A Conceptual Reference Model of Modeling and Verification Concepts for Hybrid Systems

Andreas Müller[1], Stefan Mitsch[2], Werner Retschitzegger[1], and Wieland Schwinger[1]

[1] Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria
[2] Computer Science Dept., Carnegie Mellon University, Pittsburgh, PA-15213, USA

Abstract. *Cyber-physical systems* (CPS), which are computerized systems directly interfacing their real-world surroundings, leverage the construction of increasingly autonomous systems. To meet the high safety demands of CPS, verification of their behavior is crucial, which has led to a wide range of tools for modeling and verification of hybrid systems. These tools are often used in combination, because they employ a wide range of different formalisms for modeling, and aim at distinct verification goals and techniques. To manage and exchange knowledge in the verification process and to overcome a lack of a common classification, we unify different terminologies and concepts of a variety of modeling and verification tools in a *conceptual reference model* (CRM). Furthermore, we illustrate how the CRM can support comparing models and propose future extension.

1 Introduction

Systems that exhibit physical behavior by interfacing and interacting directly with their real-world surroundings through sensors and actuators are known as *cyber-physical systems* (CPS). CPS become increasingly autonomous (e. g., autonomous cars); consequently, significant demands are imposed on the *safety* of such a CPS and the knowledge needed to design and implement them correctly. Therefore, the field of *formal verification*, i. e., mathematically proving that a CPS behaves as intended, is key to engineering CPS for safety-critical application domains. The behavior of a CPS can be described using *hybrid system* models [2], which simultaneously capture the continuously evolving real-world behavior and the discrete control decisions of the CPS within one model.

In order to model a CPS and formally verify the desired behavior, the computational and the physical behavior of a CPS need to be considered in conjunction, which introduces unprecedented complexity into verification. Modelers face many modeling and verification tools, which employ a wide range of modeling formalisms (e. g., hybrid automata [18], hybrid programs [36,38]), aim at distinct verification goals (e. g., safety, liveness) and incorporate heterogeneous verification techniques (e. g., theorem proving, reachability analysis). Often, using multiple tools in combination is beneficial because their capabilities differ

strongly. The downside of this diversity are *compatibility* issues, where specifically questions of knowledge management arise, such as: *(i)* which model representation is useful for which aspect of the system? *(ii)* which parts of the system can be formally verified using which tools? *(iii)* what are the trade-offs between modeling and verification (detail vs. automation)? *(iv)* which parts of a system are verified and how should the verification results be composed to a comprehensive correctness argument? A major difficulty for systematically analyzing modeling and verification and managing knowledge in the verification process of hybrid systems is a lack of a common classification of *hybrid system modeling and verification* concepts.

To overcome this lack of a common classification, we unify different terminologies and concepts of a variety of modeling and verification tools in a *conceptual reference model* (CRM), methodologically adhering to our previous work (e. g., [49]). We illustrate how the CRM can assist in classifying the capabilities of modeling formalisms and tools. Additionally, we identify future extensions and enhancements for modeling and verification tools for CPS.

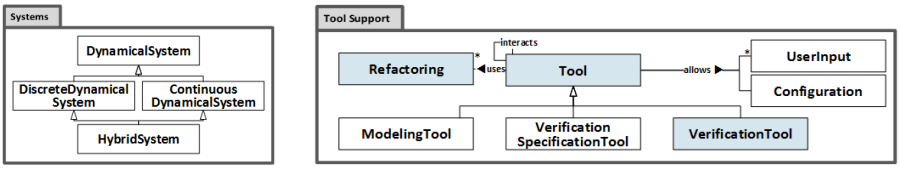
2 Related Work

To the best of our knowledge, no other CRM for hybrid system modeling and verification concepts has been proposed so far. Nevertheless, prior surveys on CPS and hybrid system modeling and verification provide classification fragments, which will be discussed below.

Broman et al. [8] introduce a coarse-grained model for categorizing hybrid systems. Their framework comprises *Viewpoints* (of stakeholders and their concerns), *Formalisms* (modeling formalisms for hybrid systems) and *Languages and Tools* (which implement formalisms). They conclude that their framework serves as a basis for assisting CPS designers in the modeling process. Their framework reviews tools primarily based on the requirements of stakeholders, whereas we focus on the engineering and knowledge representation aspects of hybrid systems design. Alur [1] reviews formal verification approaches, but not modeling and tool support. Carloni et al. [9] analyze the syntax and semantics for hybrid systems modeling w.r.t. verification and simulation. We, in addition, discuss modeling and tool support.

