrid automata [1] can easily be adapted.

Acknowledgment. We thank Ursula Hinkel, Ingolf Krüger, Olaf Müller and Jan Philipps for their constructive criticism after reading draft versions of this 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. *Theoretical Computer Science*, 138:3–34, 1995.
2. R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *CONCUR 97: Concurrency Theory*, LNCS 1243. Springer-Verlag, 1997.
3. M. Branicky, V. Borkar, and S. Mitter. A unified framework for hybrid control. Technical Report LIDS-P-2239, MIT, June 1994.
4. M. Broy. Refinement of time. In *ARTS'97*, LNCS 1231. Springer-Verlag, 1997.
5. R. Grosu and T. Stauner. Modular and visual specification of hybrid systems - an introduction to HyCharts. Technical Report TUM-I9801, Technische Universität München, July 1998.
6. R. Grosu, Gh. Stefănescu, and M. Broy. Visual formalisms revisited. In *Proc. Int. Conf. on Application of Concurrency to System Design (CSD)*. IEEE, 1998.
7. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8, 1987.
8. Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, 2nd International Symposium*, LNCS 571. Springer-Verlag, 1992.
9. N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.
10. O. Müller and P. Scholz. Functional specification of real-time and hybrid systems. In *Proc. Hybrid and Real-Time Systems (HART)*, LNCS 1201. Springer, 1997.
11. Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley and Sons Ltd, Chichester, 1994.
12. T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In *Proc. Hybrid and Real-Time Systems (HART'97)*, LNCS 1201. Springer-Verlag, 1997.
13. Gh. Stefănescu. Algebra of flownomials. Technical Report TUM-I9437, Technische Universität München, 1994.