Hybrid Sequence Charts

Radu Grosu, Ingolf Krüger and Thomas Stauner*
Institut für Informatik, Technische Universität München
D-80290 München, Germany
Email: {grosu,kruegeri,stauner}@in.tum.de

Abstract

We introduce Hybrid Sequence Charts (HySCs) as a visual description technique for communication in hybrid system models. To that end, we adapt a subset of the well-known MSC syntax to the application domain of hybrid systems. The semantics of HySCs is different from standard MSC semantics. Most notably, we use a shared variables communication model and assume the existence of a continuous, global clock. Similar to their classic counterpart HySCs can be advantageously used in the early phases of the software development process. In particular, in the requirements capture phase, they improve the dialog between customers and application experts. They complement existing formalisms like hybrid automata by focusing on the interaction between the system's components. We outline the key concepts and the usage of HySCs along an example, the specification of an electronic height control system. Then we define the formal semantics of their basic elements.

1. Introduction

Designing software for embedded systems usually requires to take the characteristics of the system's environment into consideration, e.g. in order to derive timing requirements. Often the environment is, to a large extent, determined by continuous processes. Sometimes the system itself also exhibits discrete *and* continuous behavior. The description of the embedded system together with its environment therefore necessitates *hybrid* description techniques, i.e. techniques which are adequate for mixed discrete and continuous systems. Hybrid systems generalize real time systems by considering further physical quantities apart from time.

In recent years a considerable number of description techniques has been developed for the specification of hybrid systems. Some of them are based on Petri nets [17], others use logic [11] and yet others are based on some kind of automata [1, 12, 9]. However, little work has been done to visualize the behavior of a hybrid system together with the communication between its components. Yet, a thorough integration of interaction-based and state-based description techniques is essential if we wish to support and improve today's development processes for hybrid and, more generally, embedded systems.

We regard a hybrid system as consisting of a set of time-synchronously operating components, each encapsulating a private state and communicating with the other components over directed channels. The behavior of a component is characterized, as intuitively shown in Fig. 1, top left, by periods where the values on the channels change smoothly and by time instants at which there are discontinuities. In our approach the discontinuities are caused by discrete actions. The smooth periods are caused by analog activities. Two attempts at visualizing the evolution of the values of a hybrid system's channel- and private variables are trajectories and timing diagrams. Their deficiencies motivate our introduction of Hybrid Sequence Charts, below.

TRAJECTORIES. *Trajectories* are a straightforward visualization approach that directly depicts the evolution of a system's variables over time (Fig. 1, top left). While this approach is simple and effective it can only depict one special case, namely the one in which all variables evolve as in the diagram. It cannot highlight qualitative differences between system states. Visualization by trajectories is supported by development tools like MATLAB [16].

TIMING DIAGRAMS. A first step from single trajectories to an abstract description of sets of trajectories is obtained by partitioning for each variable the time period under consideration into qualitatively equivalent intervals and by only giving a predicate specifying the variable's evolution within the respective interval. In the diagram of Fig. 1, bottom left, for example, it is only important to know whether variable $fHeight$ is inside or outside a given tolerance interval. Therefore, the concrete trajectory $fHeight(t)$

*This work was supported with funds of the Deutsche Forschungsgemeinschaft under the Leibniz program within project SysLab, and under reference number Br 887/9 within the priority program *Design and design methodology of embedded systems*.

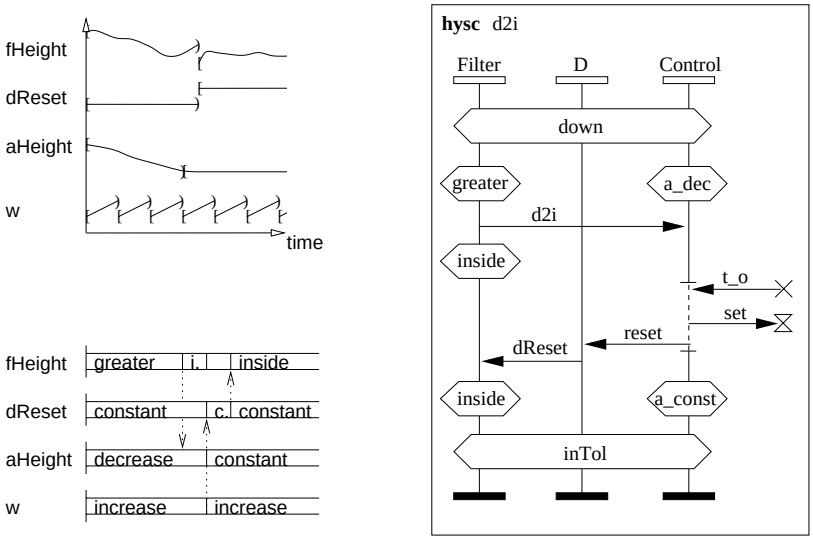

Figure 1. Description techniques for the behavior of hybrid systems.

