URL: http://www.elsevier.nl/locate/entcs/volume28.html 20 pages

Prototyping of Hybrid Systems
– From HyCharts to Hybrid Data-Flow Graphs – [1]

Thomas Stauner [a] and Christoph Grimm [b]

[a] *Institut für Informatik, Technische Universität München, D-80290 München, Germany, Email: stauner@in.tum.de*

[b] *Fachbereich Informatik, Johann Wolfgang Goethe-Universität, D-60054 Frankfurt am Main, Germany, Email: grimm@ti.informatik.uni-frankfurt.de*

Abstract

In this paper, a translation of the visual description technique *HyCharts* to *Hybrid Data-Flow Graphs (HDFG)* is given. While HyCharts combine a data-flow and a control-flow oriented formalism for the specification of the architecture and the behavior of hybrid systems, HDFG allow the efficient and homogeneous internal representation of hybrid systems in computers and their automatic manipulation. HDFG represent a system as a data-flow network built from a set of fundamental functions.

The translation permits to combine the advantages of the different description techniques: The use of HyCharts for specification supports the abstract and formal interactive specification of hybrid systems, while HDFG permit the tool based optimization of hybrid systems and the synthesis of mixed-signal prototypes.

1 Introduction

According to investigations, 50% of the problems that occur within delivered embedded systems and that are reported by the customers are caused by misconceptions in capturing the requirements [1]. The ability to produce prototypes at early stages of the system development process can help to increase clarity about the requirements between system engineers and customers and to improve the dialog between them. The prototyping of pure digital systems is well-established. Tools like MATLAB/Stateflow [15] or MatrixX/BetterState [8] permit the code generation from abstract specifications, often given as a combination of statecharts and block diagrams. Furthermore, there are tools

[1] This work was supported with funds of the Deutsche Forschungsgemeinschaft under reference numbers Br 887/9 and Wa 357/14 within the priority program *Design and design methodology of embedded systems.*

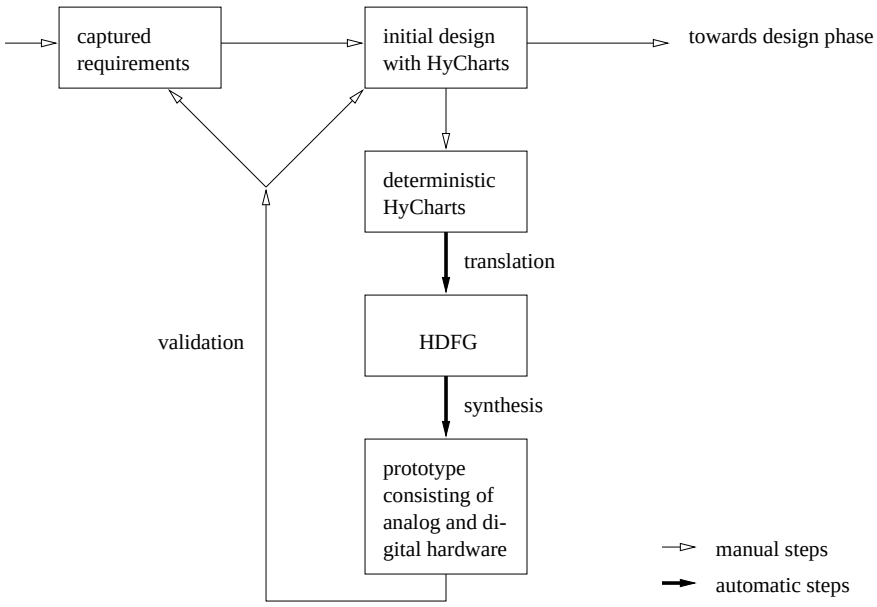

Fig. 1. Analysis phase with prototyping.

such as Ptolemy [9] which also support the design of application-specific digital hardware. However, not all required functions can be implemented on a microprocessor or using digital hardware. Most notably the amplification and filtering of small signals before an A/D conversion has to be implemented by analog circuits. The design of such mixed analog/digital implementations is only supported by very few tools [2,10]. Presently however, these tools do not start the design process with abstract specifications such as statecharts, or statechart variants like HyCharts. This makes them less suitable for prototyping.

In this paper, we give a translation from a subclass of *HyCharts*, namely deterministic HyCharts, to *hybrid data-flow graphs (HDFG)*. This translation allows us to combine the expressive power of HyCharts with the ability of HDFG to optimize and map descriptions of hybrid systems onto analog and/or digital hardware, as implemented in the tool KANDIS/2 [11]. In Fig. 1, the application of the translation in a rapid-prototyping methodology is shown.

The difficulty in the translation is, that HyCharts [5] provide two description techniques, one for the specification of the data-flow, or the *architecture*, of a system (*HyACharts*) and one for the control-flow, or behavior, of a system's components (*HySCharts*). This concept of views, known in software engineering e.g. from UML [12], enables the user to choose appropriate views for the specification of a system's functionality. On the other hand, in hybrid data-flow graphs both views, the architectural and the behavioral view, are expressed in a unified, data-flow like way, because such a unified representation permits the free choice between different implementations (microprocessor, ASIC, analog circuit), independent from the description technique used for the description of the intended functionality.

Overview.

The remainder of this paper is organized as follows. In the next two sections HyCharts and HDFG are introduced in more detail. Section 4 defines the translation of HyCharts to HDFG. In Section 4.1 the translation principle for HyACharts is explained, while Section 4.2 presents the detailed translation of HySCharts. Two examples illustrate the translations. A conclusion ends the paper.

2 HyCharts

HyCharts [6,5] consist of two graphical description techniques that are modular and based on a clear computation model. HyCharts regard a system as a network of components communicating over directed channels in a time-synchronous way. HyCharts come in two variants: *HyACharts* for the specification of the system architecture and *HySCharts* for the specification of the behavior of a hybrid system's components. HySCharts are similar to the Statechart variant ROOMcharts [13], but extend them with continuous activities for the specification of analog behavior.

Each component which is specified by a HySChart is implemented by a *hybrid machine*, as graphically shown in Fig. 2 as a HyAChart. This machine consists of five parts: a time extended *combinational* (or discrete) part ($Com^\dagger$), an *analog* (or continuous) part (Ana), a *feedback* loop, an infinitesimal delay (Lim_z), and a projection ($Out^\dagger$). The labels at the channels indicate their types. The feedback models the *state* of the machine. Together with Lim_z it allows the component to remember at each moment of time t the input received and the output produced "just before" t.

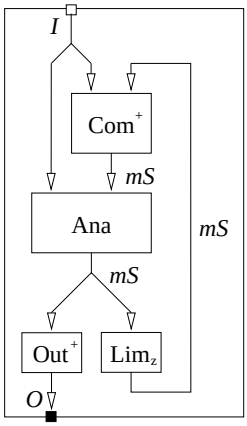

Fig. 2: HyAChart for the hybrid machine computation model.

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*, which specifies the continuous evolution of the machine's variables, 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. It may select a new activity whenever there is a discrete change in the input it receives from the environment or the

combinational part.