A large body of research addresses solely the systematic modeling and specification of CPS, but does not address verification: Giese et al. [15] survey visual model-driven development of software-intensive systems. Shi et al. [45] provide a short overview and further research challenges of CPS. Sanislav et al. [43] focus their work on challenges, concepts and research goals in the area of CPS. Wan et al. [48] investigate the applicability of different composition mechanisms for cyber-physical applications. Kim et al. [23] provide a broad overview of CPS research and Lee [27] examines the challenges in designing CPS. Finally, the Open Model Community's[3] formal definition of modeling methods [13] may act as an alternative to UML for representing our CRM.

[3] `http://www.omilab.org`



(a) Systems (b) ToolSupport

Fig. 1: Systems and tool support concepts of the CRM

3 Conceptual Reference Model

In this section, we present a CRM of modeling and verification concepts for hybrid systems. In principle we followed a top-down approach for constructing the CRM, meaning that several concepts of the CRM have been adopted from existing other surveys in this area as referred to in Section 2. We supplemented our CRM in a bottom-up way with concepts prevalent in existing tools. Finally, we structured our CRM into four packages: (i) the `Systems` package describes the real world systems; (ii) the `Modeling` package abstracts from real-world systems to models of their behavior and specifications of important properties; (iii) the `Verification` package aims at verifying the modeled systems; (iv) the `ToolSupport` package contains tool related aspects. In the following sections, the concepts of the CRM are described along these four packages (see `http://cis.jku.at` for complete CRM).

We express the CRM as Unified Modeling Language (UML) classes, since UML is the prevailing standard in object-oriented modeling[4] and expose the basic components of hybrid systems and the interrelations between them. Naturally, the CRM thus serves also as a *framework*, which can be extended by means of sub-classing if further hybrid system concepts need to be captured.

3.1 Systems

The classes in the systems package (cf. Fig. 1a) describe a high-level systems perspective to anchor modeling and verification tools. We follow Teschl [46], and distinguish `DynamicalSystems` into `DiscreteDynamicalSystems` (state space is $\mathbb{N}/\mathbb{Z}$) and `ContinuousDynamicalSystems` (state space is $\mathbb{R}$); systems that have both characteristics are `HybridSystems` [18], focused on in this paper. Specifically, the dimensions of space and time are important characteristics for many systems. A difference in handling those in a discrete or continuous manner indicates a potentially fundamental conflict between modeling concepts and tools.

[4] UML meta-model as included in the OMG "Unified Modeling Language: Superstructure" version 2.4.1, available at `http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/`.

3.2 Modeling

For the modeling package (cf. Fig. 2a) we follow Gupta [16] to distinguish a model and its correctness specification (implementation and specification).

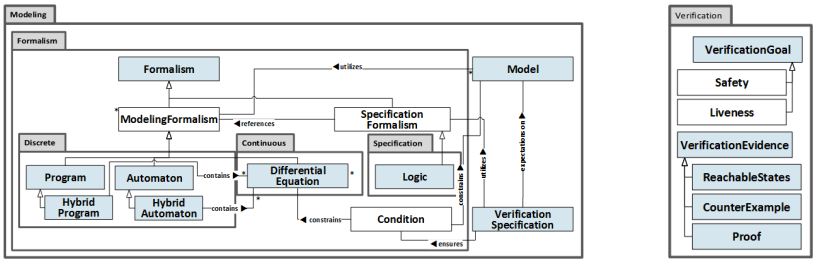
Model. A `Model` captures the relevant features of a dynamic system. It is expressed in a `ModelingFormalism` and can be constrained by `Conditions`.

Verification Specification. A `VerificationSpecification` describes a model's expected behavior utilizing a `SpecificationFormalism` [11], [16]. A verification specification, being a logical formula in many approaches (e. g., [26], [36]), consists of a `StartCondition` that specifies the initial conditions under which we want a system to be safe to start, and a `CorrectnessCriterion` that we want a system to fulfill (e. g., throughout, after all, or after at least one of its executions). Furthermore, it is often possible to annotate models with `Hints/Strategies`, that guide a verification tool but do not influence the behavior of a model directly (e. g., *invariants* in KeYmaera [40] and UPPAAL [26]).