from Fig. 1, top left, can be abstracted to the sequence of intervals with the predicates $greater$, meaning that *fHeight* is outside the tolerance interval, $inside$, which is abbreviated by $i.$ in the figure and means that *fHeight* is inside the tolerance interval, the unlabeled interval, meaning that the value of *fHeight* is arbitrary, and $inside$ again.[1] Note that the resulting diagram has some similarity with timing diagrams [3], which are widely used in hardware design, and the *constraint diagrams* introduced in [6]. Causality can be indicated in the diagram by drawing vertical arrows between the abstract time axes of two variables if a change in the first variable is relevant, i.e. may provoke a qualitative change, for the evolution of the second one.

HYBRID SEQUENCE CHARTS. In this paper we go a step further and also abstract from the individual variables in the graphical representation of system behavior. Thus, instead of partitioning and giving predicates for individual variables, we project the trajectories of all variables of one system component on a single abstract time axis. One axis for each component is appropriate, because we are interested in the sequence of qualitative states that each component traverses. Such a qualitative state of a component is usually characterized by a predicate over all its variables (see Fig. 1, right). This projection was motivated by notations for component interaction that have gained increasing popularity in the domain of telecommunication systems (cf. [10]), and, more generally, in object-orientation (cf. [13, 5, 4]). We are aware, of course, that the semantic models – if existent – of such notations do not necessarily match the time-synchronous hybrid system model with communication proceeding over shared channels that we have sketched above. Yet, we believe that by adapting notation from, say, MSCs (cf. [10]) to the application domain we consider here, we can carry over much of the intuition that has contributed significantly to the popularity of sequence charts in general. In fact, we consider capturing interaction sequences among system components an important step of *any* development process. Therefore, we borrow a subset of the syntax of MSC-96 (cf. [10]) for the specification of interaction sequences within hybrid systems[2]; we call the resulting notation "Hybrid Sequence Charts (HySCs)". In particular, we use arrows to denote events; arrows are directed from the originator of the event to its destination. Angular boxes denote conditions on the component's variables; they may span a single instance axis (local conditions), or multiple axes (non-local condition), and even all component axes (global condition). The remaining syntactic elements in Fig. 1, right, are introduced later. Every HySC specifies a typical evolution, or *scenario*, of the system under consideration in connection with its environment over some finite time interval. If the environment does not behave as depicted in the HySC, no statement is made about the system's evolution. By composing such typical evolutions appropriately, we can achieve a specification of the system's behavior upon different inputs from the environment. Even a complete specification covering all possible inputs is possible. We use *High-level HySCs* (HHSCs), whose syntax we also borrow in part from MSC-96, to specify the composition of HySCs. To make HHSCs applicable in the context of hybrid systems we provide notation for

[1] Label $c.$ is used as abbreviation for $constant$ in the figure.

[2] This has the further advantage that developers can use standard syntax-directed graphic editors for their specifications.

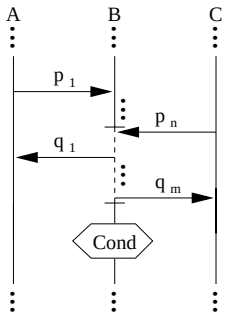

Figure 2. Basic segment of a HySC.

expressing preemption, which is an important concept for embedded systems.

HySCs IN THE DEVELOPMENT OF HYBRID SYSTEMS. Just as MSCs [10] or sequence diagrams [13] in the discrete case, HySCs can be used for requirements specification, interface specification, test-case specification, validation, and documentation. Due to their intuitive appearance they are particularly well-suited for capturing and specifying system requirements in the dialog among engineers from different disciplines, as well as among engineers and customers.

OVERVIEW. The rest of this paper is organized as follows. In Section 2 we introduce HySCs informally and explain our understanding of them. In Section 3 we present an example hybrid system; in particular, we discuss the key parts of its formal specification with HySCs in Section 3.2. Section 4 contains the formal semantics of the basic elements of HySCs. We summarize our work, and draw conclusions in Section 5.

2. Hybrid sequence charts – HySCs

We start with a short introduction to the syntax and informal semantics of basic HySCs that consist of interactions, conditions, and coregions only. Then we cover HHSCs, which allow us to specify hierarchic "roadmaps" through sets of HySCs.

