# Decomposition and Refinement in Verification-Driven Engineering of Cyber-Physical Systems

Research Report for the Marshall Plan Scholarship funded stay

at Carnegie Mellon University, Pittsburgh

Andreas Mueller
Johannes Kepler University, Linz
Department of Cooperative Information Systems

February 17, 2014

**Abstract**

This report summarizes the work of Andreas Mueller during his Marshall Plan scholarship funded research stay at Carnegie Mellon University, Pittsburgh. Overall, the addressed research is anchored in the area of *verification-driven engineering* of *cyber-physical systems*. Computers that control physical processes become more and more pervasively embedded into our everyday lives and are called *cyber-physical systems* (CPS). At first—derived from the original proposal—the overall vision created during the research stay is presented. This vision encompasses the development of a verification-driven engineering framework, including verified component-based modeling of CPS, model refactorings and verified CPS model transformations. Before going into further details, we present a conceptual reference model based on a comprehensive literature study, to categorize CPS. After a brief presentation of related work, the research focus on modeling and verification of modular traffic nets and is presented. The concept of a modular and verified traffic network is, to divide a traffic network into small thus simple parts, that can be verified and then connected to form a larger network of blocks. Finally, we conclude with possible future work on verified modular traffic nets and the overall vision.

# Contents

# 1 Introduction

This report summarizes my research conducted during a stay at Carnegie Mellon University (CMU), Pittsburgh. The research stay was funded by a Marshall Plan Scholarship.

## 1.1 Acknowledgments

I would like to thank my advisers Prof. Werner Retschitzegger and Prof. Wieland Schwinger, who support my work and were especially helpful when applying for the Marshall Plan Scholarship. Furthermore, I wand to thank Prof. Andre Platzer, who invited me to join his group at CMU and who supported my work at CMU with valuable feedback and fruitful discussions. Finally, I would like to thank my colleague and friend Dr. Stefan Mitsch, for all the time he spared for questions, feedback, discussion and support, before, after and during my research stay at CMU.

## 1.2 Structure of the Report

Section (Section 2) contains the original proposal, as submitted in advance of my research at CMU. Although, the exact research focus slightly deviates from the originally submitted one, all proposed research goals have been targeted and furthermore extended to form the overall vision presented in Section 3. As a prerequisite for the work with hybrid systems, at CMU we first attempted to formalize the nomenclature of the field of hybrid system verification and tools by means of a conceptual reference model, as presented in section Section 4. Section 5 summarizes relevant related work, we found and studied during our research. Section 6 describes the exact research focus at CMU, and Section 7 presents possible future research directions.

# 2 Original Proposal

## 2.1 Motivation and Problem Statement

Traffic management, emergency rescue, and military operations are domains that naturally exhibit several control systems, operated by different authorities in a distributed way, sharing an environment which is only partially observable, information-intensive and constantly evolving. Current insufficient system support induces huge communication efforts, potential misunderstandings, information overload, and timely pressure on the human system operators. This ultimately endangers their ability to respond to critical situations with serious real-world consequences by either pro-actively preventing them or achieving their correct and timely resolution. To counteract information overload, collaborative situation awareness aims at supporting human operators in assessing current situations

and in predicting possible future ones to take appropriate actions proactively, preventing critical events.

**Prior research.** Recently, several ontology-based situation awareness systems have been proposed, including our prior work in the FFG FIT-IT Semantic Systems project BeAware! [5]. In the course of this project, a qualitative approach to situation prediction has been developed [6]. The follow-up FIT-IT project CSI extends BeAware! to further deal with the issues introduced by the mandatory collaboration between different authorities monitoring a shared environment. CSI supports operators, to gain relevant perspectives on mutually affecting situations and control measures, allowing them to estimate the possible outcomes of situations based on the selected control actions [1][3][4].

**Open research issues.** In this qualitative approach, the effects of control actions of different authorities must be approximated by the system to provide accurate information to the operators and support their decision making process. As a pre-requisite for reasoning about these effects, object and event evolution and the induced behavioral impact of control actions thereupon must be modeled in a fine-grained manner. For instance, the potential impacts of rerouting traffic through an urban area as opposed to accepting an ever growing traffic jam on a highway should be described in detail in terms of domain specific languages, including also potential tradeoffs such as an increased traffic volume in other areas. Such a detailed model would represent the pre-requisite for more expressive predictions of possible future critical situations and appropriate actions, ensuring as ultimate visions, a safe and timely resolution of these critical situations. As a pre-requisite for this, the validity of these actions must be verified, guaranteeing certain invariants (e.g., the control actions must never result in causing an accident).

## 2.2 Synergies With Carnegie Mellon University

Regarding the above mentioned open research issues of fine-grained modeling and verification of control actions in traffic control systems, the expertise on formal methods at Carnegie Mellon University (CMU) and its School of Computer Science (SCS) as one of the world's premier institutions in this area would be highly beneficial. In particular, the focus is on logic, formal verification and automated reasoning about cyber-physical systems, with applications to automotive, train, and air traffic control. Such cyber-physical systems combine continuous world models (e.g., traffic jams growing and shrinking in a continuous manner) with discrete control (e.g., set the speed limit to a certain value) [8]. It has to be further emphasized, that scientific synergies between CMU and the Department of Collaborative Information Systems (CIS) at Johannes Kepler University Linz (JKU) have been already exploited fruitfully in the past, resulting in a successful three-month Marshall Plan Scholarship of DI Dr. Stefan Mitsch in 2011 and a subsequent successful Marie Curie International Outgoing Fellowship and several joint scientific publications. This proposed project,

together with our ongoing research, as detailed below, further exploits the synergies with the CMU by combining the knowledge gathered throughout our research project CSI with the expertise on cyber-physical systems at CMU.