Formalism. A major part of the Modeling package is the `Formalism` sub-package, which is divided into modeling formalisms for creating models and specification formalisms for creating verification specifications (these may reference the created models). The included formalisms are the most commonly used in literature, namely `Automata` and `Programs` [16] for modeling of discrete systems, `Differential Equation` for modeling of continuous systems [10], as well as their combination in the form of `HybridAutomata/HybridPrograms` [18], [36]. In order to constrain a model to realistic behavior (e. g., the "bouncing ball" can never fall through the floor), the CRM introduces conditions. Following Meyer [28], these conditions are further subdivided into `PreConditions`, `PostConditions` and `Invariants`. Moreover, a modeling formalism can have multiple characteristics further describing its capabilities. We include subclasses of `Characteristic` in the CRM to handle `Compositionality` (compositional models, for instance, through `Parallelism` [19], `Urgency` [5], `Synchronization` or *sequences* [44]) and `Non-Determinism` (e. g., *non-deterministic choice* in $\mathtt{d}\mathcal{L}$ [36]).

Automaton and Program. An automaton comprises a set of `States` and a set of `Transitions`, and can be visualized as a directed graph [20]. A condition, when attached to an `AutomatonElement`, restricts or details the behavior of the automaton. Another modeling formalism are programs, representing a sequence of instructions. Although they are often interchangeable we separate these formalisms, because their structural differences can be utilized by verification tools.

Differential Equations. Following [51] we classify differential equations (DE) into `PartialDE` (PDE) and `OrdinaryDE` (ODE). Both can have special restricted constant and linear forms (i. e., `LinearPDE`, `LinearODE`). In accordance with [37], we allow conditions to restrict a DE to remain within a particular region, transforming DEs into `DifferentialAlgebraicEquations` (DAE). DAEs can further be equivalently transformed into differential inclusion, which is useful to express disturbance in continuous dynamics.



(a) Modeling (b) Verification

Fig. 2: Modeling and Verification Concepts of the CRM

HybridAutomaton (HA) and HybridProgram (HP). The generic concept of HA is the basis for numerous hybrid formalisms with different levels of expressiveness and detail, such as *HybridUML* [6] or *Hybrid Petri Nets* [12], [34]. For HA, we introduce a `ContinuousState` that references a set of DEs. These differential equations represent the continuous behavior of a system while their respective state is active. `TimedAutomata` are restricted to modeling real-time systems and correspond to HA with only clocks [4]. Like above, we introduce HPs [36], [38] as sub-class of programs which allows differential equations as instruction. As already mentioned, although HA can be encoded as HPs [36], they differ in structural aspects that can be exploited by hybrid system verification tools.

Logic. A verification specification is expressed in terms of a `Logic` [16], such as Temporal Logic (TL) [41] or differential dynamic logic (`d`$\mathcal{L}$) [36]. These logics differ in terms of capabilities and expressiveness, and support various kinds of quantifiers and modalities (e. g., []-safety or <>-liveness modalities in `d`$\mathcal{L}$, A or E path quantifiers in CTL). Logics currently included in the CRM are `CTL`, `LTL`, and their common superset `CTL*`, `dL` used together with HP for deductive verification and `TCTL` used in model checking of timed systems.

3.3 Verification

Verification Goal. Verification is usually defined to check the behavior of a system w.r.t. its intent [24]. Kern et al. [22] conceptually distinguish between *property-* and *implementation verification.* Property verification is concerned with specifying properties that are desired for a design (equivalent to what we consider *verification* in our CRM), while implementation verification deals with the relationship between high-level models and the implementation. In the CRM (cf. Fig. 2b) we focus on property verification, as implementation verification is rather a topic of model transformation which is not dealt with here. For ensuring such an intended behavior, formal verification methods are typically distinguished into *Model Checking* and *Verification by Deduction* [24]. Such methods provide rigorous evidence (e. g., a proof) that a specification aiming at a `VerificationGoal` is correct. Common verification goals include `Safety` (i. e.,

something will *not* happen) or `Liveness` (i. e., something *must* happen) [25], `Controllability/Reactivity` [38], `Fairness` [47] and `Deadlock Freedom` [7].