Basic HySCs. Basic HySCs contain one vertical axis, an abstract time axis, for each component, or *instance*, under consideration. Time advances from top to bottom. Sequences of incoming and outgoing arrows partition the time axis of each component into intervals. According to our view of hybrid systems, which we have sketched in Section 1, we require the existence of a global clock, and assume that communication occurs without delay. We assume further that the *components* occurring in the HySC are connected by *channels* along which message exchange occurs. Hence, a HySC is built up from sequences of segments of the form given in Fig. 2. Each such segment denotes the execution of an *action* by component B. The action is triggered by the occurrence of all events p_1 through p_n; we say that the *action guard* becomes true. The result of executing the *action body* is that B simultaneously emits the events q_1 through q_m, and changes its state to the one specified in the condition labeled $Cond$ in Fig. 2. Actions in hybrid systems usually depend on the values of continuous variables; therefore, we consider action guards and action bodies carefully, below.

Before we regard the actions in detail, it is necessary to explain our classification of variables. In our view each component has a set of *input* variables, which are written by the environment or by other components and a set of *controlled* variables that are written by the component itself. The set of controlled variables of a component is further partitioned into a set of *private* variables, whose elements are only visible to the component, and a set of *output* variables, whose elements may be read by the other components or the environment. The input and the output variables are the *observable* variables.

The *action guard* $p_1 \wedge \ldots \wedge p_n$ is a conjunction of predicates p_i. Each predicate p_i that labels an arrow from a component, say A, to B may depend on the old and current values of the output variables of A that are input by B and optionally on the old values of some other private variables of B[3]. The arrow indicates the moment of time (the event) when p_i becomes true. A similar arrow must be drawn if p_i becomes false again, before the action is executed. However, no second arrow needs to be drawn if the predicate possibly only holds for a single point in time, i.e. if the predicate depends on the occurrence of an event or on the exact value of a continuous variable.

The *action body* $q_1 \wedge \ldots \wedge q_m$ is also a conjunction of predicates q_i. Each predicate q_i that labels an arrow from B to, say, A specifies the current values for the output variables of B that are input by A. These values may depend on the current value of all input variables and on the old and current value of all controlled variables of sender B.

As soon as all conjuncts of the action guard are true, the action body is executed. All the changes that it causes on the output variables simultaneously become visible to those other components which read these variables. Simultaneity is expressed graphically by a *coregion*, i.e. by drawing a region of the time axis of one component as a dashed line; all the predicates in this coregion are evaluated simultaneously (see Fig. 2).

We allow the use of predicates as condition labels to indicate a component's state, and adopt the convention that no new condition symbol is drawn if the control-state does not change. Conditions ranging over a set of components are also allowed, and express a global state of the referenced

[3] Actually, the old values of the output variables of A that are input by B are kept in private variables of B.

components. A local as well as such a *hierarchic* condition $Cond$ remains valid up to the next condition symbol that references the same or a superset of the components referenced by $Cond$.

Events can be expressed in terms of (event) predicates by toggling boolean variables. For example, we write $e?!$ for $e' = \neg e$ meaning that the current value of e (denoted by e') is the negation of the old value (denoted by e) [2, 9]. The old value of a variable e at a time t is defined as the limit from the left $lim_{u \nearrow t} e(u)$ for this variable, i.e. as the value just before t.

Note that an arrow from A to B can in general be labeled with the conjunction of a part of an action body q_i of A and a part of an action guard p_j of a different action of B. This may be the case if the current values specified for the output from A to B are relevant for p_j.

A qualitative state in a hybrid system is characterized by a set of trajectories that are allowed for the variables in that state. Therefore, the condition after an action in a HySC not only determines the next qualitative state, but it also specifies how input and controlled variables of the component are expected to evolve in this qualitative state. Controlled variables may only evolve continuously, because in our view discontinuities may only be caused by qualitative changes, which in turn result from actions.

HySCs can also be used to specify timing requirements like "at least time t_s passes between the arrows a and b", as proposed in [14] for timed MSCs. Basically, this is achieved by local variables which evolve in pace with global time and which measure durations. For instance, a timeout can be specified by using a private variable, which evolves in pace with global time, and an action guard that becomes true when the variable has reached a certain threshold. Setting the variable to a certain value corresponds to resetting the timer. We therefore use the set-timer and timeout symbols borrowed from MSC-96 to denote this (see Section 3.2).