The data state space of the machine contains the output space and a copy of the input space. The time extended projection $Out^\dagger$ selects the output space from the state space of the machine and makes it visible outside. The copy of the input space is used to store at any point in time t the external inputs $i \in \mathcal{I}$ received just before t. These latched inputs enable the combinational part to detect and to react to discrete changes in the input. They are updated by the analog part.

The combinational and the analog part of a hybrid machine are specified with a HySChart. HyACharts specify the structuring of systems into components, like in Fig. 2 where the HyAChart defines the interconnection of the components of the machine model. From a syntactic point of view, HyACharts and HySCharts are both constructed from primitive nodes by the application of node operators and arrow operators, which we also call *connectors*, to build a *hierarchic graph*. According to the ideas in [14] and [7] these graphs are given a *multiplicative interpretation* for HyACharts, while for HySCharts the graphs are interpreted by an *additive interpretation* of the operators. The syntax of the node and the arrow operators of HyCharts can be given both in a graphical and in a textual manner. While the system designer uses the graphical representation of HyACharts and HySCharts, the textual respresentation is handsome for further processing.

In the following we briefly describe the semantics of the multiplicative and the additive operators. Note that a time extended version of the additive operators is defined in Section 4.2.1.

Multiplicative semantics.

In the semantics based on the multiplicative interpretation, each node is active and produces output on its output arrows depending on the input received along its input arrows. Formally, a node is seen as an input/output relation $n \subseteq \mathcal{I}^{\mathbb{R}_+} \times \mathcal{O}^{\mathbb{R}_+}$ that nondeterministically maps an input stream in $\mathcal{I}^{\mathbb{R}_+}$ to a non-empty set of output streams in $\mathcal{O}^{\mathbb{R}_+}$, where $\mathcal{I}$ and $\mathcal{O}$ may be tuples and $A^{\mathbb{R}_+}$ denotes the set of all *piecewise smooth* functions that map the non-negative reals $\mathbb{R}_+$ to the set A.[2] Relations are used instead of functions to be able to express nondeterminism. We require that the relation is total in its input, i.e. there is a $(a, b) \in n$ for every input $a \in \mathcal{I}^{\mathbb{R}_+}$, and time guarded, i.e. the output up to time t is completely determined by the input up to t. For notational convenience, we use the same name (or symbol) for a node (or operator) and its associated relation (or relational operator). Note, however, that the names and symbols are syntactic entities whereas the relations and relational operators are semantic entities. We assume that data-flows are time

[2] We say that a function $f \in \mathbb{R}_+ \to M$ is piecewise smooth iff every finite interval on the non-negative 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.

synchronous, i.e. time flows in the same way for all components and channels. Therefore the set $(A \times B)^{\mathbb{R}_+}$ is isomorphic to the set $A^{\mathbb{R}_+} \times B^{\mathbb{R}_+}$. The semantics of the operators is then given as follows:

- $n_1; n_2 = \{(a, c) \mid \exists b.(a, b) \in n_1 \wedge (b, c) \in n_2\}$
- $n_1 \times n_2 = \{((a_1, a_2), (b_1, b_2)) \mid (a_1, b_1) \in n_1 \wedge (a_2, b_2) \in n_2\}$
- $n\uparrow_\times^C = \{(a, b) \mid \exists c.(b, c) \in n(a, c)\}$, where $n \subseteq (A \times C)^{\mathbb{R}_+} \times (B \times C)^{\mathbb{R}_+}$

Hence, $n_1; n_2$ corresponds to sequential composition of relations, $n_1 \times n_2$ to independent parallel composition, and $n\uparrow_\times^C$ to a fixpoint calculation, i.e. recursion. The semantics of the arrow operators is: $\mathsf{I}_A = \{(a, a) \mid a \in A^{\mathbb{R}_+}\}$ for identity, $\curlyvee_A = \{((a, a), a) \mid a \in A^{\mathbb{R}_+}\}$ for identification of two inputs, $\curlywedge^A = \{(a, (a, a)) \mid a \in A^{\mathbb{R}_+}\}$ for ramification (or copying) of the input and ${}^A\mathsf{X}^B = \{((a, b), (b, a)) \mid (a, b) \in (A^{\mathbb{R}_+} \times B^{\mathbb{R}_+})\}$ for transposition of the inputs. Note that the connectors are needed in HyCharts, because the composition of nodes is not defined by using channel or port names. Instead the *position* of the arrows is important.

Additive semantics.

In the additive semantics, the graphs are viewed as control flow graphs. The intuition is that at any point in time, the control resides in *exactly one* node of the graph. Each node receives the control at one of its disjoint input arrows and forwards it on one of its disjoint output arrows. As only one arrow should transport the control and exactly one node should have it, the basic composition operation here is the disjoint sum, as given below. The *control*, or *program state*, is represented as $k.s$, where s is the data state in the data state space $\mathcal{S}$ and $k \in \mathbb{N}$ (the program counter) denotes the control state.[3]

As usual, the disjoint sum of sets $A + B$ is defined as $A + B = \{l.a \mid a \in A\} \cup \{r.b \mid b \in B\}$, so that elements $a \in A \cup B$ are identified by the prefix l or r as belonging to A or B, respectively. For multiple sums $A_1 + \ldots + A_m$, we use prefixes $0 < k \leq m$ to denote that a in $k.a \in A_1 + \ldots + A_m$ stems from A_k. Moreover, instead of adding m times the same set $\mathcal{S}$, we simply write $m\mathcal{S}$. Note that elements $k.s \in m\mathcal{S}$ can be encoded as tuples (k, s) for $k \in \{1, \ldots m\}$ and $s \in \mathcal{S}$.

In the additive semantics each node is interpreted as a relation $n \subseteq (\mathcal{I} \times \ell\mathcal{S}) \times m\mathcal{S}$, where $(x, a, b) \in n$ means that if the input $x \in \mathcal{I}$ is read in the control a, b can be the next control. Note that we again use the same name (or symbol) for a node (or operator) and its associated relation (or relational operator) in order to simplify notation. The semantics of the node operators is then:

- $n_1; n_2 = \{(x, a, c) \mid \exists b.(x, a, b) \in n_1 \wedge (x, b, c) \in n_2\}$, i.e. sequential composition of relations which both get the same external input x.

[3] Actually k refers to arrow numbers. However, in the additive interpretation arrow numbers closely correspond to a program counter, indicating *where* control is.

- $n_1 + n_2 = \{(x, l.a, l.b) \mid (x, a, b) \in n_1\} \cup \{(x, r.a, r.b) \mid (x, a, b) \in n_2\}$, where $l.a$ or $r.a$ denote that a stems from the left or right summand, respectively. Therefore, in the additive composition only one of the nodes is active and determines the next control, depending on the current control and input.
- $n\uparrow_+^i = n_{l,l} \cup n_{l,r}; n_{r,r}^*; n_{r,l}$ where $n \subseteq \mathcal{I} \times (h\mathcal{S} + i\mathcal{S}) \times (\ell\mathcal{S} + i\mathcal{S})$, $n_{i,j} = \{(x, a, b) \mid (x, i.a, j.b) \in n\}$ for $i, j \in \{l, r\}$ and m^* is the arbitrary, but finite iteration of m. Hence, $n\uparrow_\times^i$ corresponds to a while-loop: n is repeated until control is in $\ell\mathcal{S}$.