Verification Evidence. `VerificationEvidence` witnesses the correctness of a model w.r.t. a verification specification. What is considered a witness typically depends on the employed formal verification method (e. g., reachable states for model checking, a formal proof for deductive verification or counterexamples in both methods to witness correctness violation).

Proof. A `Proof` consists of arbitrary many proof steps and aims at verifying that the model is correct w.r.t. the specification. Entire proofs or parts of a proof might be transferred to other users using either the same tool or other tools to make progress or even close the entire proof. An implementation of the CRM should support composition and decomposition of proofs, as well as exchanging proofs, partial proofs, and lemmas. An important aspect arises from exchanging evidence: how can the correctness of exchanged artifacts be substantiated (e. g., certificates or by providing an exact listing of all proof steps)?

CounterExample. A specification for a selected technique can be refuted by a `CounterExample`, which is mutually exclusive to a successful proof. Multiple counter examples might be found for each open proof step. As soon as one is found, the refutation of the entire verification specification is inferred.

ReachableStates. Another possible output is the set of `ReachableStates`, if a reachability analysis was performed.

3.4 Tool Support

Because of the large number of tools available for modeling and verification of CPS (cf. Fig. 1b), in this section we restrict the discussion to classes related to modeling and proof collaboration, as motivated by our previous work [30]. However, as usual, the package can be extended to fit other tools as required.

Tool. The ToolSupport package includes concepts for `Tools`: (i) a `ModelingTool` supports users in creating a model using one of the respective formalism, (ii) a `VerificationSpecificationTool` allows formulation of a verification specification about a model, and (iii) the `VerificationTool` takes the model and its specification and produces a verification result.

A tool manipulates various `Artifacts`; it uses input (e. g., a model) to produce a corresponding output (e. g., a verification specification). Tools might require `UserInput` at run-time and additional prior `Configuration` (i. e., meta information required to run the tool, e. g., library paths). For example, verification tools often make a trade-off between expressiveness to achieve full automation (e. g., affine linear dynamics in SpaceEx [14]) and user interaction to handle undecidability (e. g., KeYmaera [40]). Furthermore, tools might interact with each other to enable `Collaboration` between instances of a single tool (e. g., multiple users might collaboratively produce a complex model) or different kinds of tools (e. g., different verification tools can be used to verify a single specification).

Refactorings. Artifacts can influence other artifacts (e. g., a counterexample may lead to revision of the initial model), since hybrid systems are often developed incrementally by model refinement. `Refactorings` support the refinement process through automated restructuring of artifacts. In case a model is accompanied with a correctness specification and a proof, it is important to spare full reverification when the behavior of the model is refactored [31].

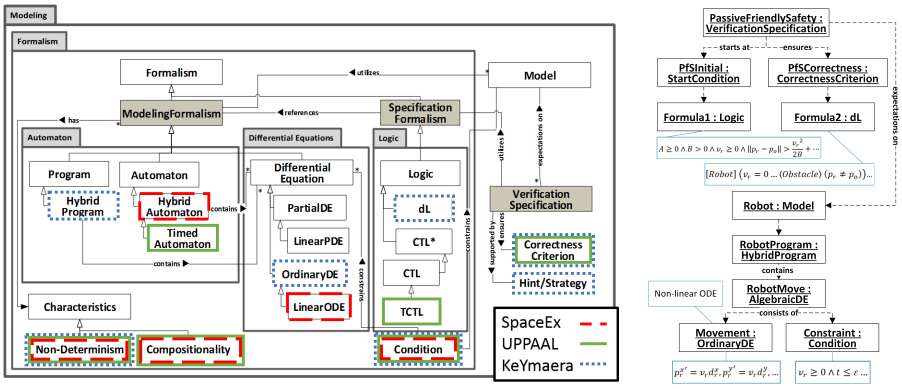
Another benefit of refactoring is to reduce verification complexity. For example, concurrent transition systems are exponentially harder to verify than sequential ones [17]; so concurrent real-world components that are independent in their read-write variables can be modeled sequentially in arbitrary order to reduce verification complexity. We can therefore further distinguish refactorings that change the behavior of a model into those increasing modeling detail (`Refinement Refactorings`) and those reducing details (`Abstraction Refactorings`).