HIGH-LEVEL HySCs (HHSCs). HySCs can be used within HHSCs to specify the *complete* behavior of a system. For this complete behavior description HHSCs provide operators for the concatenation of HySCs, the choice between HySCs and the iteration of HySCs. The choice is controlled by *global* conditions, i.e. by conditions ranging over all components. A branch of a choice in the HHSC may be taken iff the condition guarding it is currently true. The system behavior is then determined by the HySC following the branch operator. It must start with the same condition as the selected branch. Syntactically, the starting point in an HHSC is represented by an outlined, downward triangle, an end-point (if it exists) by a filled, upward rectangle. References to other HySCs appear in rounded boxes. Conditions are depicted as in basic HySCs. Lines (or arrows) determine the "road-map", i.e. the sequence in which the interactions appearing in the referenced HySCs may occur (see Section 3.2 for examples).

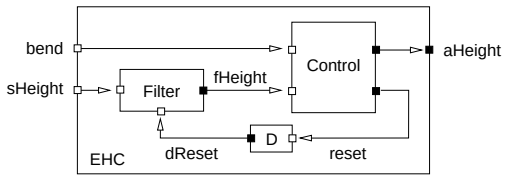

Figure 3. Architecture of the EHC.

In this paper we introduce the additional concept of preemption to HySCs, which is not supported by the popular notations for component interaction, like [10] or [13]. Graphically preemption is depicted as a labeled, dashed arrow between two HySC references in an HHSC. Its meaning is that the system behavior is as determined by the HySC reference that is the arrow's source, as long as the *preemptive predicate*, to which the arrow's label refers, is false. As soon as the predicate becomes true, the system behavior is as specified by the HySC reference to which the arrow is pointing. Preemption is widely used in the programming of embedded systems. We believe that this is a highly important concept.

3. HySCs in practice

3.1. An electronic height control system

To explain the capabilities and usage of HySCs, we formally specify an electronic height control system (EHC), taken from a former case study carried out together with BMW, and discuss the key parts of this specification. The purpose of the EHC 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 [15]. It basically works as follows: whenever the chassis level $sHeight$ 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 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 $bend$ informs the controller whether the car is going through a curve. Periodical sampling of $fHeight$ occurs in dependence of a timer, which is local to the controller. Besides the environment, the basic components of the system are the filter and the controller (see Fig. 3). The escape valve and the compressor are modeled within the controller. The component labeled D introduces a delay and ensures that the feedback between the filter and the controller is well-defined. A specification of the EHC with HyCharts, a state-based description technique for hybrid systems, can be found in [9].

3.2. Specification with HySCs

We specify behavior required by the EHC by using HySCs. First, we present HHSCs for the top-level requirements. Then, we consider one of the basic HySCs in detail.

3.2.1. High-level HySCs (HHSCs). The top-level description of the EHC is given by an HHSC, as shown in Fig. 4, left. On this abstraction level, we distinguish between two scenarios: the car is either inside a curve or going straight. The behavior inside a curve is characterized by the HySC `inBend`. The behavior outside a curve is characterized by the HySC `outBend`.

PREEMPTION. The EHC switches between these two behaviors each time the boolean value provided by the variable $bend$, which is controlled by the environment, is toggled. In other words, toggling $bend$ is a *preemption* event. To describe this situation we use the preemption mechanism outlined in Section 2. Recall that we use a special kind of arrows, *preemption arrows*, to denote preemption in HHSCs, which is represented visually by a *dashed arrow* connecting a source HySC reference to a destination HySC reference, and labeled by the *preemptive predicate*. Intuitively, any prefix of the traces described by the source HySC reference may be followed by a time instant at which the *preemptive predicate* is true and then by a trace of the destination HySC reference. The labels `inBend` and `outBend` in the rounded HySC boxes refer to further HySCs. The labels `inBendC` and `outBendC` in the angular condition boxes refer to the condition predicates $bend = True$ and $bend = False$, where variable $bend$ signals whether the car is in a curve. The labels `b2n` and `n2b` both stand for the event predicate $\texttt{b2n} \equiv \texttt{n2b} \equiv bend?!$, i.e. for the occurrence of an event which toggles the value of $bend$.

(NONDETERMINISTIC) CHOICE. As long as the car is outside a curve the behavior of the EHC is descibed by HHSC `outBend` (Fig. 4, right). On this level we use the nondeterministic choice operator, graphically depicted as branching arrows, to distinguish between two cases. In the first case, the compressor and the escape valve are off, because the value of $fHeight$, which was read last, was inside the tolerance interval. A further choice operator splits this case into two sub-cases: If $fHeight$ remains inside the interval, the behavior is given by the HySC `i2i`. If the chassis level gets outside the interval, then we have a behavior as described by the HySC `i2o`. The second case describes the behavior if compressor or escape valve are on, because of the last value of $fHeight$ being outside the tolerance interval. This part of the HySC is symmetric to the first one.

