Systematic Development of Hybrid Systems[*]

Thomas Stauner

BMW Car IT, Petuelring 116, D-80809 München, Germany
Email: thomas.stauner@BMW-CarIT.de, Phone: +49 89 189 311-51

This extended abstract gives an overview over the results achieved in the author's PhD thesis on the systematic development of hybrid embedded systems. An idealized development process for hybrid, i.e. mixed discrete and continuous, embedded systems is presented and used to point out where the results of the thesis support the development of hybrid systems. Besides the proposal of the development process itself, the thesis' results fall into three main categories. First, graphical notations with a precise mathematical semantics supporting an integrated view on hybrid systems are developed. Second, methods for the stepwise refinement of hybrid systems described with the proposed notations are elaborated. This in particular includes methods for moving from an abstract continuous time scale to an implementation oriented discrete time scale in a way which preserves essential system properties. Third, properties of hybrid systems are formalized, classified and put in relation to known classes of properties in computer science. This also uncovers and formalizes parallels between proof methods known in control theory and methods from computer science. Thereby the thesis fosters a deeper understanding of hybrid systems by computer scientists.

1 Introduction

Today more and more computing elements are used in technical plants and in all kinds of devices. Basic reasons for this tendency are the declining price and size of computing elements accompanied by the possibility of realizing new functionality and by the flexibility software-based solutions offer. In all such *embedded systems* software plays a central role in achieving the functionality of a product. Hence, the development of the software plays a key role in the overall development of the product. A well-founded methodology based on formal methods can assist the systematic development of embedded systems and help to reduce the risk of errors. This is vital for embedded systems as they typically target the mass market or perform safety-critical functions.

Hybrid (embedded) systems are a subclass of embedded systems that are characterized by involving discrete as well as continuous dynamics, such as digital software interacting with an analog environment. Examples of hybrid systems can, e.g., be found in automotive

[*] This work originated at the Institut für Informatik, Technische Universität München, and was supported with funds of the DFG under reference number Br 887/9 within the priority program *Design and design methodology of embedded systems*.

electronics. These include simple systems, such as software-based automatic gear shifting, as well as complex products, such as autonomous cruise control systems (ACC). These ACC systems adjust a car's speed and distance to the car in front depending on different traffic situations, e.g., highway or stop-and-go traffic. Further examples include walking robots, where different discrete phases follow one another and each one is characterized by different continuous dynamics, or manufacturing plants.

In the design of hybrid systems, different time models and techniques from different disciplines—mainly computer science and control theory—are usually employed. An isolated consideration of the discrete and continuous parts of a hybrid system already at the beginning of the development process together with imprecisely defined interaction between these parts can lead to inappropriate or incorrect designs. Therefore, this thesis introduces formal, integrated notations and development techniques for hybrid systems which support development processes that postpone this partitioning of a hybrid system to the later development phases.

Furthermore, the thesis provides a detailed study of the properties which are usually required for hybrid systems and some of their proof methods. This fosters a better understanding of hybrid systems by computer scientists and formalizes parallels between computer science and control theory.

Overview. Section 2 discusses an integrated development process for hybrid systems and thereby shows at which areas of improvement the thesis targets at. The succeeding sections briefly outline the most important ideas and results in [Sta01] on description techniques, refinement and properties of hybrid systems.

2 Integrated Development Process

In this section the integrated development process for hybrid systems which is introduced in [Sta01] as an ideal process to which the thesis contributes is outlined. This process aims at bringing more mathematical rigor and an integrated view on the different aspects of hybrid systems to their development. The process relies on integrated formal notations, analysis and refinement techniques. In contrast to a conventional development process for hybrid systems, it targets at reducing design risks by earlier validation of abstract, integrated system models, before implementation-oriented decisions must be made.