Verification Tools. In our CRM we include two types of verification tools corresponding to the two major verification techniques: *model checking* and *theorem proving* [33]. For theorem proving, we introduce the class `DeductiveVerificationTool`. These tools are based on some logic and use inference rules (i. e., proof rules) to transform formulas until they yield axioms or logical tautologies. Kern et al. [22] refer to these techniques as *Deductive Methods*, while Clarke et al. [11] call them *Theorem Proving*. Model checkers calculate the states that can be reached by a model and check if all are desirable. Since the term model checking, however, refers to a technique mainly used for verification of purely discrete systems, we choose *reachability analysis* as the equivalent of model checking for hybrid systems. A `ReachabilityAnalyzer` calculates reachable states either in an exact manner by limiting the continuous dynamics to simple abstractions or in an approximate manner by over-approximating the set of reachable states [3]. Many of the techniques that work with (over-)approximations have to deal with floating point issues. The exactness of a verification technique (e. g., floating point variables do not store exact real valued numbers, but might result in rounding errors) has to be exchanged when different tools interact.

4 Knowledge Representation in the CRM

In this section we use the CRM to represent knowledge about hybrid systems.In order to simplify comparison of verification and modeling tools with the help of our CRM, we deduce a set of criteria that allow to classify and evaluate hybrid systems modeling and verification tools.

Detailed criteria are analogous to the concepts of the CRM and reveal fundamental differences between the hybrid systems modeling and verification tools as discussed below. As sample tools, we chose verification tools for hybrid systems (SpaceEx [14], and KeYmaera [40]) and for timed systems (UPPAAL [26]). The tools have different capabilities (cf. Fig. 3a): SpaceEx is a verification platform



(a) Knowledge about SpaceEx, UPPAAL and KeYmaera (b) Robot Example

Fig. 3: Knowledge representation in the CRM

for hybrid systems by performing reachability analysis, UPPAAL is an environment for verification of timed automata through model-checking and reachability analysis, and KeYmaera is a theorem prover for hybrid systems.

Modeling Formalism. Each of the tools uses a slightly different modeling formalism. SpaceEx and UPPAAL both work with networks of automata, where SpaceEx accepts HA and UPPAAL is restricted to timed automata. KeYmaera chooses a different approach and uses HPs. As for *continuous dynamics*, UPPAAL is restricted to the use of synchronous clocks, while the other tools can handle differential equations. However, SpaceEx is limited to affine linear dynamics while KeYmaera can also handle non-linear ODEs, algebraic DEs and differential inclusion. Additionally, SpaceEx and UPPAAL—as they use networks of automata—support compositionality (e. g., by parallel composition).

Timed automata can be translated into HA, but not necessarily the other way around, as modeled through timed automata being a subclass of HA. This in turn means that transformations from SpaceEx models containing differential equations will fail when translated to UPPAAL. Similarly, specifications provided in UPPAAL (e. g., safety properties) in terms of TCTL, will be lost when translating a model to SpaceEx. Furthermore, both SpaceEx and UPPAAL support guards and invariants (highlighted in the CRM by means of the marked Condition class), allowing loss-less translation of conditions.

Additionally, the CRM can be implemented as an object-oriented *knowledge representation scheme* to provide automated support for model selection and compatibility checks. For example, the *modeling formalism* for UPPAAL is timed automaton while it is HA for SpaceEx, and HP for KeYmaera. For a concrete model using a timed automaton as modeling formalism a knowledge representation query would return tools that can handle timed automata directly (SpaceEx and UPPAAL), and those tools that can handle the model with some transformation (KeYmaera). In the latter case, the models resulting from

the transformation, however, are likely to not benefit from KeYmaera in the fullest possible extent, because the structure of the source model is different from the expected program structure. When comparing SpaceEx and UPPAAL, the knowledge representation would also provide that an instance of a SpaceEx model (containing a HA) cannot necessarily be assigned to UPPAAL, as the modeling formalism of UPPAAL cannot handle HA in general.