The labels `inTol` and `outTol` in the HySC refer to the predicates $\frac{d}{dt}aHeight = 0$ and $\frac{d}{dt}aHeight \neq 0$, respectively, which characterize global states of the system. Variable $aHeight$ (actuator height) models how the chassis level is influenced by the compressor and the escape valve. If the derivative of $aHeight$ is zero, i.e. $aHeight$ remains constant, then the chassis level is not modified by the compressor or the escape valve.

FEEDBACK. After the behavior specified by the HySCs `i2i`, `i2o`, `o2i` and `o2o` is finished, a new cycle starts in which we again have to distinguish the cases from above. This is modeled by the *feedback arrows* in the HySC leading from the bottom of it up to those points in the HySC from where the following behavior must continue. Thus, feedback allows us to specify infinite behavior.

FINITE BEHAVIOR. The HHSCs `i2o` and `o2i` are examples for HySCs that do not specify infinite behavior. Instead of feedback arrows, an arrow leading to a black triangle is drawn in them to mark their end. As they are fairly straightforward, we omit them in this paper and refer the reader to [7].

3.2.2. Basic HySCs. All the basic HySCs referenced directly or indirectly by HHSC `outBend` describe the behavior of the EHC in the interval between two expirations of the Controller's timer. In the following we will analyze HySC `i2d` in detail. This HySC describes the scenario in which the chassis level increases from within the tolerance interval to a value above the upper bound (Fig. 5, left). It is referenced by HHSC `i2o`.

CONDITION PREDICATES. HySC `i2d` starts with the condition box labeled `inTol`. As mentioned in the previous section this label refers to predicate $\frac{d}{dt}aHeight = 0$. Because the condition box ranges over all components of the diagram it is a global condition. The following conditions `inside` and `a_const` range over only one component. Hence, they are local conditions. They add some more detail on the evolution of the variables. Label `inside` refers to predicate $fHeight \in [lb, ub]$, where lb and ub are constants denoting the lower and upper bound of the tolerance interval. Label `a_const` stands for $\frac{d}{dt}aHeight = 0 \ \wedge \ w \leq w_s \ \wedge \ \frac{d}{dt}w = 1$. The first conjunct of this condition means that the chassis level is not modified by $aHeight$, the second conjunct means that variable w is less than constant w_s, the sampling period, and the third conjunct provides that w evolves in pace with the global time, i.e. it is a clock variable or a timer. No local predicate is given for component D. By convention this means that it implicitly has local predicate $True$.

EVENTS. The very moment $fHeight$ reaches the upper bound of the tolerance interval is given by the horizontal arrow labeled by `abv`, which stands for event predicate $fHeight \geq ub$. After the event `abv` has occurred, the chassis level is above the tolerance interval. Again, this property (or interval invariant) is given by a local condition predicate, the condition predicate `greater`, which stands for $fHeight \geq ub$.

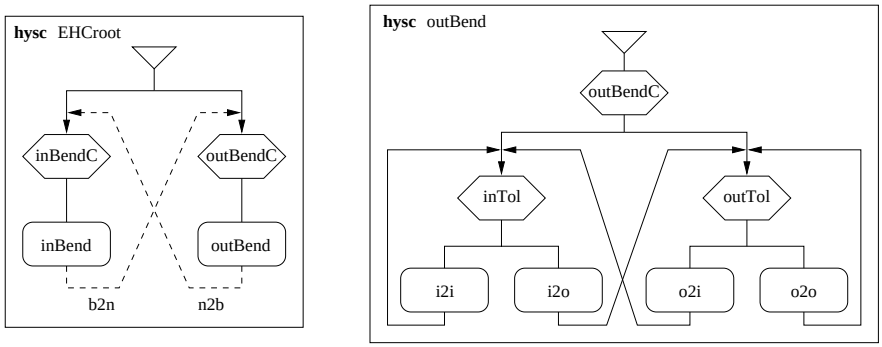

Figure 4. The HySCs `EHCroot` and `outBend`.