## 2.3 Research Goals and Method

The research method followed is based on the well-known design research approach [2] focusing on three research goals:

1. **Developing a domain specific language (DSL) for traffic network modeling.** In a traffic environment, situations occur at various locations throughout the network (e.g., crossroads or highway exits). These locations, the corresponding situations and available control actions must be modeled in order to allow verification of viable and safe actions. In a first step we plan proposing a DSL to simplify the modeling process.
2. **DSL-driven modeling of reusable traffic building blocks.** Based on the traffic network DSL, recurring parts of a traffic network (e.g., traffic lights, highways) should be modeled, partly sharing a common behavior. These core building blocks can then be reused in order to develop models of more sophisticated traffic situations.
3. **Verification of traffic models.** Finally, the validity of the composed traffic models has to be verified, in order to ensure safe resolution of critical situations. For this, the modeled continuous behavior and the control actions will be transformed into differential dynamic logic [7], and complemented with a model of system invariants, which altogether are verifiable by the KeYmaera prover developed by the partner institution at CMU.

## 2.4 Background

Overall, the addressed research is anchored in the area of *verification-driven engineering* of *cyber-physical systems*. Computers that control physical processes become more and more pervasively embedded into our everyday lives and are called *cyber-physical systems* (CPS). The importance and impact of expertise in this field was highlighted in a recent study on CPS engineering [22]. CPS—from an engineering point of view—can be described in terms of *hybrid systems*. Such hybrid systems comprise discrete control decisions (the cyber-part, e. g., setting the acceleration of a car) and differential equations modeling the entailed physical continuous dynamics (the physical part, e. g., motion of the car gaining speed) and are often modeled as *hybrid automata* [27]. According to [47], the key challenge in engineering hybrid systems is the question of how to ensure their correct functioning in order to avoid incorrect control decisions w.r.t. safety requirements. This is especially important, as many CPS operate in safety-critical environments and their malfunctioning could entail severe conse-

quences. Whereas testing can only give an indication about safety, only formal verification can provide a proof of correctness. Techniques from the field of *model-driven engineering* (MDE), used to incrementally develop systems in a well-defined and traceable manner, together with formal verification methods seem to be very well fitted, to ensure safety of CPS. These techniques form the vision of *verification-driven engineering* (VDE) for CPS, which has recently emerged as a new paradigm and is currently target of intense ongoing research (e. g., [36, 47]).

## 2.5 Application Scenario

Examples of CPS where VDE is crucial can be found in various application domains, such as electricity (i. e., smart grids), health care, road-, rail- and air traffic. The application scenario for VDE targeted in this report is the domain of road traffic management. This is since we are currently pursuing the FFG FIT-IT project CSI[1], aiming at supporting traffic operators in perceiving critical situations on Austrian roads and reacting in an appropriate manner, where safety is key, in order to prevent accidents and traffic injuries.

In road traffic control, models can describe a microscopic level of traffic (e. g., a single autonomous cars, as mentioned above), or a macroscopic level of traffic (e. g., the traffic flow as a whole) [26]. This way not only single cars, but whole traffic control systems can be modeled as CPS using flow models to describe the continuous dynamics (i. e., the continuous flow of cars along a road) and traffic operators making discrete decisions in the form of actions (e. g., set detour, change speed limit) [46]. In the context of our current project CSI[2] at JKU, we identified that both views (micro- and macroscopic) provide a major benefit for the traffic operators.

In CSI, to address this, we currently employ MDE techniques to model traffic *objects* (e. g., a "traffic jam" object) and to aggregate them into critical *situations* (e. g., a "traffic jam" object, close to a "railroad crossing" object, results in a critical "jam close to a railroad" situation). After identifying these situations, the selection of appropriate *actions* that can be taken in order to resolve these critical situations (e. g., reroute traffic in order to avoid traffic jams on railway crossings) is a crucial task. However, for now, there is little or no support for the traffic operators, to verify the safeness of their control decisions.

# 3 Overall Vision

Based on the original proposal, through discussions with colleagues at CMU and resting upon a comprehensive literature study, we developed

[1] `http://cis.jku.at/index.php/projects/csi`

[2] Work 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.

an overall vision to support *verified control systems with verified control actions*, satisfying certain safety conditions. In detail, the aim is to achieve this through verification-driven engineering comprising three concrete research goals backed up by the experiences gained through our research project CSI:

**Component-based Modeling Framework.** For achieving a verified control system with verified control actions, the different situations and actions, the control part, as well as the effects of the action an the situation, the physical part, need to be formalized in form of a model such that verification can take place. It is a well-understood problem that verification becomes more complicated and complex as the size of the model increases. This is aggravated by the fact that a plethora of different situations and actions are encountered in control systems as the one focused on. Despite the size and differences of the various situations and actions required, they however share recurring building blocks like semaphore crossings, on/off-ramps, etc., each sharing common behavior with respect to influencing the safety conditions. One research goal for to archive the overall vision therefore is to come forward with a component-based modeling framework for CPS with allows to decompose the control system and model verified components and compose verified larger models (i.e. traffic models), that such allow to predict possible outcomes with respect to their safety requirements.

**Refactorings for Hybrid System Models.** As the controlled environment changes over time, so do the requirements on the CPS (e. g., adding lanes to roads). This naturally implies that CPS themselves, the overall model, as well as the components they are composed of, need to be iteratively modified as well. Furthermore, also safety conditions need to be reassured. Consequently, another research goal is to reduce complete re-verification in case of modification on hybrid system models through dedicated refactorings that maintain guaranteed safety conditions. This furthermore also splendidly benefits the iterative process of CPS development and the production of variants of verified components (e. g., two-, three-lane highway).

**Verified Model Transformation.** Ultimately, the modeled and verified control decisions need to be put into actual operation through appropriate control code in the system. However, incorrect implementation of those may jeopardize and corrupt the safety conditions and thus lead to malfunctions and entail severe consequences. To close the development loop, thus, the final research goal is to come up with verified model transformations to bridge the development gap between the models and the code and thus ensure to produce correct executable code.

Altogether, this targets the vision of a comprehensive *verification-driven engineering framework* which allows to appropriately model, evolve and implement verified CPS.

# 4 The Conceptual Reference Model

As a first step towards development of a verification-driven engineering framework, we took a closer look at hybrid systems and proposed a conceptual reference model for the field.