Specification Formalism. KeYmaera and UPPAAL use logical statements to define desired properties about a system, whereas SpaceEx computes a set of reachable states that can be compared either for intersection with the set of unsafe states (safety) or with the set of goal states (liveness).

The CRM also supports comparison of specification formalisms, as a similar class hierarchy as described above, also exists for logics. While UPPAAL uses a subset of TCTL for its specifications, KeYmaera uses $\mathsf{d}\mathcal{L}$.

Verification Specification. For conditions, all tools support guards and invariants to restrict the behavior of the models. While SpaceEx returns a set of reachable states which has then to be analyzed for intersections with desirable or undesirable sets of states, UPPAAL supports the use of path formulae (further classified into reachability, safety and liveness) and KeYmaera allows arbitrary $\mathsf{d}\mathcal{L}$ fomulae to specify safety and liveness properties. KeYmaera furthermore allows annotating its models with additional conditions (e. g., variants and invariants), to support the tool during verification of the models.

The CRM supports comparing the available kinds of correctness criteria specifiable within different tools. When translating from SpaceEx to UPPAAL, additional correctness criteria must be specified after the translation so that UPPAAL can verify the model. From UPPAAL to KeYmaera we may want to enrich the model with annotations to guide the proof search.

Applying the CRM. We use a robot collision avoidance model [29] to illustrate how hybrid systems models can be captured in the CRM. The collision avoidance model was designed using *hybrid programs* as a modeling formalism; its correctness requirements are defined by a specification. Since the robot moves along sequences of circular arcs and must not drive backwards, the hybrid program contains non-linear differential-algebraic equations. The starting condition *PfSInitial* of the system is a logical formula, which describes conditions on acceleration A, braking B, robot speed v_r, robot direction d_r, robot positions p_r, and obstacle positions p_o, under which it is safe to start the robot. The correctness criterion *PfSCorrectness* expresses that all possible behavior of the robot has to avoid collision, and furthermore has to retain at least one maneuver for obstacles to avoid collision as well, which means that it uses nested modality operators and, hence, can only be expressed in $\mathsf{d}\mathcal{L}$. From these instances we immediately see that KeYmaera is the tool which fits best for verification of this model, since $\mathsf{d}\mathcal{L}$ and non-linear DE are not supported by SpaceEx or UPPAAL (cf. Fig. 3b).

5 Conclusion and Future Work

In this paper, we introduced a conceptual reference model that can be used to analyze the properties of hybrid system modeling and verification tools and classify them accordingly. The resulting CRM can be used to (i) unify the used terminology, (ii) compare the capabilities of modeling and verification methods, and (iii) represent and exchange knowledge about those methods and about models. Furthermore, we see several promising application areas for such a CRM, spanning from education purposes in order to provide students with a road-map for CPS development and verification, to an object-oriented knowledge representation system, which can be used to automatically check the compatibility between models and tools, and exchange information between different tools as emphasized for interchange formats (e. g., [5], [32], [35]).

The CRM is a first step towards a comprehensive classification framework of hybrid system modeling and verification concepts and should be extended with further details: For future work, we plan to extend the CRM with *stochastic* or *probabilistic hybrid systems* (e. g., [21], [39], [50]), hybrid games (e. g., [42]), component-based modeling, and transformation. Finally, we plan to conduct a survey of hybrid system modeling and verification approaches using a criteria catalog derived from the CRM.

Acknowledgements This work was funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) grant FFG FIT-IT 829598, FFG BRIDGE 838526 and FFG Basisprogramm 838181, and by PIOF-GA-2012-328378. The authors thank Andre Platzer for fruitful discussions and feedback.