2.1 The Process and its Underlying Ideas

Like other embedded systems, hybrid systems often perform safety critical tasks or are part of mass products. To reduce the risk of highly expensive errors in them, we advocate the use of formal methods and notations wherever possible in their development. *Formal* here means that the methods and notations have a well-defined mathematical foundation and semantics. This helps to obtain unambiguous specifications and it is the basis for rigorous validation and verification techniques. Moreover, it provides the foundation to pre-

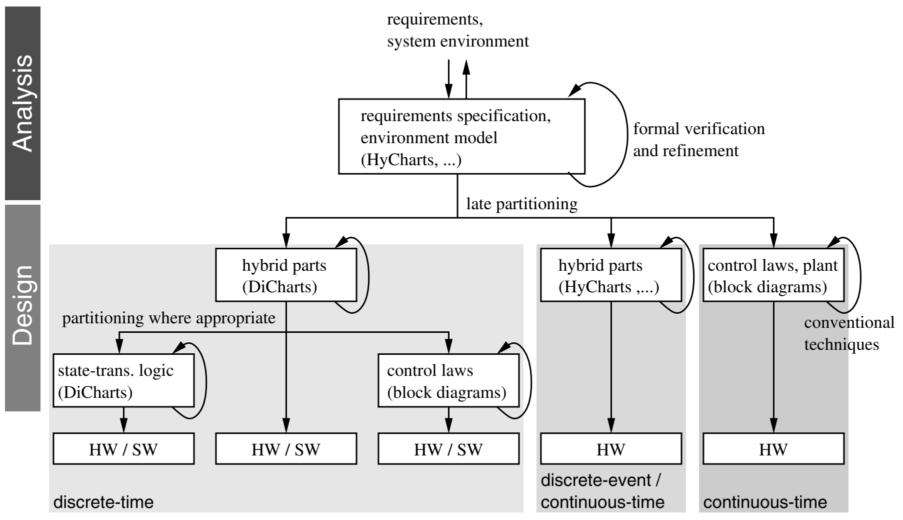

Figure 1: Integrated development process, based on hybrid description techniques (indicated in parenthesis). Note that pure discrete-event parts are subsumed in the hybrid parts.

cisely define the interaction between different formalisms, and—if applied consequently—formal methods can provide a mathematically guaranteed requirements traceability.

In a development process with formal hybrid description techniques, such as the one desired in [Sta01] and depicted in Figure 1, the designer is able to formally specify models for mixed discrete and continuous requirements at early stages of the development process, thereby getting precise documentation. If techniques such as simulation, model checking and consistency checking exist for validation and verification of the requirements captured in such an initial model, design risks can be greatly reduced as early as in the analysis phase and the model can be *systematically* designed to reflect the hybrid requirements (Figure 1, top). Rudimentary versions of such techniques already exist and are an area of current research (e.g., [DFG01]). In later steps, when enough confidence in the model has been obtained, it can be gradually refined towards the detailed design. If mathematical techniques which ensure that the systems' functionality is augmented without violating previously established requirements are used for this stepwise refinement, the traceability of the previously established requirements is automatically guaranteed. In the context of formal methods we also call this *formal refinement*, or simply refinement when the context is clear. For some model components, refinement towards detailed design involves partitioning them into discrete-time, continuous-time and mixed discrete-event/continuous-time subcomponents, or partitioning state-transition logic[1] and control algorithms into different subcomponents (Figure 1, bottom). Typically, the model of the system environment[2], analog filters and analog-digital and digital-analog converters will be isolated into

[1]We use the term *state transition logic* to refer to the logical decision making in hybrid systems. It usually causes qualitative changes in a systems state and is characteristic for hybrid systems.

[2]Such an environment model is typically needed in the development of control systems and embedded systems. It is comparable to the domain model in usual software development, but often more detailed. In particular it also considers environment behavior.

continuous-time or hybrid components. Additionally, some fast control laws may also be implemented in continuous time. The state-transition logic and most control laws will be implemented digitally, and are therefore specified with an underlying discrete-time execution model in the low-level requirements.[3] Here, the state-transition logic and control laws may be split into isolated components. However, as long as there are no specific needs enforcing this separation, such as the application of specialized methods, they can remain in a single component as well (see the branches to discrete-time implementation in Figure 1, left). Digital implementation is motivated by the benefits software-based solution offer. Provided computing hardware is present, the same hardware can be used to run software fulfilling a variety of different tasks in parallel. Replication costs for software virtually do not exist. Furthermore, additional system requirements and performance optimization can be realized with software updates without necessarily requiring modified hardware.