The semantics of the connectors is as follows: $\mathsf{I}_m = \{(x, a, a) \mid x \in \mathcal{I} \wedge a \in m\mathcal{S}\}$ for m-ary identity, ${}_m\!\succ\!\bullet = \{(x, k.a, a) \mid x \in \mathcal{I} \wedge k.a \in m\mathcal{S}\}$ for identification, or joining, of m input arrows, $\bullet\!\prec_m = \{(x, a, k.a) \mid x \in \mathcal{I} \wedge k.a \in m\mathcal{S}\}$ for m-ary ramification, or splitting of one input arrow into m output arrows , and ${}_j^i\!\!\diagdown\!\!\diagup = \{(x, k.a, (i+k).a) \mid x \in \mathcal{I} \wedge k.a \in j\mathcal{S}\} \cup \{(x, (j+k).b, k.b) \mid x \in \mathcal{I} \wedge k.b \in i\mathcal{S}\}$ for (i, j)-ary transposition, commuting its first i input arrows with the last j input arrows. Note that the arrow numbers can be regarded as program counters in our machine model.

3 Hybrid Data-Flow Graphs (HDFG)

HDFG [4,3] are a graph-based respresentation technique that is homogeneous and simple even for hybrid systems. This permits the efficient *intermediate representation* of such systems and the formulation of methods that deal with a hybrid system uniformly, most notably for the purpose of computer-based optimization.

Like HyCharts, HDFG regard a system as a network of components (represented by nodes V) connected by edges e, which transmit signals $\sigma_e \in T \to S_e$, where the time base T is an arbitrary fully ordered set, and S_e gives the possible values of the signal along edge e. For a set E of edges we also write S_E to denote the set product of the S_e for $e \in E$. In difference to HyCharts, HDFG only focus on data-flow, there is no specific representation for control-flow, and the function of the components in a data-flow network can only be chosen from a small set of predefined functions, such as integration over time (*intdt*), elementary delays, common arithmetic and boolean functions, and two primitives for control-flow (see below). A HDFG is specified by a set of nodes N, a set of edges E and a time-base T, which defines the points in time, where inputs are consumed and outputs are produced. Each node v of a HDFG is specified by $v = (E_{in}, E_{out}, E_a, f, a)$, where:

- E_{in} is a set of edges (*in-edges, in-ports*), whose signals are arguments of the function f.
- E_{out} is a set of edges (*out-edges, out-ports*), whose signals are results of f.
- $E_a \subseteq E_{in}$ specifies the in-edges whose signals' current values $\sigma_{E_a}(t)$ are used by f to compute the current output. Hence, these signals are used without delay. $E_z = E_{in}/E_a$ are in-edges, from whose signals at time $t \in T$ only

values at t' with $t' < t$, i.e. old values, are used. We also call edges in E_z *storing edges*. Nodes with $E_{in} = E_a$ are called *combinational nodes*, nodes with $E_{in} \neq E_a$ are nodes with an internal state.

- f is a function $f \in (T \to S_{E_{in}}) \to (T \to S_{E_{out}})$ from a set of predefined functions F. Informally, f maps segments or discrete values of input-signals to segments or discrete values of output-signals. This permits the use of continuous signal-processing functions such as integration.
- a is an *activation rule*, which consists of two conditions c_t, c_a. $c_t \in T \to \mathsf{Bool}$ defines a time-base $T_v = \{t \in T \mid c_t(t) = true\}$ of v. $c_a \in S_{E_a} \times S_{E_a} \to \mathsf{Bool}$ is evaluated for all $t \in T_v$. If $c_t = false$, the output is defined by a signal-type conversion function (see below). Otherwise, if $c_a = false$, the output remains unchanged. If $c_a = true$, the output is defined by f, and the node is called *active*.

As nodes v_o, v_d connected by an edge can have different activation rules a_o, a_d, the type of an output signal $\sigma_o \in T_o \to S$ of v_o is not necessarily the same as the type of the input signal $\sigma_d \in T_d \to S$ required by the destination v_d. In HDFG, the signal-type is implicitly converted by a signal-type conversion function $conv \in (T_o \to S) \to (T_d \to S)$. For $t \in T_d$ the signal $\sigma_d = conv(\sigma_o)$ is defined by $conv$ as $\sigma_d(t) = \sigma_o(t)$ if $t \in T_o$. Otherwise, if $t \notin T_o$, $\sigma_d(t) = \sigma_o(t_o^-(t))$, where $t_o^-(t)$ denotes the last point in the time base T_o before t, where node v_o was active.

A HDFG $g = (N, E, T)$ consists of a set of nodes N, a set of edges E and a time base T. Let $E_s = \{e \in E \mid e \in E_z(n), n \in N\}$ be the set of all edges which appear in E_z of a node of N, i.e. the set of all storing edges, σ_i the signal on the i-th edge of the m edges of $E_s = \{e_1, \ldots e_m\}$, and $t_{z,i}(t)$ be the last point in T before and including t, where the destination of $e_i \in E_s$ has been active. A HDFG then defines the following dynamic behavior: [4]

(i) At the start $t_0 \in T$, all nodes are active, and $t_{z,i}(t_0) = t_0$ for all $e_i \in E_s$. The state Z_0 of a system at $t_0 \in T$ is then given by the current values of the signals of the edges in E_s: $Z_0 = \{\sigma_1(t_0), \ldots, \sigma_m(t_0)\}$.

(ii) Let $t \in T$ with $t > t_0$ be a point in time at which at least one node of the HDFG becomes active. All combinational nodes map their current inputs to their current outputs without any delay. Active nodes with internal state use their storing edges' values of the last point in time strictly before t, when they were active, and the current values of their other in-edges to compute their output. The output of inactive nodes remains unchanged, according to the signal type conversion.

(iii) At time t, the state is then given by the values of the signals of edges

[4] For simplicity, we argue on single values here. Segments can be regarded as consisting of infinitesimal small pieces, where they are equal to a single value. Furthermore, we assume edges with only one destination, and no states introduced by the signal-type conversion function.

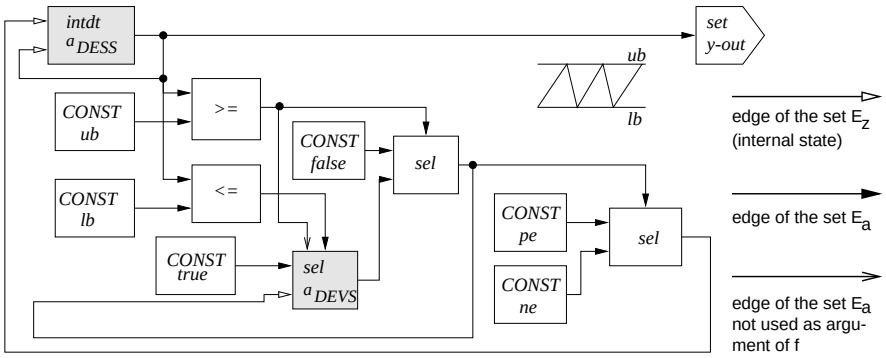

Fig. 3. Representation of a saw tooth generator by a HDFG.