## 4.1 Introduction

**Cyber-Physical and Hybrid Systems.** Computing devices that directly interface and interact with their real-world surroundings by means of their sensors and actuators are commonly known as *cyber-physical systems* (CPS). CPS leverage the construction of increasingly autonomous systems (e. g., autonomous cars). Consequently, significant demands are imposed on the *safety* of such a CPS. 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. In order to specify (i. e., model) a CPS and verify the desired behavior, its complex real-world interactions need to be considered as well, which introduces unprecedented complexity into the verification problem. The behavior of a CPS can be described by means of a *hybrid system model*, which allows to simultaneously specify both the continuously evolving real-world interactions and the discrete control decisions of the CPS within one model.

**Diversity of Modeling and Verification Tools.** Users face diverse modeling and verification tools, which employ a wide range of modeling formalisms (e. g., hybrid automata, hybrid programs), aim at distinct verification goals (e. g., safety, liveness) and incorporate heterogeneous verification techniques (e. g., theorem proving, reachability analysis). Often, the use of multiple tools in combination is beneficial because their capabilities differ strongly. The downside of this diversity are *compatibility* and *model exchange* issues. An approach of dealing with this problem is the introduction of *hybrid systems interchange formats*, which tackle the problem of model exchange between otherwise syntactically incompatible tools. Naturally, these interchange formats support a much larger range of concepts than any single tool. This allows translation of almost any modeling formalism to the interchange format. However, translation back might entail issues, if the formats connected by the interchange format do not share all concepts. Team verification efforts are even more aggravated, since interchange formats do not yet focus on exchanging verification results and supporting refactorings, components and other practices that are common in other engineering disciplines.

**Contributions.** To fill this gap, we introduce a *conceptual reference model* (CRM) of hybrid system modeling and verification tools, with a focus on concepts currently neglected by interchange formats. We identify relevant core parts, the interrelations and dependencies between them and unify the terminology. Furthermore, we illustrate how the CRM can assist in the use of hybrid system interchange formats by classifying the capabilities of modeling formalisms and tools. Additionally, we depict future

extensions and enhancements for hybrid systems interchange formats on the one hand, and for modeling and verification tools on the other hand.

## 4.2 Related Work

To the best of our knowledge, no other CRM depicting hybrid system modeling and verification tools has been proposed in literature so far. Nevertheless, several surveys on CPS and hybrid system modeling and verification, partly aiming at providing some classification, have been conducted, which will be discussed and compared to our work below.

**Frameworks for Categorizing Hybrid Systems..** Broman et al. [10] provide a coarse-grained model for categorizing hybrid systems. Their framework comprises *Viewpoints* (depend on stakeholders and their concerns), *Formalisms* (represent formalisms which are useful in modeling hybrid systems) and *Languages and Tools* (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 stakeholders' requirements, while we focus on the engineering aspects of hybrid systems.

**Surveys On Verification of Hybrid Systems..** Overall, only a few surveys actually deal with the verification of hybrid systems: Alur [2] reviewed formal verification approaches without focusing on modeling and tool support. Carloni et al. [11] analyzed the syntax and semantics of various languages and tools for hybrid system modeling with respect to verification and simulation, but in contrast to our work do neither extensively discuss modeling nor general tool support.

**Systematic Modeling and Specification..** A large body of research addresses solely the systematic modeling and specification of hybrid systems, but does not address verification: Giese et al. [23] survey visual model-driven development of software-intensive systems. Shi et al. [61] provide a short overview and further research challenges of CPS. Sanislav et al. [60] focus their work on challenges, concepts and research goals in the area of CPS. Wan et al. [65] investigate the applicability of different composition mechanisms for cyber-physical applications. Kim et al. [34] provide a broad overview of CPS research from a historical point of view and Lee [41] examines the challenges in designing CPS.

In summary, this paper addresses CPS modeling and verification concepts as follows.

- Unlike [10], who provide a coarse-grained framework relating viewpoints, formalisms, languages and tools, we focus on engineering aspects of hybrid systems as a basis for our CRM.
- Unlike [2], who briefly review selected hybrid system verification approaches, and [11], who survey syntax and semantics of hybrid system verification and simulation tools, we extensively consider aspects of modeling, verification and tool support alike in our CRM.

- Unlike [23, 34, 41, 60, 61, 65], who focus on modeling and specification of hybrid systems, we include verification and tool support into our CRM.

## 4.3 Hybrid Systems Interchange

In this section we discuss *hybrid systems interchange formats* in order to focus our CRM on important concepts, which are not sufficiently detailed in current interchange formats. Hybrid system interchange formats are an approach to exchange models between different tools for hybrid system modeling, as they tackle the problem of model exchange between otherwise syntactically incompatible tools. The goal is to provide a common intermediate format used by various (ideally all) available tools to simplify the exchange of models between them. Instead of having to provide a translator for each pair of formalisms, it suffices to provide a single translator per tool to and from the common interchange format. Due to the variety of relatively novel and immature tools such an interchange format is a way to allow mixing and matching of different tools optimized for different tasks [52] (e. g., KeYmaera for formal verification [57], SpaceEx for reachability analysis [20], Modelica for simulation [21]).

Over the last few years, several different such interchange formats have been proposed, each with a slightly different approach. Example interchange formats are the *Hybrid System Interchange Format* (HSIF) [48], the *Compositional Interchange Format* (CIF) [6], and the Metropolis-based interchange format [53]. The formats support hybrid system model exchange as follows: HSIF, defined as part of the DARPA MoBIES project, uses a network of hybrid automata for model representation. CIF aims at being more general than the HSIF by not incorporating tool limitations and avoiding further restrictions that were imposed by HSIF [6]. The Metropolis-based interchange format is designed to complement HSIF and other formats with an abstract semantics for hybrid systems.

These interchange formats share several requirements, including (i) support for all existing tools, modeling approaches and languages [53], (ii) support of a wide range of concepts [6], and (iii) support of hierarchy, modularity and object orientation [6]. These requirements and the pure focus on model exchange result in several limitations that do not yet sufficiently support verification teams, as discussed below.

## 4.4 Exchange of Models

**Wide range of modeling concepts.** Since an interchange format must support numerous available tools and formalisms, a support for a wide range of modeling concepts is inevitable. Typically, interchange formats provide the union of concepts found in different formalisms. As a result, only the interchange format supports the whole range of modeling concepts, while the different tools support only a subset thereof.