For the state-transition logic, the step from high-level requirements towards implementation-oriented, low-level requirements also includes a change in the time model, from an initial discrete-event model disregarding implementation specific sampling effects to a discrete-time model as present in digital computers. This transition may introduce undesireable effects caused, for example, by missing some events due to inappropriate sampling. Unlike in a conventional process where no common mathematical basis is available at this point of the development process, in the integrated process formal methods, like (formal) refinement techniques, can be used to ensure that the transformation maintains desirable properties. In order to obtain discrete-time control laws, standard techniques can be used to synthesize discrete-time controllers of continuous plants which yield the desired dynamics of the overall system. However, if the coupling of state-transition logic and control laws leads to control moding, i.e. to the discrete switching between control laws, the behavior of the coupled system has to be analyzed in an integrated manner considering both aspects, since the switching can cause instability or poor performance, despite locally optimal control laws in each control mode. In our view, splitting state-transition logic and control laws into isolated components should be avoided in presence of control moding, because it entails the danger of focusing on control laws while disregarding the effect of mode changes initiated by the state-transition logic. If a partitioning is required, e.g., for implementation purposes, the acceptability of the behavior of the coupled system should be ensured before the partitioning, because after the partitioning it would be necessary to reassemble the parts for effective analysis, which may be difficult. Nevertheless, it is important to bear in mind that integrated analysis and synthesis methods are still a topic of research. However, the integrated development process at least enables formalization of the problem, while in a conventional process there is no integrated view on state-transition logic and control laws. Thus, clearer documentation is obtained which thereby helps designers to focus on the problem. Besides that, if partitioned submodels are constructed, conventional techniques can still be applied to realize those aspects which only affect the respective part (Figure 1, bottom), just as in a conventional development process which relies on an early partitioning of state-transition logic and control laws.

Summary. To summarize the previous paragraphs, the availability of formal hybrid de-

[3]Note that even if the state-transition logic builds upon the event mechanism of an underlying operating system, it can nevertheless only react in dependence of the clock rate given by the digital hardware.

scription techniques and supporting methods for them pushes the point at which systematic development can begin to the beginning of the analysis phase. (Systematic development here means development with mathematically precise documentation.) For those system components which can be implemented in a digital or analog manner, a partitioning into discrete-time and continuous-time submodels can be postponed to subsequent development phases. A separation of state-transition logic and (discrete-time) control laws can even be avoided completely. In any case, a development process with hybrid description techniques allows us to obtain greater confidence in the model before a partitioning. Namely, testing and model-checking techniques can be used to analyze requirements and formal refinement techniques can be used to guarantee the traceability of these requirements. By postponing implementation-related questions, changing requirements can be taken into account more easily. Thus, errors made in the initial development phases can be found earlier which in turn makes them cheaper to correct.

2.2 Supporting Techniques Developed in the Thesis

Description techniques. [Sta01] contributes to the development process outlined above in several ways. It proposes the formal hybrid description technique *HyCharts*, which supports modular specification of hybrid systems (cf. Section 3). HyCharts resemble the notation introduced in the software engineering method for real-time object-oriented systems (ROOM) [SGW94], but extend it to the description of hybrid and continuous behavior. ROOM is one basis for the ongoing standardization of the UML [Gro00] dialect for real-time systems, UML-RT. As in ROOM and in agreeance with the UML's concept of different system views, HyCharts consist of two subnotations: *HyACharts* for the specification of system architecture and *HySCharts* for the specification of component behavior. The main application area for HyCharts ranges from requirements specification to the design phase. For the discrete-time models which are derived from HyCharts and which occur in these later development phases, *DiCharts* are introduced. They exactly correspond to HyCharts but use an underlying discrete-time execution model. In particular, they allow us to closely couple state-transition logic and discrete-time control laws.

Refinement. As far as potential unexpected effects are concerned, a highly critical step in the transition from the analysis phase to design and implementation is to move from a mixed discrete-event/continuous-time model to a discrete-time model, which enables efficient implementation. The thesis therefore elaborates methods which guide this transition to discrete-time models while maintaining vital classes of properties of the initial model (cf. Section 4). An important characteristic of these methods is that they help to make assumptions about the environment explicit. Since they result in constraints on the sampling rate, they can also be used to verify the adequacy of sampling rates present in legacy components or in components developed by third parties w.r.t. a given environment model. Furthermore, the thesis also outlines how models can be partitioned into discrete-time and remaining continuous-time and hybrid subsystems, which is useful if a separate development process is intended to be pursued for subsystems that are supposed to be implemented in analog, digital or mixed-signal hardware.