$E_s = \{e_1, \ldots, e_m\}$ at the last points in time before and including t, when their destination has been active: $Z_t = \{\sigma_1(t_{z,1}(t)), \ldots, \sigma_m(t_{z,m}(t))\}$

Cycles in HDFG are permitted, if it can be ensured, that there are no combinational cyclic dependencies. This can be ensured, if each cycle contains at least one edge in E_z or one node with the function *intdt*.

For the description of hybrid systems, we use the predefined activation rules a_{DESS}, a_{DTSS}, and a_{DEVS}. $a_{DESS} ::= (c_t = true\, \forall t \in T; c_a = true)$ permits the modeling of **d**ifferential **e**quation **s**pecified **s**ystems. $a_{DTSS} ::= (c_t = true\, \forall t \in T \mid t = n * t_c, n \in \mathbb{N}, t_c \in \mathbb{R}; c_a = true)$ permits the description of **d**iscrete **t**ime **s**pecified **s**ystems, where time is measured in units of t_c. $a_{DEVS} ::= (c_t = true\, \forall t \in T; c_a = true$ if $\sigma_{E_a}(t_a) \neq \sigma_{E_a}(t_z^-))$, where t_a denotes the actual time, and t_z^- denotes the time of the last activation before t_a, permits the representation of **d**iscrete-**ev**ent specified **s**ystems. It specifies that the node is active whenever a signal in E_a changes. [5]

Two special functions f *select* and *iterate* permit representation of control-flow in a functional way. *iterate* does a fix-point iteration on a HDFG, which is an additional parameter, and *select* selects one from a set of inputs. HDFG can be structured hierarchically. In this case, the function f of a node v is described by a HDFG, and not by one of the pre-defined functions.

In Fig. 3, the dynamic behavior of a saw tooth generator is described by a HDFG. Starting from zero, its output uniformly increases until the upper bound *ub* is reached. Then is decreases uniformly down to the lower bound *lb*. When it is reached the process starts again. Note, that we do not have to specify an activation rule a for all nodes; in this example, a is only relevant for nodes with internal states. In the figure, the two nodes with internal states are shaded in grey. The node with $f = intdt$ has activation rule $a = a_{DESS}$, which activates the node continuously, independent from its inputs. The other node with internal state is activated only, when $\sigma_{E_a}(t_a) \neq \sigma_{E_a}(t_z)$.

KANDIS/2 [10] supports the design of mixed-signal circuits from HDFGs.

[5] For smooth signals, we define $c_a ::= true$ whenever the signal is not constant, i.e. when its derivative is different from zero.

One possible architecture that implements the behavior specified by the HDFG in Fig. 3 is shown in Fig. 4, left. Other architectures (pure digital, etc.) can be chosen in an interactive way; the design-methodology is described in more detail in [3,4].

4 Translating HyCharts to HDFG

The translation from HyCharts to HDFG presented here is limited to HyCharts of the following kind:

- Every *primitive* component in the HyAChart, i.e. every component which is not refined into further subcomponents, is defined by a HDFG with activation rule a_{DESS}. Thus in contrast to usual HyACharts components may not be given by arbitrary stream processing functions.
- Alternatively, the primitive component must be specified by a *deterministic* HySChart whose discrete actions and continuous activities are given by HDFGs.
- Every feedback loop in a HyAChart (and in all of its subcharts in the case of hierarchy) contains a non-zero delay.

  Furthermore, we allow feedback loops without delay which do have a unique fixed point for any given set of initial values. To ensure that the resulting HDFG is well defined we require an integrator block *intdt* or an infinitesimal delay Lim_z, like in our machine model (Fig. 2), in the feedback loop. Uniqueness of the fixed point ensures that the resulting HyAChart is deterministic if all its components are.
- HyACharts do not contain identification connectors. The identification connectors introduce synchronization between channels to HyACharts, which is related to synchronization in Petri nets. It is unclear how this is realized operationally. In fact, this connector has not been used in any application of HyACharts so far.

We think these limitations still cover most deterministic systems of practical interest. Note that zeno behavior, i.e. behavior in which a system performs infinitely many discrete steps within a finite time interval, is explicitly excluded by the type of HyCharts, restricting them to piecewise smooth inputs and outputs. Nevertheless it is syntactically possible to specify zeno systems with HyCharts. Of course, such systems cannot be implemented. Thus, the prototype generated from such a HyChart will be a "best effort" implementation, i.e. it will work as fast as possible, but it will not have zeno behavior.

4.1 Translation of HyACharts

Due to the data-flow paradigm which is the basis for HyACharts as well as for HDFG, the translation from HyACharts to HDFG if fairly straightforward. Therefore, we only sketch the principle here.

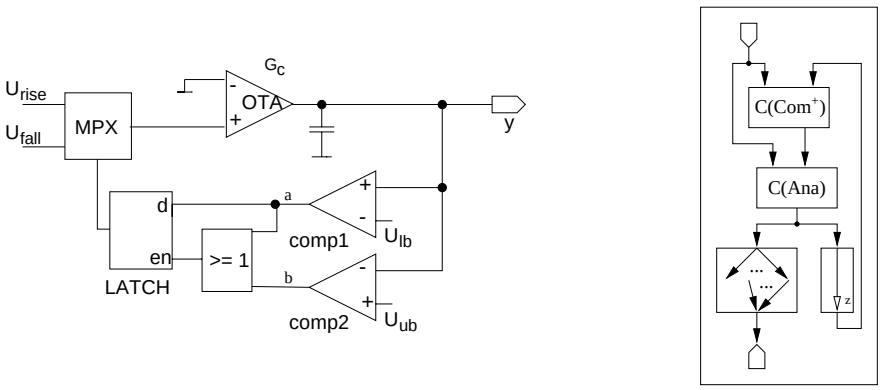

Fig. 4. An architecture designed with KANDIS/2 (left) and the translation of the hybrid machine model (right).

Starting with the textual representation of a HyAChart with the operators on hierarchic graphs given in Section 2, the translation to HDFG is defined by structural induction. The allowed multiplicative connectors, identity, ramification and transposition, can be regarded as predefined nodes.

- As provided by the assumption every primitive node in the HyAChart either is given by a HDFG or by a HySChart. In the first case the HDFG is the node's translation. In the second case we have to apply the translation to the HyAChart for the hybrid machine model (Fig. 2). The machine's combinational and analog part are translated to HDFGs in the way defined in Section 4.2. The HDFG for Lim_z and $Out^\dagger$ are explained in the example below.
- The composition of nodes with one of the multiplicative hierarchic graph operators, sequential composition, parallel composition and feedback, is translated to a HDFG which couples the HDFG translations of the nodes in the respective way, e.g. by feeding back one output in case of feedback.
- The translation of each of the allowed multiplicative connectors results in a HDFG whose input and output ports are connected in the respective way, e.g. copying the input in case of ramification. Note that the processing functions of the HDFG nodes appearing in the translation of the connectors all are the identity function.

All HDFGs resulting from the translation have activation rule a_{DESS}. Note that due to hierarchy in HDFG each HDFG can be regarded as a HDFG node at the next hierarchic level.

Example: Translation of the machine model.