Consider, for example, a model for the hybrid theorem prover KeYmaera [57], which supports hybrid programs and differential dynamic logic (d$\mathcal{L}$). Among others, d$\mathcal{L}$ supports non-linear differential equations, which could in principle be translated, for instance, to CIF, because CIF supports arbitrary differential equations (basically any well-formed formula). It should then be possible to transform the resulting interchange format model into any other format, given a respective translator. However, transforming the same model to any hybrid systems tool only supporting linear DEs (e. g., SpaceEx) will fail. This semantic gap remains unnoticed until the transformation is executed, because current interchange formats lack a detailed classification of the unified concepts.

Nadales et al. [49] put the burden on the modeler to identify the common subset of two formalisms that can be used for transformation. The Metropolis-based interchange format identifies this issue and proposes to define the concrete semantics between any two tools [52]. We propose a CRM with concepts to detail the concrete semantics of hybrid systems, i. e., discrete behavior, continuous dynamics and specification formalisms. If used as a type system, such a CRM could compare different kinds of hybrid system modeling concepts (e. g., kinds of differential equations) and help to identify a common subset between different formalisms.

**Support for all existing tools, modeling approaches and languages.** The support of most available tools, modeling approaches and languages is crucial for the versatility of an interchange format. Even though (composite) hybrid automata are a widely spread formalism among hybrid system tools and languages, and are furthermore used by many interchange formats, several other formalisms (e. g., hybrid programs) exist. Although those formalisms might be transformable to hybrid automata, useful characteristics that are helpful for verification (e. g., program structure) are likely to be lost.

For example, KeYmaera uses hybrid programs as a modeling formalism, whereas SpaceEx uses hybrid automata. Although hybrid automata and hybrid programs can be mutually encoded without loss of expressivity [54], the conversion of an automaton to a program usually results in awkward program structure with additional variables and tests just to identify the different states of the automaton in the program (or a huge number of states in the opposite direction). Such programs are often harder to verify than a well-formed equivalent hybrid program with proper structure. As KeYmaera uses quantifier elimination for verification [57], which is doubly exponential in the number of variables [17], the resulting model can be unnecessarily hard to prove. Often, more natural and directly written programs are more useful.

We propose a CRM to help identifying the various modeling formalisms used by different tools and, for example, be used to recognize compatible ones (e. g., hybrid automata are compatible with timed automata).

**Composition by hierarchy, modularity, and object-orientation.** Compositional models are supported by most of the interchange formats, as well as by many formalisms used in tools and languages. However, most ver-

ification tools only support specific compositional concepts (e. g., parallelism, synchronization, or urgency).

For example, most formalisms do not support specifying urgency and synchronization constraints, which leads to models that cannot be translated directly [49]. In another example, SpaceEx bases on hybrid automata, which support parallel composition, whereas KeYmaera uses hybrid programs, which do not yet directly support parallel composition.

We propose to anchor support for hierarchical models in the CRM. Compositionality and subsequent concepts (e. g., synchronization, urgency or parallelism) describe different ways of combining hierarchical models and must be included in the CRM. Although the concepts vary between the formalisms, a lossless translation might be possible, as long as some kind of compositionality is revealed by the CRM. For example, *urgent actions* (i. e., actions that must execute if possible) can be implemented by means of the *time can progress predicates* (i. e., time can only move forward in a state, if the predicate holds), by not allowing the system to progress in a state, if the execution of an action that should be urgent is possible.

After discussing limitations of current interchange formats with respect to model exchange, in the following we illustrate requirements on verification result exchange and tool support that we deem highly necessary for verification teams.

### 4.4.1 Exchange of Results

Team-based verification environments, such as proposed in [47], not only require model exchange but also exchange of verification results. However, different verification tools implement different kinds of methods and algorithms for analyzing a model, which makes them often useful in different scenarios. While one tool reaches an impasse, others might be able to perform further steps. The CRM includes concepts to represent verification results for their inclusion into future exchange formats.

A concrete scenario would be a hybrid theorem prover (e. g., KeYmaera), that reaches a proof step where quantifier elimination does not finish in reasonable time because of a large number of variables. Provided with an interchange format for verification results, the tool could exchange the partial proof with another tool, that is able to further analyze the current status. This analysis might reveal inherent properties of this particular branch, that allow removing some of the variables in KeYmaera, which in turn leads to an easier quantifier elimination and thus allows closing of the branch. Of course this works in both directions, as a SpaceEx model could also be translated to a KeYmaera hybrid program to get results using KeYmaera and then apply SpaceEx reachability analysis to this results, as suggested in [9].

### 4.4.2 Tool support

Since research on modeling and verification of hybrid systems is a rather young and active research field, we outline tool support concepts for future tools in the CRM.

**Components.** A modeling tool should support users in specifying the discrete control logic as well as the continuous dynamics of a model. A tool, for instance, could include a library of predefined building blocks with classical continuous dynamic phenomena (e. g., Newtonian rigid body dynamics). Furthermore, as it is good engineering practice to reuse modeling components, a tool should support reuse of previously designed models. Such tools reduce the effort of comprehensive manual specifications as well as the risk of errors due to improper specification of these phenomena.

**Refactoring.** Refactoring was established as a best-practice procedure to systematically improve existing artifacts, such as models. Classical refactoring as known from programming is defined by Fowler [19] as a modification technique that improves the internal structure of a model but does not change the observed external behavior. When it comes to refactoring of CPS in general the question of what is considered external behavior arises [44].

In the CRM, we will propose a classification scheme for refactoring techniques that makes it possible to distinguish refactoring operations according to their effect on model structure, model behavior, and verification complexity.

**Artifacts.** To support good engineering practice, such as division of labor by separating a modeling and verification task into distributable units of work, an interchange format should provide means for relating different artifacts to each other. If models and proofs have been split into parts, verified, exchanged and merged, an interchange format should allow relating previously verified (partial) proofs to different model versions. For example, a used proof rule could be related to a proof step to discover suitable rules for similar proof steps later on (e. g., if a model was changed by a refactoring, parts of the proof might still be applicable in the refined version).