Refinement techniques for architecture and automata diagrams in the style of those in

[Sch98] which do not affect the time model are also considered in the thesis.

Properties. Obviously, for meaningful refinement techniques it is necessary to examine which classes of properties they maintain. Hence, the thesis studies and classifies typical properties that hybrid systems have to obey (cf. Section 5). This provides a deeper insight into the characteristics of hybrid systems and is also used to make relationships to control theory and computer science explicit. As an addition to the general definitions for properties of hybrid systems which is given in the thesis, the thesis outlines proof methods for some of them and explains why certain classes of properties have not received much attention in computer science so far.

3 HyCharts – Specification of Hybrid Behavior

In this section we present the basic ideas of *HyCharts*, the visual, modular specification technique for hybrid systems proposed in [Sta01].

Based on a simple and powerful computation model for hybrid systems and with a collection of operators on hierarchic graphs, which captures the syntax of the visual notations, as tool-set, [Sta01] defines HyCharts by following the ideas in [GSB98]: They consist of two different relational interpretations of hierarchic graphs, an *additive* one and a *multiplicative* one. Under the additive interpretation the graphs are called *HySCharts* and under the multiplicative one they are called *HyACharts*. HySCharts are a visual representation of hybrid, hierarchic *state transition* diagrams. They model the control-flow within hybrid components. HyACharts are a visual representation of hybrid *data-flow* graphs (or architecture graphs). They model the data-flow between the components of a hybrid system and allow the designer to compose components in a modular way. The behavior of these components can be described by using HySCharts or by any technique from system theory that can be given a compatible semantics, including differential equations. Simple syntactic transformations, corresponding to macro expansion, lead from the graphical notation used by the designer to a hierarchic graph whose semantics results from the respective interpretation of the graph. For HyACharts, the syntactic transformation is trivial and applying the multiplicative interpretation to the resulting graph yields the semantics of a component. For HySCharts, transformation is more complex. Here, two transformations are necessary, one to extract the discrete dynamics from the diagram and one to extract the analog dynamics. For the analog dynamics, a time-extended variant of the additive interpretation yields its semantics. For the discrete dynamics, the additive interpretation directly yields its semantics. The coupling of the analog and the discrete dynamics is formalized in the hybrid computation model.

The algebra-based semantics which maps graphs to relations has three main advantages. First, up to (rather simple) syntactic transformations, it corresponds almost one-to-one with the visual notation used by software engineers. Second, as shown in [GBSS98], it comes equipped with a set of graph equations (algebra) which define how to (visually) transform components in a semantics preserving way. As a result, the algebra may be used by engineers both for optimizations and to check the equivalence of different components.

Third, similarly to [Bro97], it comes equipped with a very simple notion of refinement and its associated compositional refinement rules. This is an essential prerequisite for proving that a successively modified implementation meets its original specification.

For the description of discrete-time components the thesis moreover introduces *DiCharts* as the discrete-time counterpart to HyCharts. Here the same hierarchic graph framework can be used, since the framework can be defined with the time model as a free parameter.

4 Refinement

In this section we explain the refinement notion used in the thesis and outline the thesis' results on understanding time-discretization as refinement.

Notion of Refinement. Before we introduce the refinement notion used in [Sta01], we have to define the mathematical model of components in HyCharts. In a hybrid system the data-flow between components may be continuous (think of analog devices), so we take the non-negative real numbers $\mathbb{R}_+$ as the abstract time axis. The data exchanged along a channel with type A over time defines a mapping $a \in \mathbb{R}_+ \to A$. We call such a mapping a *dense communication history* (or *dense stream*). On the level of semantics the behavior of a component C can be completely described by an *input/output relation*, i.e. by a relation between the histories of its input channels and the histories of its output channels. For input channel type I and output channel type O, the type of C is $C \subseteq (\mathbb{R}_+ \to I) \times (\mathbb{R}_+ \to O)$.[4] Relations are used instead of functions, because this allows us to express nondeterminism. The relations must be total in the input histories, i.e. for every input history an output history must exist which is related to the input history by the relation. Furthermore, we assume that the relations are *causal*, i.e. that they are defined such that the data occurring in the output histories up to time t only depends on the input history received up to t.