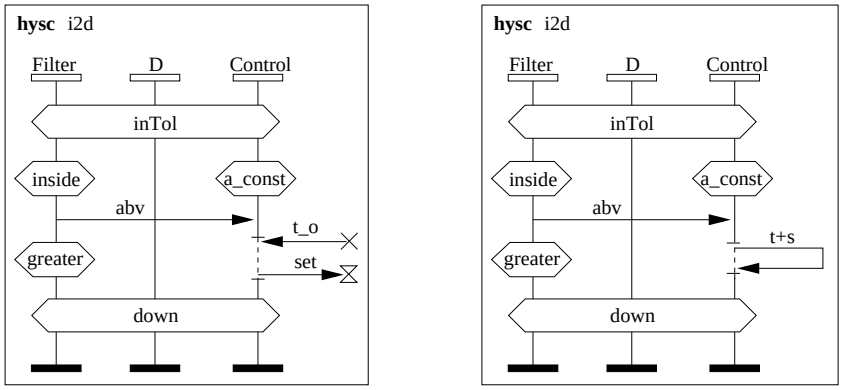

Figure 5. The HySC `i2d` and its reduction without timeout arrows.

TIMERS. The control component senses that the chassis level is too low, only when the timer has expired, i.e., with some delay. As a consequence, neither the escape valve, nor the compressor are actuated before the expiration. Correspondingly, the local condition `a_const` continues to hold for the controller.

In the diagram we draw the timeout and set-timer arrows `t_o` and `set` borrowed from MSC-96 to represent an event the control component sends to itself. Predicate `t_o` stands for $w = w_s$, i.e. the timer has reached the threshold, and `set` stands for $w' = 0$ which starts a new sampling period by resetting the timer. On the level of semantics these arrows can be reduced to a single arrow labeled `t+s` pointing from the axis of the control component to itself (see Fig. 5, right). The label refers to event predicate $w = w_s \wedge w' = 0$.

SCOPING OF CONDITIONS. As mentioned previously, conditions remain valid until the next condition on the same or on a higher level of hierarchy is given. Thus, before the timer has expired, the overall behavior of the EHC still has to satisfy the global condition `inTol`, because no other global condition occurred up to that point. Correspondingly, the set of behaviors characterized by the conjunction of the predicates `inside` $\wedge$ `a_const` and by `greater` $\wedge$ `a_const` is a subset of the behaviors characterized by `inTol`.

4. Semantics of HySCs

Suppose we are given a set of HySCs with the components (or instances) $C_1, \ldots, C_n$. For each component C_i, we assume its interface, i.e. the set of input and controlled variables, to be given. In the following let S_i be the data space associated with the controlled variables of component C_i. For uniformity, let S_0 be the data space associated with the variables controlled by the environment and $S = S_0 \times \ldots \times S_n$. Then we define the semantics of a HySC M to be a set $[\![M]\!] \subseteq S^{\mathbb{R}_+} \times \mathbb{R}_+^\infty$ of pairs (φ, t) where $\varphi \in \mathbb{R}_+ \to S$ is a *piecewise smooth* function (also called *a dense communication history* or *dense stream*) that exhibits the behavior required by M inside the time interval $[0, t]$. If $t = \infty$ then the behavior of φ is constrained by M along the whole time axis, i.e., the HySC M never terminates.

$\mathbb{R}_+^\infty$ is defined as the set of the nonnegative real numbers, $\mathbb{R}_+$, plus the special element ∞. We say that a function $f \in \mathbb{R}_+ \to Q$ is piecewise smooth iff every finite interval on $\mathbb{R}_+$ can be partitioned into *finitely* many left closed

and right open intervals such that on each such interval f is infinitely differentiable for $Q = \mathbb{R}$ or f is constant for $Q \neq \mathbb{R}$. Infinite differentiability allows us to assume that all differentials of f are well-defined. A tuple of functions is infinitely smooth iff all its components are. For a possibly infinite interval $A \subset \mathbb{R}_+$ we write Q^A to denote the set of functions from A to the set Q that are piecewise smooth on A.

With writing $V = S_1 \times \ldots \times S_n$ for the data space of the controlled variables, we can also interpret the semantics of a HySC $[\![M]\!]$ as a relation between the dense histories of the input variables, the dense histories of the controlled variables and the considered time intervals, i.e., $[\![M]\!] \subseteq S_0^{\mathbb{R}_+} \times V^{\mathbb{R}_+} \times \mathbb{R}_+^\infty$. We do not demand that this relation is total in the set of input streams $S_0^{\mathbb{R}_+}$. In fact HySCs may constrain the evolution of the input variables. This takes into account that a single HySC describes a system's response to a particular input from the environment.

ZENONESS. Specifications which demand that a system performs infinitely many discrete moves within a finite interval are called *zeno*. Like with other powerful description techniques for hybrid systems, such as hybrid automata [1], it is possible to write down zeno specifications with HySCs. For instance, zenoness can result from specifying that the system always reacts discretely when a continuous input signal crosses a boundary value. In a high-level specification technique we do not want to exclude such specifications which certainly make sense for many input signals. Hence, zeno behavior has to be ruled out later in the design process.