References

1. Alur, R.: Formal verification of hybrid systems. In: Proc. of the 9th ACM Intl. Conf. on Embedded software. pp. 273–278. EMSOFT '11, ACM, NY (USA) (2011)
2. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theor. Comput. Sci. 138(1), 3–34 (1995)
3. Alur, R., Dang, T., Ivančić, F.: Reachability Analysis of Hybrid Systems via Predicate Abstraction. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC, LNCS, vol. 2289, pp. 35–48. Springer (2002)
4. Alur, R., Dill, D.: The theory of timed automata. In: Bakker, J., Huizing, C., Roever, W., Rozenberg, G. (eds.) Real-Time: Theory in Practice, LNCS, vol. 600, pp. 45–73. Springer (1992)
5. van Beek, D.A., Reniers, M., Schiffelers, R., Rooda, J.: Foundations of a Compositional Interchange Format for Hybrid Systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC, LNCS, vol. 4416, pp. 587–600. Springer (2007)
6. Berkenkötter, K., Bisanz, S., Hannemann, U., Peleska, J.: The HybridUML profile for UML 2.0. J. on Software Tools for Technology Transfer 8(2), 167–176 (2006)
7. Bingham, B.D., Greenstreet, M.R., Bingham, J.D.: Parameterized verification of deadlock freedom in symmetric cache coherence protocols. In: Formal Methods in Computer-Aided Design (FMCAD 2011). pp. 186–195 (2011)

8. Broman, D., Lee, E.A., Tripakis, S., Törngren, M.: Viewpoints, Formalisms, Languages, and Tools for Cyber-Physical Systems (preprint). In: Proc. of the 6th Intl. Workshop on Multi-Paradigm Modeling (MPM 2012) (2012)
9. Carloni, L.P., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, A.L.: Languages and Tools for Hybrid Systems Design. Foundations and Trends in Electronic Design Automation 1(1), 1–193 (2006)
10. Cellier, F.: Continuous System Modeling. Springer (1991)
11. Clarke, E.M., Wing, J.M.: Formal Methods: State of the Art and Future Directions. ACM Comput. Surv. 28(4), 626–643 (1996)
12. David, R., Alla, H.: On Hybrid Petri Nets. DEDS 11(1-2), 9–40 (2001)
13. Fill, H.G., Redmond, T., Karagiannis, D.: Formalizing Meta Models with FDMM: The ADOxx Case. In: Cordeiro, J., Maciaszek, L., Filipe, J. (eds.) Enterprise Information Systems - 14th International Conference, ICEIS 2012, Wroclaw, Poland, June 28 - July 1, 2012, Revised Selected Papers, LNBIP, vol. 141. Springer (2013)
14. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable Verification of Hybrid Systems. In: G. Gopalakrishnan, S.Q. (ed.) CAV. LNCS, Springer (2011)
15. Giese, H., Henkler, S.: A survey of approaches for the visual model-driven development of next generation software-intensive systems. Journal of Visual Languages & Computing 17(6), 528–550 (2006)
16. Gupta, A.: Formal Hardware Verification Methods: A Survey. In: Kurshan, R. (ed.) Computer-Aided Verification, pp. 5–92. Springer (1993)
17. Harel, D., Kupferman, O., Vardi, M.: On the complexity of verifying concurrent transition systems. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR, LNCS, vol. 1243, pp. 258–272. Springer (1997)
18. Henzinger, T.A.: The Theory of Hybrid Automata. In: LICS. pp. 278–292. IEEE Computer Society Press (1996)
19. Hoare, Charles Antony Richard: Communicating sequential processes, vol. 178. Prentice-hall Englewood Cliffs (1985)
20. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston, MA, USA, 3 edn. (2006)
21. Hu, J., Lygeros, J., Sastry, S.: Towards a Theory of Stochastic Hybrid Systems. In: Lynch, N., Krogh, B. (eds.) Hybrid Systems: Computation and Control, LNCS, vol. 1790, pp. 160–173. Springer (2000)
22. Kern, C., Greenstreet, M.R.: Formal Verification in Hardware Design: A Survey. ACM Trans. Des. Autom. Electron. Syst. 4(2), 123–193 (1999)
23. Kim, K.D., Kumar, P.: Cyber-Physical Systems: A Perspective at the Centennial. Proc. of the IEEE 100(Special Centennial Issue), 1287–1308 (2012)
24. Kreiker, J., Tarlecki, A., Vardi, M.Y., Reinhard Wilhelm: Modeling, Analysis, and Verification - The Formal Methods Manifesto 2010 (Dagstuhl Perspectives Workshop 10482). Dagstuhl Manifestos 1(1), 21–40 (2011)
25. Lamport, L.: Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering 3(2), 125–143 (1977)
26. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Intl. Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
27. Lee, E.: Cyber Physical Systems: Design Challenges. In: 11th IEEE Intl. Sym. on Object Oriented Real-Time Distributed Computing. pp. 363–369 (2008)
28. Meyer, B.: Applying Design by Contract. Computer 25(10), 40–51 (1992)
29. Mitsch, S., Ghorbal, K., Platzer, A.: On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles. In: Robotics: Science and Systems (2013)