We define refinement on basis of set inclusion. For two relations A and B, A is a refinement of B iff $A \subseteq B$ holds. If A and B are components, i.e. if they are input/output relations over dense streams, refinement of B by A means that the behavior of A is contained in that of B. Thus, A may only be more precise than B. Every behavior $(\iota, o) \in A$ also is a possible behavior of B. Properties which constrain all possible behaviors of a system clearly are maintained by refinement. As the thesis shows, some further important classes of properties are also maintained by this refinement notion (cf. Section 5).

Refinement and Time. Moving from an abstract model based on a continuous time scale to implementation amounts to changing to a discrete-time execution scheme for those components in the model which are implemented on (or in) digital hardware. Such discrete-time execution usually is desired for large parts of a hybrid system. In order to ensure that vital properties of the abstract model are satisfied in the implementation oriented discrete-time model, the change of the execution scheme must be performed in a controlled way. Therefore, the thesis identifies conditions under which the transition from continuous-time to discrete-time is a formal refinement.

[4]Note that I and O may be tuples of inputs and outputs.

The most interesting step in the transition from HyACharts with components operating in continuous-time to HyACharts containing subcomponents that operate in discrete-time is the time-discretization of primitive components specified with HySCharts. This time-discretization causes two kinds of effects. First, discrete transitions cannot be taken at the ideal point in time when they become enabled for the first time, but can only be taken with some delay at the next sampling instant when they are still enabled. Second, the time-discrete component can not permanently compute output histories from control laws and the input, but it can only compute new output at sampling instants. Between the sampling instants, the output history must be extrapolated, usually to a step function. This leads to further deviations between the ideal continuous-time output and the output constructed from a discrete-time signal. As the used refinement notion only allows to reduce the amount of non-determinism in a component, time-discretization can only be understood as refinement if the abstract component already contained tolerances within which all its produced behaviors lie. A discrete-time refinement of a component must then ensure that all its behaviors also satisfy the specified tolerances.

As a main result, the thesis provides methods to systematically derive a discrete-time refinement of components specified with HySCharts. The methods require various knowledge about the dynamics of the input a component receives. The minimum time between events, Lipschitz constants and possible errors associated with continuous periods of evolution are typically needed. For the discretization of a HySChart's analog part the thesis explains how methods from numerical mathematics and control theory can be employed. For the time-discretization of the state-transition logic specified with a HySChart, the thesis shows how to systematically derive constaints on the sampling rate for the discrete-time refinement from the tolerances allowed in the abstract model and the dynamics of the input the component receives.

5 Properties of Hybrid Systems

Developing hybrid systems as well as designing formal methods, such as validation and refinement techniques for them requires a deeper understanding of their essential properties. Therefore, the thesis studies and classifies important properties of hybrid systems, and formalizes their relationship to properties usually considered in computer science. As a further result, proof methods for some of these properties (*stability and attraction*) are provided and related to abstraction in computer science.

Classification. The properties considered result from the evaluation of nine hybrid systems case studies and a number of text books on control theory. [Sta01] defines a black box and a glass box view on hybrid system components which corresponds to HyCharts and allows the formal definition of the considered properties for hybrid systems. In contrast to other general definitions of (control) system properties in system theory or control theory the definitions the thesis gives can also be applied to nondeterministic systems, as nondeterminism is typical for system designs in computer science in early development steps. Based on the semantic models relative to which the validity of the properties is defined the

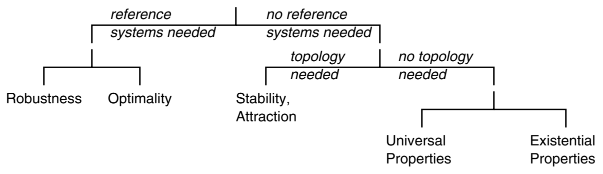

Figure 2: Classification of properties.

thesis classifies the properties into ones that are expressed w.r.t. other systems which we call *reference systems*, into properties that depend on topologies and into properties independent of reference systems and topologies. Figure 2 depicts the resulting classification.