4.1. Predicates

CONDITION PREDICATES. The condition predicate p that holds in a certain section of the abstract time axes of all the components in a HySC can be derived as the conjunction of all, local, and hierarchic condition predicates that are valid in this section. The derived condition ranges over all the components. Its semantics is a relation $[\![p]\!] \subset \bigcup_{A \in Itv} S_0^A \times V^{A_c}$, where Itv is the set of possibly infinite right-open intervals starting from zero, and, for a set X, the notation X^{A_c} denotes the set of piecewise smooth functions X^A which furthermore are continuous, hence $X^{A_c} \subset X^A$. This type of the predicates' semantics permits discontinuities in the input, while the controlled variables must still evolve continuously. This reflects that discrete jumps in the evolution of the controlled variables are interpreted as events, hence they are only allowed when an event arrow is drawn in the HySC. Furthermore, the type allows that a condition predicate specifies finite behavior of varying length.

EVENT PREDICATES. The semantics of the event predicates e which label the arrows is a relation between the old and the new values of the variables $[\![e]\!] \subset S \times S$, where we demand that $[\![e]\!]$ is topologically closed (see Section 4.2 for the justification of this restriction). The semantics of simultaneous events, which are graphically denoted by arrows emanating from or pointing to a dashed region of the abstract time axis of a component in a HySC, is defined as the conjunction of the individual predicates of all the simultaneous events within the dashed region under consideration. Those variables for which the event predicates do not specify new values remain constant.

4.2. Basic HySCs

The basic idea behind the semantics of a HySC M is that it defines a set $[\![M]\!]$ of tuples such that for each $(\varphi, t) \in [\![M]\!]$ the dense history φ behaves inside the time interval $[0, t]$ as required by M and arbitrarily outside of $[0, t]$. In the definition of $[\![M]\!]$ it is useful to generalize the lower bound 0 to an arbitrary value $u \in \mathbb{R}_+$ and to work with sets $[\![M]\!]_u$ where the dense histories φ are constrained inside the time interval $[u, t]$. In the following we define $[\![M]\!]_u$ inductively on the structure of M. Then obviously the semantics of a HySC M is $[\![M]\!] \stackrel{\text{def}}{=} [\![M]\!]_0$.

NEUTRAL HySC. HySCs without events act as the neutral elements with respect to our semantics. All the conditions in them are ignored, and no time elapses in them: $[\![M]\!]_u \stackrel{\text{def}}{=} \{(\varphi, u) \mid \varphi \in S^{\mathbb{R}_+}\}$

SINGLE EVENT HySC. Suppose p is the condition predicate that results from the conjunction of all the condition predicates that are valid in the section of the HySC before event e happens. Note that e may be the conjunction of a set of simultaneous events. $[\![M]\!]_u$ is defined as follows:

$$\begin{aligned}[\![M]\!]_u \stackrel{\text{def}}{=} \{(\varphi, t) \in S^{\mathbb{R}_+} \times \mathbb{R}_+ \mid \\ t = min\{v > u \mid (lim_{x \nearrow v} \varphi(x), \varphi(v)) \in [\![e]\!]\} \\ \wedge\ \varphi_u{\downarrow_{[0,t-u)}} \in [\![p]\!]{\downarrow_{[0,t-u)}}\}\end{aligned}$$

where $min\,\emptyset \stackrel{\text{def}}{=} \infty$, $\varphi_u(x) \stackrel{\text{def}}{=} \varphi(u + x)$ and $\varphi{\downarrow_\delta}$ denotes the restriction of a dense stream to the time interval δ. Restriction is extended to tuples of dense streams in a componentwise style. To constrain φ inside $[u, t]$ without violating the time's origin assumption we constrain the translation φ_u of φ by the condition predicate p inside the interval $[0, t - u)$.

The definition requires that a finite, non-zero amount of time passes before the event becomes true. The HySC then terminates at the first time instant t at which e is true. Provided e does not hold initially, this first time instant, defined as the minimum of a set, is guaranteed to exist, because $[\![e]\!]$ is topologically closed. (See [8] for a proof under similar assumptions.) Demanding that some time passes before the event occurs is motivated by the visual representation. If we

wanted to specify that no time passes between two consecutive events, we would have to use simultaneous events.