The HDFG resulting from the translation of the HyAChart for the machine model (Fig. 2) is depicted in Fig. 4, right. Note that we have resolved most of the hierarchy in the resulting HDFG to increase clarity. $C(Com^\dagger)$ is the translation of the time extended combinational part and $C(Ana)$ is the translation of the analog part. The next section explains how these HDFG

are obtained from the HySChart, which specifies them.[6]

The bottom left box is the translation of $Out^\dagger$. In it, the received tuple representing the program state is split into its components. Then only those tuple components which are part of the output state space are combined to a new tuple. The bottom right box is the translation of the infinitesimal delay Lim_z. Its HDFG translation is a storing edge (in E_z), which reproduces its input with an infinitesimal delay (for activation rule a_{DESS}). The initial value z of the edge for $t = 0$ is written besides its arrowhead. The HDFGs for $Out^\dagger$ and Lim_z both have a_{DESS} as activation rule.

4.2 Translation of HySCharts

In this section we define the translation of the time-extended combinational part and the analog part, which are both specified by a HySChart, to HDFG and give an example. The translation is based on a time extended-additive semantics of hierarchic graphs defined in Section 4.2.1.

4.2.1 The Timed-Extended Combinational Part

The combinational part Com is defined by a hierarchic graph under the additive interpretation. This graph is derived from the HySChart in the way explained in [5]. According to the additive interpretation of the graph operators, Com itself is a relation without time. It is extended in time by the operator †, which is defined as $R^\dagger = \{x \mid \forall t \in \mathbb{R}_+ . x(t) \in R\}$ for relation R. In HDFG there are no nodes without time. Hence, it is not possible to translate Com and then apply a translated time-extension operator. However, it is possible to define a time extended version of the additive graph operators such that time extension is compositional, $(A \ominus B)^\dagger = A^\dagger \ominus^\dagger B^\dagger$ for $\ominus \in \{;, +, \uparrow_+\}$. Due to compositionality the translation to HDFG can then be defined on time-extended additive hierarchic graphs.

Time-extended additive semantics.

For time extended relations $N_1 \subseteq (\mathcal{I} \times h\mathcal{S} \times \ell\mathcal{S})^{\mathbb{R}_+}$ and $N_2 \subseteq (\mathcal{I} \times m\mathcal{S} \times n\mathcal{S})^{\mathbb{R}_+}$ we formally define:

- $N_1 ;^\dagger N_2 = \{(\iota, \phi, \psi) \mid \exists \rho . (\iota, \phi, \rho) \in N_1 \ \wedge \ (\iota, \rho, \psi) \in N_2)\}$
- $N{\uparrow_+^i}^\dagger = \{(\iota, \phi, \psi) \mid \forall t . (\iota, \phi, \psi)(t) \in (N|_t){\uparrow_+^i}\}$, where $N|_t$ is the restriction of N to time t. Hence, additive feedback for time extended relations is defined pointwise.

[6] Apart from the edges transmitting the external input and the output, all edges in the figure carry values of type $\mathbb{N} \times \mathcal{S}$, thus encoding the control and data state. At the interface of $C(Com^\dagger)$ and $C(Ana)$ we implicitly split this tuple into control and data state, because this simplifies the definition of $C(Com^\dagger)$ and $C(Ana)$ in the next section.

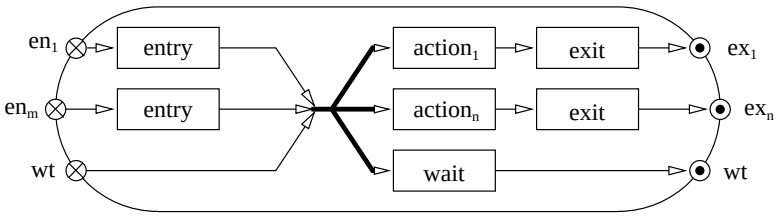

Fig. 5. Hierarchic graph for a primitive state.

- $N_1 +^\dagger N_2 = \{(\iota, \kappa.\sigma, \kappa'.\sigma') \mid \exists \sigma'_l, \kappa'_l, \sigma'_r, \kappa'_r.$
$$(\iota, \kappa.\sigma, \kappa'_l.\sigma'_l) \in \bar{N}_1 \ \wedge$$
$$(\iota, (\kappa - h).\sigma, \kappa'_r.\sigma'_r) \in \bar{N}_2 \ \wedge$$
$$\forall t.\kappa(t) \leq h \Rightarrow \kappa'.\sigma'(t) = \kappa'_l.\sigma'_l(t) \ \wedge$$
$$\kappa(t) > h \Rightarrow \kappa'.\sigma'(t) = (\kappa'_r + \ell).\sigma'_r(t)\}$$
where $\bar{R}$ is the extension of $R \subseteq (\mathcal{I} \times n\mathcal{S} \times m\mathcal{S})^{\mathbb{R}_+}$ to the type $(\mathcal{I} \times (\mathbb{N} \times \mathcal{S}) \times (\mathbb{N} \times \mathcal{S}))^{\mathbb{R}_+}$ by $\bar{R} = \{(\iota, \kappa.\sigma, \kappa'.\sigma') \mid \forall \delta.(\kappa|_\delta \leq n \Rightarrow (\iota, \kappa.\sigma, \kappa'.\sigma')|_\delta \in R|_\delta) \ \wedge \ (\kappa|_\delta > n \Rightarrow \kappa'.\sigma'|_\delta = f|_\delta\}$, where $\varphi|_\delta$ denotes the restriction of a stream, a stream tuple or a set of streams φ to interval δ, and f is a constant function mapping the real numbers to an arbitrary "dummy" element in $\mathbb{N} \times \mathcal{S}$. As long as a relation only gets well-defined inputs (in $n\mathcal{S}$), the value of the dummy element does not play any role. With $\kappa - n$ and $\kappa + n$ we denote the application of the respective arithmetic operation to every $\kappa(t)$. Furthermore, we write $\kappa \leq n$ and $\kappa > n$ to denote that $\kappa(t)$ is in the respective relation to n for all considered t.

  Informally, the definition expresses that the output of $N_1 +^\dagger N_2$ is as determined by N_1 whenever the received control state refers to the input arrows of N_1 (i.e. $\kappa(t) \leq h$). If it refers to the input arrows of N_2, the output is as determined by that node. Note that N_1 and N_2 are permanently active in this definition, even if they are not selected for output.

Regarding the connectors as predefined nodes, their time extension is defined pointwise by applying the time extension operator to them.

HySCharts and the combinational part.

Before we define the translation a few words on the derivation of the combinational part from a HySChart are necessary (for details see [5]). The basic step of the derivation is to replace each primitive node in the HySChart, i.e. each node which is not refined into further subnodes, by the hierarchic graph in Fig. 5. The additive ramification operator $\bullet\!\!<$, drawn with bold lines in the figure, receives control and non-deterministically outputs it on one of its output ports.[7] Connected to these ports we have *actions*, which consist of

[7] Note that the incoming and outgoing arrows of an additive hierarchic graph are also called ports.