30. Mitsch, S., Passmore, G.O., Platzer, A.: Collaborative verification-driven engineering of hybrid systems. Mathematics in Computer Science 8(1), 71–97 (2014)
31. Mitsch, S., Quesel, J.D., Platzer, A.: Refactoring, refinement, and reasoning: A logical characterization for hybrid systems. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM (2014)
32. MoBIES team: HSIF semantics (version 3): Technical Report (2002)
33. Ouimet, M., Lundqvist, K.: Formal Software Verification: Model Checking and Theorem Proving (2007)
34. Pettersson, S., Lennartson, B.: Hybrid Modelling focused on Hybrid Petri Nets. In: 2nd European Workshop on Real-time and Hybrid systems. pp. 303–309 (1995)
35. Pinto, A., Sangiovanni-Vincentelli, A.L., Carloni, L.P., Passerone, R.: Interchange formats for hybrid systems: review and proposal. In: HSCC. vol. 3414, pp. 526–541. Springer (2005)
36. Platzer, A.: Differential Dynamic Logic for Hybrid Systems. J. Automated Reasoning 41(2), 143–189 (2008)
37. Platzer, A.: Differential-algebraic Dynamic Logic for Differential-algebraic Programs. J. Log. Comput. 20(1), 309–352 (2010)
38. Platzer, A.: Logic and Compositional Verification of Hybrid Systems (Invited Tutorial). In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. LNCS, vol. 6806, pp. 28–43. Springer (2011)
39. Platzer, A.: Stochastic differential dynamic logic for stochastic hybrid programs. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE. LNCS, vol. 6803, pp. 431–445. Springer (2011)
40. Platzer, A., Quesel, J.D.: KeYmaera: A Hybrid Theorem Prover for Hybrid Systems. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. LNCS, vol. 5195, pp. 171–178. Springer (2008)
41. Pnueli, A.: The temporal logic of programs. In: Proc. of the 18th Annual Symposium on Foundations of Computer Science. pp. 46–57. SFCS '77, IEEE Computer Society, Washington, DC, USA (1977)
42. Quesel, J.D., Platzer, A.: Playing hybrid games with keymaera. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR. LNCS, vol. 7364, pp. 439–453. Springer (2012)
43. Sanislav, T., Miclea, L.: Cyber-Physical Systems - Concept, Challenges and Research Areas. Journal of Control Engineering and Applied Informatics 14(2) (2012)
44. Schmidt, D.C., Buschmann, F., Henney, K.: Pattern-oriented software architecture. Wiley series in software design patterns, Wiley, Chichester and New York (2000)
45. Shi, J., Wan, J., Yan, H., Suo, H.: A survey of Cyber-Physical Systems. In: Intl. Conf. on Wireless Communications and Signal Processing. pp. 1–6 (2011)
46. Teschl, G.: Ordinary differential equations and dynamical systems, Graduate studies in mathematics, vol. 140. American Mathematical Society (2012)
47. Völzer, H., Varacca, D.: Defining Fairness in Reactive and Concurrent Systems. Journal of the ACM (JACM) 59(3), 13:1–13:37 (2012)
48. Wan, K., Hughes, D., Man, K.L., Krilavicius, T., Zou, S.: Investigation on Composition Mechanisms for Cyber Physical Systems. Intl. Journal of Design, Analysis and Tools for Integrated Circuits and Systems 2(1), 30–40 (2011)
49. Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W., Kapsammer, E.: A survey on UML-based aspect-oriented design modeling. ACM Computing Surveys 43(4), 28:1–28:33 (2011)
50. Zhang, L., She, Z., Ratschan, S., Hermanns, H., Hahn, E.: Safety Verification for Probabilistic Hybrid Systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV, LNCS, vol. 6174, pp. 196–211. Springer (2010)
51. Zwillinger, D.: Handbook of differential equations. ACADEMIC PressINC (1998)