Note that $\infty \notin \mathbb{R}_+$ and therefore if $t = \infty$ then $[\![M]\!] = \emptyset$. Thus, the semantics requires that the event eventually occurs, which is also motivated by the visual representation. The event arrow in the diagram would be misleading, if we allowed it to never occur.

SEQUENTIAL COMPOSITION. The sequential composition of the HySCs M_1 and M_2, textually denoted as $M_1; M_2$, is syntactically well formed only if M_1 ends with the global condition with which M_2 starts. In particular, this includes the case that M_1 and M_2 are successive parts of a single, larger HySC. The semantics is given only for well formed terms.

$$[\![M_1; M_2]\!]_u \stackrel{\text{def}}{=} \{(\varphi, t) \in S^{\mathbb{R}+} \times \mathbb{R}_+^\infty \mid \exists v \in \mathbb{R}_+ \,.\, (\varphi, v) \in [\![M_1]\!]_u \ \wedge \ (\varphi, t) \in [\![M_2]\!]_v\}$$

Note that whereas the HySC $M_1; M_2$ may describe an infinite computation ($t \in \mathbb{R}_+^\infty$) any of its prefixes exhibiting the behavior required by M_1 has to be finite ($v \in \mathbb{R}_+$).

4.3. HHSCs

Due to space limitations the reader is referred to [7] for the detailed semantics of nondeterministic choice, feedback and (nested) preemption. Basically, the semantics of feedback is obtained by a fixed point construction. The definition for preemption is rather technical and in large parts similar to the semantics definition for single event HySCs and sequential composition. The semantics of nondeterministic choice is fairly straightforward.

5. Conclusion

Borrowing from the standardized syntax of MSC-96, we have introduced a description technique that allows the system developer to specify the communication between the components of a hybrid system graphically. Basically, this is achieved by giving precise meaning to the conditions and events in HySCs. Motivated by the specific needs of embedded systems we have, furthermore, included a construct into our definition of HHSCs that allows us to specify preemption. We demonstrated the usage of HySCs along a non-trivial example and defined their formal semantics. HySCs are more abstract than drawing trajectories of the system variables, and are more detailed than other forms of graphical interaction specifications that do not handle continuous variables, e.g. [10, 13]. Thus we believe they are a good supplement to state-based hybrid techniques like hybrid automata or HyCharts [1, 9], just as ordinary sequence diagrams are beneficial in the development of discrete systems. In particular, they seem to be well-suited for bridging the gaps between requirements capture, specification, and later phases of system development. Note that, apart from their syntax, HySCs are substantially different from standard MSCs.

ACKNOWLEDGMENT. We thank Manfred Broy, Jan Philipps and Olaf Müller for their constructive criticism after reading a draft version of this paper.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. 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. Henzinger. Reactive modules. In *Proc. of the 11th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[3] T. Amon, G. Borriello, T. Hu, and J. Liu. Symbolic timing verification of timing diagrams using presburger formulas. In *Proc. of the 34th Design Automation Conference*. ACM, 1997.

[4] M. Broy, C. Hofmann, I. Krüger, and M. Schmidt. A graphical description technique for communication in software architectures. Technical Report TUM-I9705, Technische Universität München, 1997.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns. Pattern-Oriented Software Architecture*. Wiley, 1996.

[6] C. Dietz. Graphical formalization of real-time requirements. In *Proc. FTRTFT'96*, LNCS 1135. Springer Verlag, 1996.

[7] R. Grosu, I. Krüger, and T. Stauner. Hybrid sequence charts. Technical Report TUM-I9914, Technische Universität München, 1999.

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

[9] R. Grosu, T. Stauner, and M. Broy. A modular visual model for hybrid systems. In *Proc. FTRTFT'98*, LNCS 1486. Springer-Verlag, 1998.

[10] ITU-TS. Recommendation Z.120 : Message Sequence Chart (MSC). Geneva, 1996.

[11] L. Lamport. Hybrid systems in TLA+. In *Hybrid Systems*, LNCS 736. Springer-Verlag, 1993.

[12] N. Lynch, R. Segala, F. Vaandrager, and H. Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[13] Unified modeling language, version 1.1. Rational Software Corporation, 1997.

[14] I. Schieferdecker. Proposal for time and performance in MSCs. In *Proc. ITU-T Meeting SG10*, Geneva, 1998.

[15] T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In *Proc. HART'97*, LNCS 1201. Springer-Verlag, 1997.

[16] The MathWorks Inc. MATLAB. `http://www.mathworks.com/products/matlab/`, 1999.

[17] R. Wieting. Hybrid high-level nets. In *Proc. of the 1996 Winter Simulation Conference, Coronado, California*, pages 848–855, 1996.