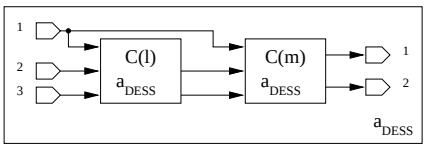

Fig. 6. HDFG for additive sequential composition.

a *guard* and a *body*. The body is executed and possibly changes the values of some variables if the action body is true. Only in this case, the action passes control further. In the context of our translation to HDFG we demand that all action guards are given as boolean functions of HDFG and that all action bodies are sequences of assignments, given as arithmetic functions of HDFG, excluding integration and derivation. Furthermore, we limit our translation to deterministic HySCharts. This guarantees that only one of the actions $action_1, \ldots, action_n$ or *wait* can be enabled. The *wait* action is defined as the negation of the disjunction of the action guards of $action_i$. The body of *wait* is the identity. If no action is enabled, a node passes control on its wait exit port wt. Due to the structure of the graph defining Com, only these wait ports are visible outside. The analog part is connected to them. The labels *entry* and *exit* refer to entry and exit actions. This special type of actions only has an action body, their guard is true.

For deterministic HySCharts the derivation of the hierarchic graph ensures that a ramification connector is always followed by actions. We therefore regard $\bullet\!\!<_m{}^{\dagger};^{\dagger}(+^{\dagger}_{i=1}{}^{m} a_i)$ as a primitive in our translation. This way we need not define a translation for ramification alone, which would require to introduce non-determinism that is resolved later anyway.

Translation of time extended additive hierarchic graphs.

We now define the translation of the time extended combination part by induction on the structure of the corresponding time extended additive hierarchic graph:

- $n = l;^{\dagger}m$ (time extended sequential composition). Fig. 6 depicts the corresponding HDFG. By convention the translation to HDFG uses the first input port of a HDFG node for the external input (type $\mathcal{I}$), the second one to transmit the data state (type $\mathcal{S}$) and the third one to transmit the control state (type $\mathbb{N}$). Moreover, the first output port carries the next data state and the second one carries the next control state. [8] As all nodes in a HySChart operate on the same external input, the input is copied for both nodes.
- $n = l+^{\dagger}m$ (time extended disjoint sum), where l has g input ports and h output ports. Fig. 7 depicts the HDFG for $C(l+^{\dagger}m)$. Essentially $C(l)$ and $C(m)$ operate in parallel in it. Two selection nodes select the output of $C(l)$ or $C(m)$ as the output of the hierarchic node, depending on whether the

[8] Note that we encode the program state space $m\mathcal{S}$ as the tuple $\mathbb{N} \times \mathcal{S}$ in HDFG.

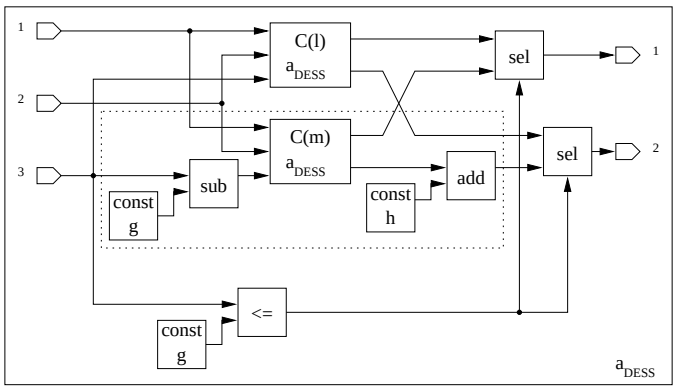

Fig. 7. HDFG for the disjoint sum.

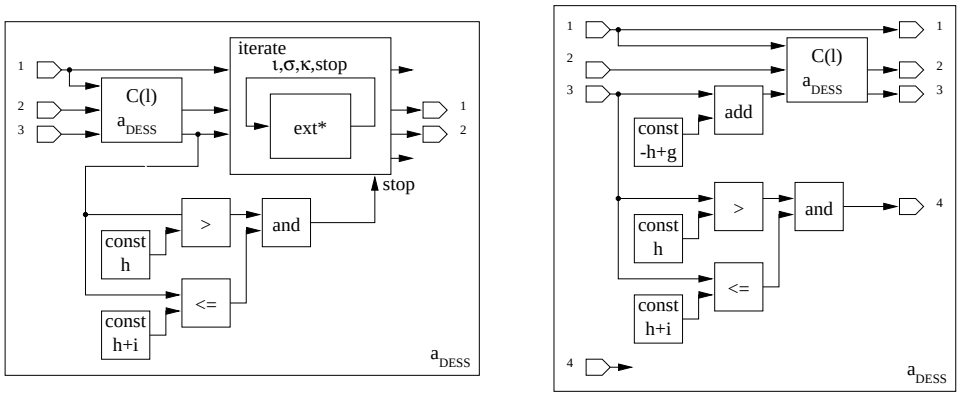

Fig. 8. HDFGs for additive feedback.

received control state κ (on input port 3) indicates that l or m is active. To understand the manipulation of κ in detail, the reader is referred to [5]. The principle is as follows: The name spaces for the input and output arrows of a node in a additive hierarchic graph are the natural numbers. The names always start with 1. When two hierarchic nodes l and m are composed, the name space of the second one must be shifted in order to still be able to refer to its arrows. This way input arrow i of m gets number $g + i$ in the composed node $l + m$, where g is the number of input arrows of l.[9] The dotted box in Fig. 7 indicates the parts of the HDFG responsible for shifting the name space and applying m.

- $n = l{\uparrow_+^i}^\dagger$ (i-ary time extended additive feedback), where l has $g + i$ input ports and $h + i$ output ports. The translation to HDFG $C(l{\uparrow_+^i}^\dagger)$ is given in Fig. 8, left and right. The node labeled *iterate* in Fig. 8, left, denotes iteration. It iterates *ext* until the condition, labeled with *stop* in the figure, is false. Hence, the node is similar to a while loop. By definition no time passes in the iteration. The graph for *ext* is given in Fig. 8, right. It basically consists of the translation of l together with a part that copies the input to enable the subsequent application of l on the same external

[9] This idea is similar to the DeBruijn notation for lambda terms.

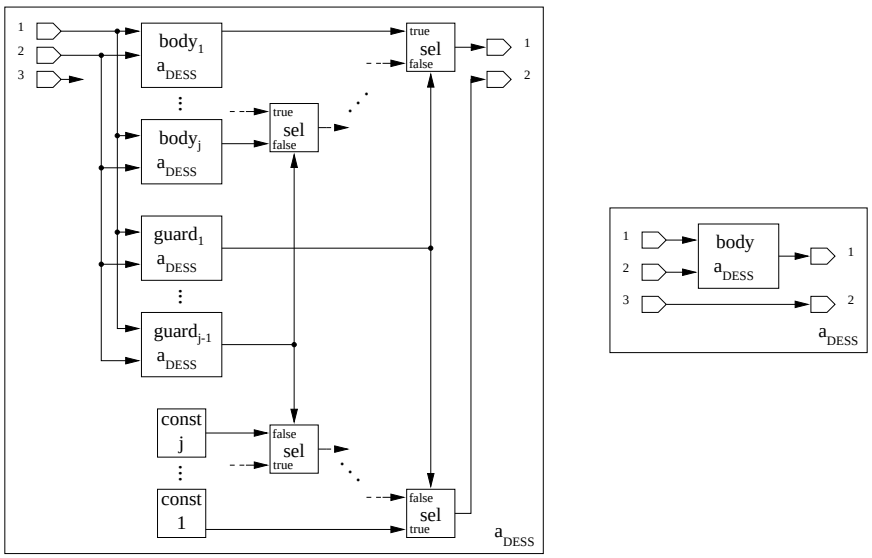

Fig. 9. HDFG for additive ramification with actions a_i and HDFG for single actions.

inputs, a subgraph which performs the name transformation from the output port numbers of l to its input port numbers and a further subgraph which computes the stop condition in each iteration. Note that the external input which is forwarded through the iterate node in Fig. 8, left, is not visible at the outside of $C(l{\uparrow_+^i}^\dagger)$. This is consistent with the type of ${l\uparrow_+^i}^\dagger$. The stop condition, on port 4 of ext, also is not visible outside the translation.