Informally, stability expresses that small disturbances of a system's input (or state) should only cause small disturbances in the system's output (or state). Attraction basically means that a given state is always reached after a finite or infinite amount of time and not left again thereafter. Invariance properties, which are well known in computer science and play a central role in many of the considered case studies, are universal properties, i.e. they are required to hold for all possible input/output behaviors (or *computations*) of a system. Also note that with respect to the depicted classification safety as well as liveness properties are universal properties as they (implicitly) involve universal quantification over a system's computations [CMP91].

Parallels to computer science. [Sin96] mentions that there is a correspondence between invariance in control theory and safety in computer science, and between attraction in control theory and liveness in computer science. Based on the formal definitions the thesis makes this correspondence explicit by identifying mathematical topologies where stability is a safety property and attraction is a persistence property in computer science terminology [CMP91].

Proof methods. For stability and attraction [Sta01] defines a proof method which is a generalization of the *Liapunov theory*[5] and results from adapting results from general systems theory to the framework in the thesis. Under certain circumstances the character of Liapunov functions as abstractions can be made explicit by relating them to Galois connections which define a popular theory of abstraction in computer science. The thesis identifies these circumstances and shows how the Liapunov function can be used to define the abstraction and concretization function of a Galois connection.

6 Conclusion

The paper outlined the main ideas and results in [Sta01]. In particular a development process for hybrid systems was discussed which aims at reducing design risks by postponing the implementation related partitioning of a hybrid system into discrete-time and

[5]The Liapunov theory is the basis for most stability proofs in control theory.

continuous-time subsystems to later development phases, where more confidence on a design has already been obtained. Moreover, the paper showed how [Sta01] contributes to the feasibility of such a development process.

Literaturverzeichnis

[Bro97] M. Broy. Refinement of Time. In *Proc. of ARTS'97*, LNCS 1231. Springer-Verlag, 1997.

[CMP91] E. Chang, Z. Manna, and A. Pnueli. The Safety-Progress Classification. In *Logic and Algebra of Specifications*, NATO ASI. Springer-Verlag, 1991.

[DFG01] DFG. Schwerpunktprogramm KONDISK (Analyse und Synthese kontinuierlich-diskreter Systeme). http://www.ifra.ing.tu-bs.de/ kondisk/, 2001.

[GBSS98] R. Grosu, M. Broy, B. Selic, and Gh. Ştefănescu. Towards a Calculus for UML-RT Specifications. In *7th OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, Technical Report TUM-I9820. Technische Universität München, 1998.

[Gro00] Object Management Group. Unified Modeling Language (UML), version 1.3. http://www.omg.org/technology/documents/new_formal/unified_modeling_language.htm, 2000.

[GSB98] R. Grosu, Gh. Ştefănescu, and M. Broy. Visual Formalisms Revisited. In *Proc. International Conference on Application of Concurrency to System Design (CSD'98)*, 1998.

[Sch98] P. Scholz. *Design of Reactive Systems and their Distributed Implementation with Statecharts*. PhD thesis, Technische Universität München, 1998.

[SGW94] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons Ltd, Chichester, 1994.

[Sin96] M. Sintzoff. Abstract Verification of Structured Dynamical Systems. In *Proc. of Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[Sta01] T. Stauner. *Systematic Development of Hybrid Systems*. PhD thesis, Technische Universität München, 2001.

Thomas Stauner studied computer science at the Technisch Universität München and the University of Edinburgh. For his diploma thesis on a case study on specification and verification of an automotive system he was awarded the Chorafas prize in 1997. After that he worked as a researcher under the supervision of Prof. Dr. Manfred Broy at the Institut für Informatik, Technische Universität München within the DFG priority program *Design and design methodology of embedded systems*. In 2001 he obtained the doctoral degree at the Technische Universität München for his thesis about the systematic development of hybrid embedded systems. Today he is working for BMW Car IT, a subsidiary of the BMW Group, where he is concerned with innovative solutions for ensuring safe and secure automotive software.