In the previous section, we revealed issues arising with the use of several hybrid system interchange formats and their current limitations regarding to our vision of verification teams. In the next section, we introduce a conceptual reference model, which provides classifications for the aforementioned requirements and can be used to enhance interchange formats.

## 4.5 Conceptual Reference Model

Although in detail current hybrid systems interchange formats have several limitations as highlighted above, overall they are extensive with support for many tools. Hence, to improve on these limitations we propose

a conceptual reference model, which not only can be used to enhance current interchange formats, but may also serve as a basis for surveys of hybrid systems modeling and verification tools. A major difficulty for systematically analyzing interchange formats, modeling and verification tools for hybrid systems is a lack of a common classification of the basic ingredients of *hybrid system modeling* and *hybrid system verification*. Although the term "verification" means to ensure the behavior of a system as intended by its constructor [37], it is used for a range of different techniques.

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., [66]). The CRM we present in this paper (cf. Fig. 2) comprises the modeling and verification of hybrid systems and the corresponding tool support. We expose the basic components of hybrid systems and the interrelations between them. We express the CRM as Unified Modeling Language (UML) classes, since UML is the prevailing standard in object-oriented modeling[3]. 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.

The rationale behind constructing the CRM is as follows: In principle we followed a top-down approach, meaning that several concepts of the CRM have been adopted from existing other surveys in this area as referred to in Section 4.2. We supplemented our CRM in a bottom-up way with complementary 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.

### 4.5.1 Systems

The classes in the Systems package describe a high-level systems being perspective to anchor modeling and verification tools, which, according to Klir [35], can be classified diversely. We follow Teschl [63], and distinguish `DynamicalSystems` into `DiscreteDynamicalSystems` (state space $\mathbb{N}/\mathbb{Z}$) and `ContinuousDynamicalSystems` (state space $\mathbb{R}$); systems that have both characteristics are `HybridSystems` [27], 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 on a high-level.

[3]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/`.

### 4.5.2 Modeling

**Model.** A dynamic system is described by a `Model` capturing its relevant features. A model is expressed in a `ModelingFormalism` and can be constrained by arbitrary `Conditions`.
**Verification Specification.** A models expected behavior is described by a `VerificationSpecification` utilizing a `SpecificationFormalism` [14, 24]. A verification specification 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 one of its executions). Furthermore, it is often possible to annotate models with `Hints/Strategies`, that support a verification tool in its workings, but do not directly influence the behavior of a model (e. g., inductive *invariants* holding throughout system execution).
**Formalism.** A major part of the Modeling package is the `Formalism` subpackage, 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` [24] for modeling of discrete systems, `Differential Equation` for modeling of continuous systems [12], as well as `HybridAutomata/HybridPrograms` as their combination [27, 54]. 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 [45], 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 the main `Characteristic` class in the CRM to handle `Compositionality` (support of compositional models through, e. g., `Parallelism`, `Urgency`, `Synchronization` or *sequences*) and `Non-Determinism` (e. g., support of *non-deterministic choices*).
**Automaton and Program.** An automaton comprises a set of `States` and a set of `Transitions`, and can be visualized as a directed graph [28]. 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.** There are many different types of differential equations (DE) and also multiple ways to classify them [68]. For our CRM we chose a classification into `PartialDE` (PDE) and `OrdinaryDE` (ODE). Both can have special restricted linear forms (i. e., `LinearPDE`, `LinearODE`). In accordance with [55], we allow conditions to restrict a differential equation to remain within a particular region, transforming differential equations into differential-algebraic equations. Differential-algebraic equations can further be equivalently transformed into differential inequalities, which is useful to express disturbance in continuous dynamics.
**HybridAutomaton and HybridProgram.** The generic concept of hybrid

automata is the basis for numerous hybrid formalisms with different levels of expressiveness and detail, such as *HybridUML* [7] or *Hybrid Petri Nets* [51, 18]. For hybrid automata, we introduce a `ContinuousState` that references a set of differential equations. These differential equations represent the continuous behavior of a system while their respective state is active. `TimedAutomata`, which are a prominent, special kind of hybrid automata are also used for modeling of (simple) hybrid systems [4]. Like above, we introduce *Hybrid Programs* [54, 56] as sub-class of programs which allows differential equations as instruction. As already mentioned, although hybrid automata can be encoded as hybrid programs [54], 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` [24], such as Temporal Logic (TL) [58] or differential dynamic logic (d$\mathcal{L}$) [54]. These logics differ in terms of capabilities and expressiveness, and support various kinds of quantifiers and modalities (e. g., [] or <> modalities in d$\mathcal{L}$, A or E path quantifiers in CTL). Logics currently included in the CRM are `CTL`, `LTL`, their common superset `CTL*`, `dL` and `TCTL`.

### 4.5.3 Verification

**Verification Goal.** As already mentioned, most definitions of *verification* are concerned with the behavior of a system that was intended by its creator [37]. Kern et al. [33] conceptually distinguish between *property-* and *implementation verification*. Property verification is concerned with specifying properties that are desired for a design (equivalent to what we term as *verification* in our CRM), while implementation verification deals with the relationship between high-level models and the implementation. In the CRM we focus on property verification. For ensuring such an intended behavior, several verification methods like *Model Checking* or *Verification by Deduction* [37] exist. These and similar methods prove that a specification, which aims at a particular `VerificationGoal`, is correct. Common verification goals include `Safety` (i. e., something will *not* happen) or `Liveness` (i. e., that something *must* happen) [39], `Controllability`, `Reactivity` [56], `Fairness` [64] or `Deadlock Freedom` [8].

**Verification Result.** During the process of verifying the correctness of a model w.r.t. a verification specification, various `VerificationResults` can be produced. Regardless of the employed technique, verifiers usually attempt to reveal certain witnesses of the correctness of a model (e. g., the set of reachable states or a formal proof).