Informally, the translation can be explained as follows. If the feedback $l\uparrow_+^i$ receives control on one of its g input ports, then relation l is repeatedly applied until it gives control on one of the output ports from 1 to h. The distinction between input and output ports ensures that l is applied at least once. As the iteration node corresponds to a while loop, i.e. execution does not start if the condition is not true, this necessitates the first occurrence of $C(l)$ outside the iteration node.

- $n = \bullet\!\!<_j{}^\dagger;^\dagger(+_{i=1}^{\dagger j} a_i)$ (j-ary ramification together with actions a_i). Fig. 9, left, sketches the HDFG for $\bullet\!\!<_j{}^\dagger;^\dagger(+_{i=1}^{\dagger j} a_i)$, where $guard_i$ is the HDFG for the guard of action a_i and $body_i$ is the HDFG for its body. The control state κ is ignored in the graph, because ramification has only one input port and therefore expects $\kappa = 1$. For other values of κ the output of the ramification operator is irrelevant, as its output will be discarded at a higher hierarchic level of the additive graph in which $\bullet\!\!<$ appears. This is ensured by the type-correctness of an additive hierarchic graph.

  Due to requiring that the HySChart is deterministic, all action guards are disjoint. Furthermore, one of the actions after a ramification must always be enabled. If the ramification results from reducing the HySChart to a hierarchic graph, this is automatically ensured by the reduction. Otherwise, it is in the responsibility of the user. With ramification corresponding to the choice points in statecharts, this is similar to the usual requirement in

tools for statecharts which demands that one of the transitions after a choice point is enabled. Hence, as one action is always enabled, the guard of action a_j is not evaluated, since it must be true if all the others are wrong.

- Primitive nodes: For any primitive node, which is an action a as guaranteed by the reduction rules that generate the additive hierarchic graph from a HySChart, we define $C(a)$ as depicted in Fig. 9, right. An isolated action, not guarded by a ramification, can only occur if actions are sequentially composed, maybe via some intermediate connectors. It is in the responsibility of the user to ensure that the second action of such a sequential composition is always enabled if the first one was executed.[10] We assume that the user has taken care of such possible design faults and do not evaluate the action guard in the resulting HDFG.

The translation for time extended identity, identification and transposition is omitted due to space limitations. It follows the same principle as the translation of the disjoint sum and ramification.

Optimizations.

All HDFG nodes resulting from the translation are permanently active (activation rule a_{DESS}). However, for those HDFG nodes which are solely depending on and manipulating the control state (on input port 3), it suffices to be activated whenever their input changes. Hence, activation rule a_{DEVS} can be used for them. Furthermore, the action bodies in Fig. 9 only have to be active when their guard is true.

4.2.2 The Analog Part

The analog part *Ana* determines how the variables evolve as long as no discrete transition in the HySChart is possible and it updates the latched inputs. *Ana* is defined as the time extended disjoint sum of sequentially composed activities. The sequentially composed activities reflect the hierarchy of states in the HySChart, see [5] for an explanation how *Ana* is derived from the HySChart. *Ana* always is of the form $Ana = +^{\dagger}_{i=1}{}^{n}(upd;^{\dagger};^{\dagger}_{j=1}{}^{m_i}\hat{Act}_{i,j})$, where *upd* is the activity that updates the latched inputs. According to the definition in [5] the $Act_{i,j}$ receive continuous, piecewise smooth input functions and produce continuous, piecewise smooth output functions. The $\hat{}$ operator extends these activities to piecewise smooth functions by allowing discontinuities in the output whenever there is a discontinuity in the input function. Note that the operators $+^{\dagger}$ and $;^{\dagger}$ used in [5] were defined differently, such that $\hat{}$ was not needed. However, in the context of deterministic activities, which we require, the definition of $+^{\dagger}$ and $;^{\dagger}$ given in the preceding section is equivalent to that in [5]. The reason why we use this modified version is that it is not straightforward to specify activities without discontinuities and to add discontinuities

[10] An alternative to this strategy is to disallow that the second action has a guard different from true. This e.g. is done in many tools for statecharts.

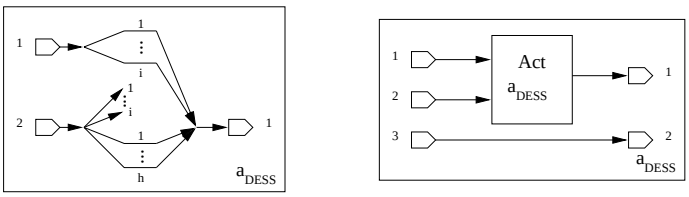

Fig. 10. HDFG for activity *upd* and extension of activities by the control state.

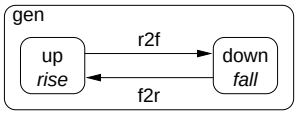

Fig. 11. HySChart for a saw tooth generator.

to them later on in HDFG. Nevertheless, activities $\hat{Act}$ with discontinuities can easily be specified. We assume that the individual activities $\hat{Act}_{i,j}$ are given as HDFG. The HDFG for activity *upd* is given in Fig. 10, left. Note that the first input port in this graph transmits external inputs, whereas the second one transmits the data state. In the graph the data state is split into the latched inputs (channel 1 to i) which are discarded and the remaining private and output variables (channels 1 to h). Then, the current external input values together with these remaining private and output variables are composed to a tuple again and form the new data state.

The activities extended with discontinuities $\hat{Act}$ which are specified by HDFG have type $\hat{Act} \subseteq (\mathcal{I} \times \mathcal{S} \times \mathcal{S})^{\mathbb{R}_+}$. As the disjoint sum of one summand $1\mathcal{S}$ is isomorphic to $\mathcal{S}$, this type allows to compose activities with $;^\dagger$ and $+^\dagger$. However, in the translation we must take care of our encoding of sums $m\mathcal{S}$ as $\mathbb{N} \times \mathcal{S}$. Hence, in the translation of primitive activities the HDFG for $\hat{Act}$ must be extended by a further channel for the control state. The resulting translation for primitive activities is depicted in Fig. 10, right. The only operators needed for the composition of activities are $;^\dagger$ and $+^\dagger$. Therefore, we can now apply the additive translation defined in the previous section to get the HDFG $C(Ana)$ for the analog part.

4.3 Example

As an example we regard the translation of a HySChart for a saw tooth generator. The HySChart is given in Fig. 11. It has no input variables (and therefore also no latched inputs), no private variables and only one real-valued output variable, called y. Hence, its data state space is $\mathcal{S} = \mathbb{R}$.

The action guard of $r2f$ and activity $rise$ are given by the HDFGs in Fig. 12. The first input port of both graphs, which by convention transmits the external input, is not connected, as the considered HySChart has no external input. The guard of $r2f$ corresponds to the predicate $y \geq ub$. The action body of $r2f$ is the identity on $\mathcal{S}$, i.e. the action does not modify y. $rise$ corresponds to the equation $y = \int pe\, dt$. To understand this, note that, due

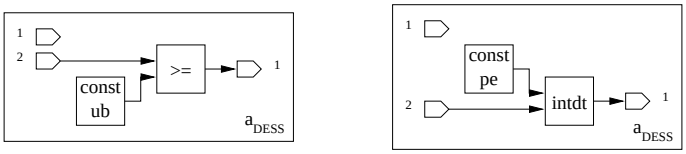

Fig. 12. HDFG for action $r2f$ and for activity $rise$.

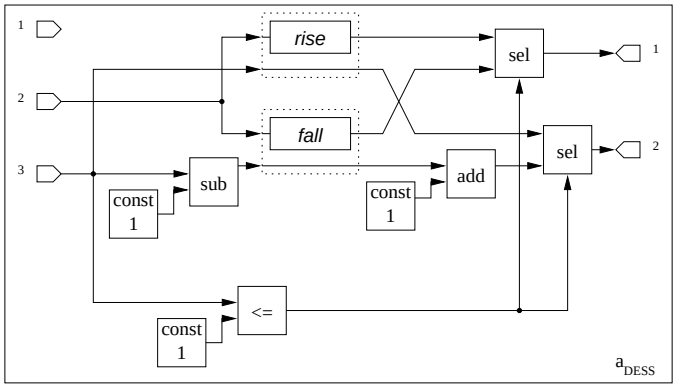

Fig. 13. HDFG for the analog part.

to the machine model, the input port of the HDFG for $rise$ will indirectly be connected to its output port via Lim_z and $Com^\dagger$ whenever the current control state corresponds to node up. As long as no action guard is true, the combinational part is the identity. Hence, $rise$ receives its output with an infinitesimal delay. The definition of the HDFG function $intdt$ provides that this results in $y = \int pe\, dt$. The action guard of $f2r$ and activity $fall$ are given by similar HDFG and denote $y \leq lb$ and $y = \int ne\, dt$, respectively. The body of $f2r$ is the identity. ub, lb, pe and ne are constants. The initial state of the HySChart is up with $y = 0$. In the derivation of the combinational part from the HySChart, node up gets number 1. Hence, the initial state is encoded as $z = (1, 0)$, where the second component of the tuple refers to the value of y.

As explained in [5] the HySChart is reduced to an additively interpreted hierarchic graph which defines its combinational part and to a second one which defines its analog part. According to the reduction, the combinational part is given by a hierarchic graph with 13 operators and 9 connectors. As each operator and connector results in a (sub-) HDFG in the translation, the HDFG resulting from the translation is fairly complex. Therefore, we do not give it here. Instead, we provide the translation of the analog part which follows the same principle, but is much easier.

The HySChart's analog part is defined by $Ana = rise{+}^\dagger fall$, where upd is omitted since we have no external inputs here. The HDFG for it is similar to that depicted in Fig. 7, where $C(l)$ is replaced by $C(rise)$, i.e. by the application of the rule for primitive activities to $rise$ (see Fig. 10, right), $C(m)$ is replaced by $C(fall)$ and the constants g and h both are 1. In the figure, $C(rise)$ and $C(fall)$ are indicated by dotted rectangles. $rise$ is selected for output if the current control state is 1 (node up of the HySChart) and $fall$

is selected if it is 2 (node *down* of the HySChart).

The HDFG for the whole HySChart is obtained by substituting $C(Com^{\dagger})$ and $C(Ana)$ in the translated computation model of Fig. 4, right, by the HDFGs resulting from the translation.

5 Conclusion

In this paper, we have presented a way to translate a statechart-like, formal visual description technique for hybrid systems, HyCharts, to a unified, graph-based representation of hybrid systems: HDFG. HDFG are an efficient, intermediate representation supporting the optimization of hybrid systems. The translation permits a through-going methodology for the rapid prototyping of embedded systems and closes a gap between abstract, visual description techniques such as HyCharts and rather simple and homogeneous intermediate representations such as HDFG.

In the future we want to evaluate the practical feasibility of our prototyping approach within a case study. Depending on the results, further optimizations may be considered. A formal correctness proof of the presented translation is supposed to be tackled when the case study has validated the utility of the approach. We expect that the proof can be performed using structural induction on multiplicative and additive hierarchic graphs.

References

[1] M. Broy. Requirements engineering for embedded systems. In *Proc. of FemSys'97*, 1997.

[2] S. Donnay, K. Swings, G. Gielen, W. Sansen, W. Kruiskamp, and D. Leenaerts. A Methodology for Analog Design Automation in Mixed-Signal ASICs. In *The European Design Automation Conference (EURO-DAC)*, pages 530–534, Paris, France, February 1994.

[3] C. Grimm. *Hybride Datenflussgraphen und ihre Anwendung beim Entwurf analog/digitaler Systeme*. PhD Thesis, J. W. Goethe-Universität (submitted), 1999.

[4] C. Grimm and K. Waldschmidt. Repartitioning and technology-mapping of electronic hybrid systems. In *Design, Automation and Test in Europe '98 (DATE)*, Paris, France, February 1998.

[5] R. Grosu and T. Stauner. Modular and visual specification of hybrid systems – an introduction to HyCharts. Technical Report TUM-I9801, Technische Universität München, 1998.

[6] R. Grosu, T. Stauner, and M. Broy. A modular visual model for hybrid systems. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, LNCS 1486. Springer-Verlag, 1998.

[7] R. Grosu, G. Ştefănescu, and M. Broy. Visual formalisms revisited. In *Proc. International Conference on Application of Concurrency to System Design (CSD'98)*, 1998.

[8] Integrated Systems Inc. Integrated Systems: Products - BetterState. `http://www.isi.com/Products/BetterState/`, 1999.

[9] A. Kalavade and E. A. Lee. A Hardware-Software Codesign Methodology for DSP Applications. *IEEE Design & Test of Computers*, pages 16–28, September 1993.

[10] I. Kanakis, C. Grimm, P. Oehler, and K. Waldschmidt. Rapid Prototyping of Mixed-Signal Systems with KANDIS. In *Int. Workshop on Logic and Architecture Synthesis (IWLAS)*, September 1997.

[11] P. Oehler, C. Grimm, and K. Waldschmidt. KANDIS - A Tool for Construction of Mixed Analog/Digital Systems. In *European Design Automation Conference*, Brighton, UK, September 1995.

[12] Rational Software Corporation. Unified modeling language, version 1.1. `http://www.rational.com/uml/resources/documentation/`, 1998.

[13] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons Ltd, Chichester, 1994.

[14] G. Ştefănescu. Reaction and control I. mixing additive and multiplicative network algebras. *Logic Journal of the IGPL*, 6(2):349–368, 1998.

[15] The MathWorks Inc. Stateflow. `http://www.mathworks.com/products/stateflow/`, 1999.