**Proof.** A `Proof` consists of arbitrary many `ProofSteps` and aims at verifying that the model is correct w.r.t. the specification. Initially, all the steps are `OpenProofSteps` until they can be verified using certain `ProofRules`, making them `ClosedProofSteps`. Ideally, a proof is a `SuccessfulProof`, meaning that it consists of closed proof steps only. If some steps remain unverified (i. e., open proof steps), the proof is a `PartialProof`. Note that a partial proof does not necessarily imply refutation of the specification, as it might still be verified with a different proof attempt.

Proofs, respectively the remaining open proof steps, might be decomposed into separately treatable sub-proofs. The resulting proof branches can have the same formalism (e. g., split a proof of a hybrid system along the system's branches into multiple proofs) or even different formalisms (e. g., ship off the arithmetic leaf of a hybrid proof).
Entire proofs or the resulting proof parts might be transferred to other users using the same tool or even other tools. There, some open proof steps might be closed or the entire proof might be finished and returned to the sender. An implementation of the CRM should support the compositions/decomposition of proofs, as well as the exchange of proofs and partial proofs, and furthermore deal with problems arising from this exchange, like verification of correctness of exchanged proofs (e. g., through 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.

### 4.5.4 Tool Support

**Tool.** The ToolSupport package includes `Tools` and additional information regarding usage and configuration. Generically speaking, a tool (e. g., a specification tool) works with various `Artifacts`; it uses some input (e. g., a model) to produce a corresponding output (e. g., a verification specification). Tools might require `UserInput` at run-time (e. g., user-guidance during verification), and additional prior `Configuration` (i. e., meta information required to run the tool, e. g., library paths). 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). Our CRM includes three kinds of 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.
**Refactorings.** Artifacts can influence an arbitrary number of other artifacts (e. g., a discovered counter example might be a valuable input for a revision of the initial model). These mutual influencing leads to an evolution of artifacts prevalent in modeling and verification of hybrid systems. Hybrid systems are typically developed in an iterative manner by model refinement, i. e., initially coarse-grained models are iteratively revised and detailed. `Refactorings` simplify this refinement by providing support for restructuring the different artifacts.

To support re-engineering and iterative development of arbitrary artifacts, we distinguish between `Behavior-` and `Specification Preserving` refactorings. `Behavior Preserving` refactorings are classical refactorings as known from programming and defined by Fowler [19] as a modification that improves the internal structure of a model but does not change the observed external behavior. When it comes to refactoring of models in general, things — especially regarding the behavior preservation property — become more difficult [44]. The common definition of "behavior" is a crucial first step prior to defining a refactoring that does not change this behavior. We propose to use the specifications, which are needed for formal verification anyway, as an indicator of preserving behavior: a refactoring preserves behavior if the refactored model still provably satisfies the same correctness properties as the original one. But the complexity of verification can differ vigorously depending on the selected models (e. g., concurrent transition systems are exponentially harder to verify than sequential ones [25]). Specification preserving refactorings are intended to minimize verification effort (e. g., components that are modeled in a parallel manner in reality, but are independent in their read-write variables, can be executed sequentially in arbitrary order to reduce verification complexity) or to improve the model's semantics (e. g., introduction of sensor uncertainty).

As a model is always an abstraction of a real-world system, it is only a more or less detailed approximation of reality. The complexity of verification can differ vigorously depending on the selected models (e. g., concurrent transition systems are exponentially harder to verify than sequential ones [25]). We therefore distinguish specification preserving refactorings into those that improve the semantics of the model at hand (i. e., `Refinement Refactorings`) and those that decrease the semantics of the model (i. e., `Abstraction Refactorings`). The former is useful if a simple model should be enhanced to become more realistic, while the latter simplifies a model that is too complex for verification.

**Verification Tools.** In our CRM we deduce two popular types of verification tools from two important verification techniques, being *formal verification* and *model checking* [50].

Concerning formal verification, we introduce the class `DeductiveVerificationTool`. These tools are based on a certain logic and use inference rules (i. e., proof rules) to transform formulas until they can be verified. Kern et al. [33] refer to these techniques as *Deductive Methods*, while Clarke et al. [14] 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. It can be roughly distinguished between exact verification and approximation (i. e., how far away is the result from the actual behavior). Collaborating tools should exchange information about the trustworthiness of a result.

## 4.6 Micro Survey

In this section we use the CRM for a short survey of hybrid systems verification tools connected through a hybrid system interchange format.

### 4.6.1 Comparison Criteria

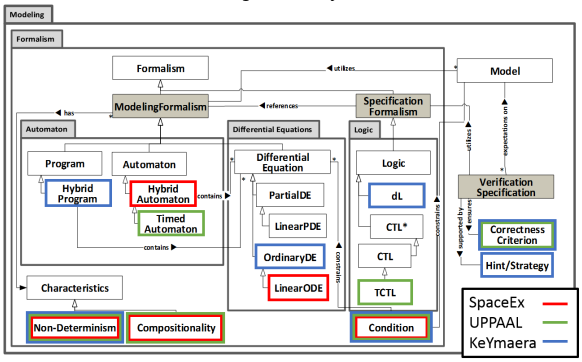
In order to simplify comparison of verification and modeling tools with the help of our CRM, we deduce a set of criteria that allow to classify and evaluate hybrid systems modeling and verification tools. We focus on the modeling package as it contains the main aspects covered by currently available interchange formats and collect the criteria into three groups, namely *Modeling Formalism*, *Specification Formalism* and *Verification Formalism*. For modeling formalisms, we further distinguish the *discrete behavior*, the *continuous dynamics* and their *characteristics*. For verification specifications, we analyze the types of *conditions*, *correctness criteria* and *strategies/hints* supported by the tools. Detailed criteria are analogous to the concepts of the CRM.

### 4.6.2 Lessons Learned

The three groups of criteria introduced above already reveal serious differences between the hybrid systems modeling and verification tools that cannot be overcome by current interchange formats as discussed below. We choose *CIF* as interchange format, since there already exist translators between CIF and various popular hybrid systems tools (e. g., SpaceEx [20], Modelica [21]), including tools for hybrid system verification. Furthermore, CIF is actively developed and its current version (CIF3 r6117) was released on September 20, 2013. As exemplary tools, we choose SpaceEx [20], UPPAAL [40], and KeYmaera [57], which are well known hybrid system verification tools. SpaceEx and UPPAAL are both supported by CIF, whereas KeYmaera is not yet connected to any hybrid systems interchange format. Fig. 1a compares the tools according to criteria derived from the CRM (which would not be possible with only interchange formats at hand). The tools have different capabilities, as SpaceEx is a verification platform 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

| | | SpaceEx | UPPAAL | KeYmaera |
|---|---|---|---|---|
| **Modeling Formalism** | **Discrete Behavior** | Networks of Hybrid Automata | Networks of Timed Automata | Hybrid Programs |
| | **Continuous Dynamics** | Affine dynamics | Synchronous clocks | Non-linear ODE, Algebraic DE, Differential inequalities |
| | **Character-istics** | Synchronization, Parallelism, Non-determinism | Synchronization, Parallelism, Urgency, Non-determinism | Non-determinism |
| **Specification Formalism** | **Logic** | n.a. (Reachability Analysis) | Subset of TCTL | dL |
| **Verification Specification** | **Conditions** | Guards, Invariants | Guards, Invariants | Guards, Invariants |
| | **Correctness Criteria** | n.a. (Reachable States) | Path Formulae (reachability, safety, liveness) | Arbitrary conditions (safety, liveness, ...) |
| | **Hints/ Strategies** | - | - | Annotations (variants, invariants, ...) |

(a) Comparison by criteria




(b) Comparison in the CRM (snippet)

Figure 1: Comparison of SpaceEx, UPPAAL and KeYmaera.

prover for hybrid systems. Fig. 1b highlights the differences in the CRM accordingly.

**Modeling Formalism..** Each of the tools uses a slightly different modeling formalism. SpaceEx and UPPAAL both work with networks of automata, where SpaceEx accepts hybrid automata and UPPAAL is restricted to timed automata. KeYmaera chooses a different approach and uses hybrid programs. 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 dynamics while KeYmaera can handle all kinds of non-linear ODE, as well as algebraic DEs and differential inequalities. Further characteristics show, that SpaceEx and UPPAAL—as they use networks of automata—support compositionality (e. g., by parallel composition). All of the tools support non-determinism in some way.

From looking at the CRM it becomes obvious, that timed automata are a subclass of hybrid automata thus, allowing conversion of timed- to

hybrid automata, but not necessarily the other way around. 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.

The example shows, that naive conversion via an interchange format between SpaceEx and UPPAAL is not sufficient. Instead, a detailed look into the capabilities of the target formats is required. Our CRM provides details for key aspects of hybrid systems that are not yet included in interchange formats. Additionally, the CRM can be implemented as an object-oriented *type-system* to provide automated support for model selection and compatibility checks. For example, the *modeling formalism* for UPPAAL is timed automaton while it is hybrid automaton for SpaceEx, and hybrid program for KeYmaera. Consider a user having a concrete model using a timed automaton as modeling formalism and requesting the system to return all tools capable of analyzing this model. The system would look up all tools, that can handle timed automata, i. e., those models whose modeling formalism allows assignment of a timed automaton. Eventually, the system would return UPPAAL as well as SpaceEx, as timed automata can be assigned to both classes' modeling formalism, since hybrid automaton is a super-class of timed automaton. When comparing SpaceEx and UPPAAL, the type-system would also know, that an instance of a SpaceEx model (containing a hybrid automaton) cannot necessarily be assigned to UPPAAL, as the modeling formalism of UPPAAL cannot handle hybrid automata.

As KeYmaera uses hybrid programs which are not directly compatible, it would not be considered. Nevertheless, the notion of hybrid programs, can be translated to hybrid automata and vice-versa. However, the resulting models might become very unhandy resulting in verification issues. In general, KeYmaera is very different compared to UPPAAL and SpaceEx. For example, no other tool supports non-linear ODE as used by KeYmaera.

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

Similar as with modeling formalisms, 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 $\mathrm{d}\mathcal{L}$. As neither of these two logics is a subclass of the other in the CRM, a direct conversion in either direction is not possible, but a translation for equivalent statements has to be implemented.

**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 dℒ fomulae to specify any kind of property (e. g., safety and liveness). KeYmaera furthermore allows annotating its models with further conditions (e. g., variants and invariants), to support the tool during verification of the models.

On the one hand, the CRM supports the comparison of the available kinds of correctness criteria specifiable within the different tools. When translating a SpaceEx model to UPPAAL, correctness criteria must be specified in UPPAAL after the translation, that can be verified using the target tool. When further translating the UPPAAL model to KeYmaera, the CRM furthermore reveals, that an enrichment of the model by means of adding annotations might be necessary to successfully verify the provided model in the target tool.

## 4.7 Conclusion and Future Work

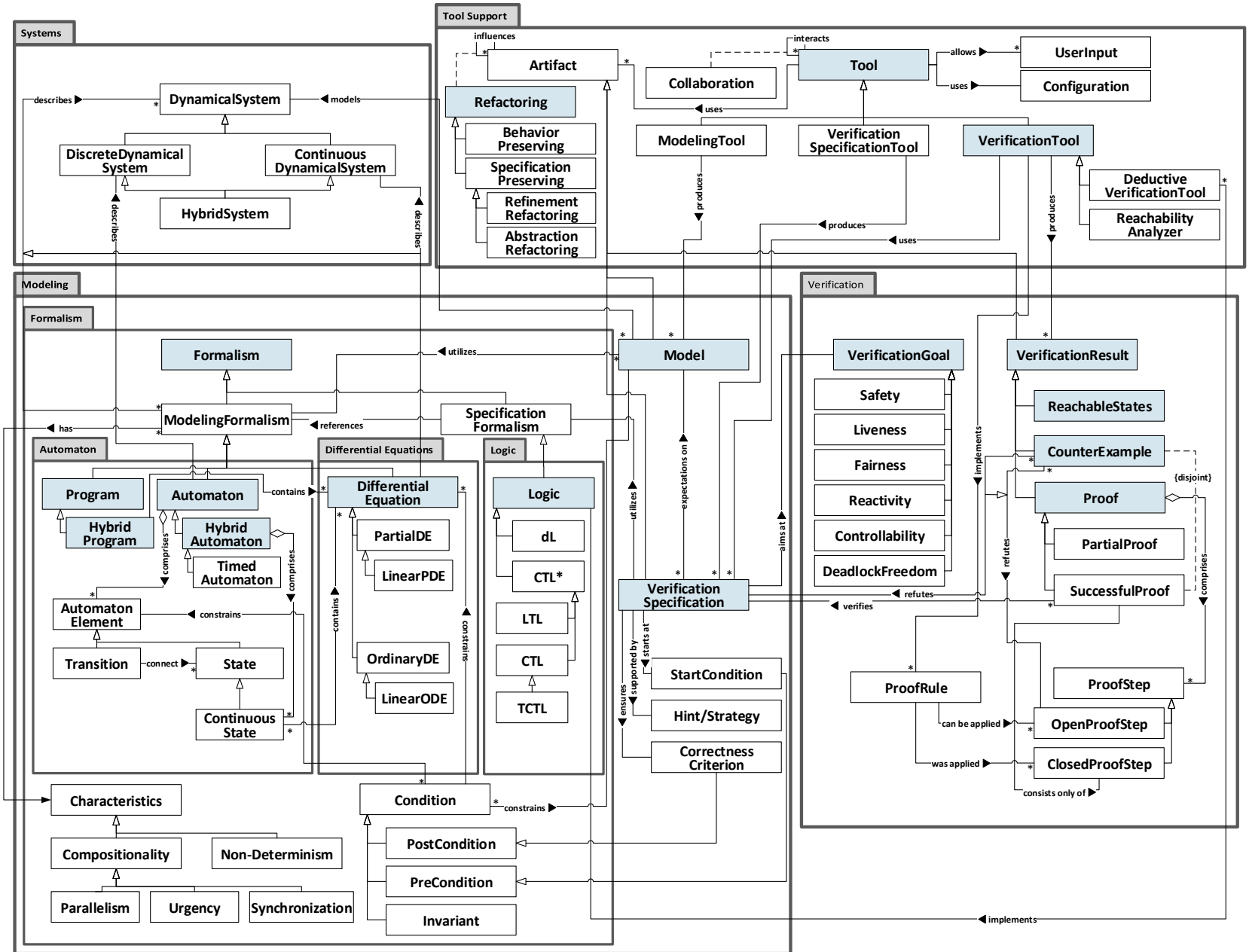
In this paper, we introduced a conceptual reference model that can be used to analyze the properties of hybrid system modeling and verification tools and classify them accordingly. We motivated the need for a reference model based on the shortcomings of current hybrid system interchange formats and used the resulting CRM to (i) compare the capabilities of formalisms supported by hybrid systems interchange formats, (ii) propose extension of these formats to include interchange of results, and (iii) highlight possible future directions for hybrid systems modeling and verification tools.

However, our CRM represents only a first step towards a complete classification framework for hybrid system modeling and verification tools and can be extended with further details in order to allow a comprehensive analysis of verification and modeling tools. Nevertheless, we are sure, that a CRM is an important step towards unification of terms and towards creating a common understanding of the parts of cyber-physical systems and hybrid system models.

For future work, the CRM must be extended in various directions to keep track with the quickly evolving world of hybrid systems engineering. A possible extension is the incorporation of *stochastic* or *probabilistic hybrid systems*. Different researchers have proposed various models and approaches for safety verification and reachability analysis of stochastic hybrid systems (e. g., [1, 5, 29, 67]), that differ most in where to introduce randomness [29]. For most approaches, the discrete states remain untouched, while the jumps between them are governed by probabilistic laws, thus extending the notion of hybrid automata resulting in the introduction of a new kind of edge in the CRM. Another way of introducing randomness is the introduction of stochastic differential equations, leading to a new kind of differential equation in the CRM. However, to fully integrate the concept of stochastic hybrid systems into the CRM, further

considerations regarding additionally required classes are neccessary.

Finally, we plan to conduct a comprehensive survey of hybrid system modeling and verification approaches based on the CRM, respectively on a sophisticated and extensive criteria catalogue derived from it.


Systems
DynamicalSystem
describes
models
DiscreteDynamical System
Continuous DynamicalSystem
HybridSystem
describes
describes
Tool Support
influences
Artifact
Refactoring
Behavior Preserving
Specification Preserving
Refinement Refactoring
Abstraction Refactoring
Collaboration
interacts
Tool
allows
UserInput
uses
Configuration
uses
ModelingTool
Verification SpecificationTool
VerificationTool
Deductive VerificationTool
Reachability Analyzer
produces
produces
produces
uses
Modeling
Formalism
Formalism
utilizes
Model
ModelingFormalism
has
references
Specification Formalism
Automaton
Program
Automaton
contains
Hybrid Program
Hybrid Automaton
Timed Automaton
comprises
comprises
Automaton Element
constrains
Transition
connect
State
Continuous State
Differential Equations
Differential Equation
PartialDE
LinearPDE
OrdinaryDE
LinearODE
contains
constrains
Logic
Logic
dL
CTL*
LTL
CTL
TCTL
utilizes
expectations on
aims at
Verification Specification
supported by
starts at
ensures
StartCondition
Hint/Strategy
Correctness Criterion
Characteristics
Compositionality
Non-Determinism
Parallelism
Urgency
Synchronization
Condition
constrains
PostCondition
PreCondition
Invariant
Verification
VerificationGoal
Safety
Liveness
Fairness
Reactivity
Controllability
DeadlockFreedom
VerificationResult
ReachableStates
CounterExample
Proof
{disjoint}
PartialProof
SuccessfulProof
comprises
implements
refutes
refutes
verifies
ProofRule
can be applied
was applied
ProofStep
OpenProofStep
ClosedProofStep
consists only of
implements

# 5 Related Work

In the following, a brief selection of related work is presented, starting with closely related hybrid component frameworks. Further valuable input for the work addressed in this proposal can be found in the field of hybrid system verification and in component-based software engineering.

## 5.1 Hybrid Component Frameworks

Hybrid component frameworks are closely related to the proposed work, although most of the presented frameworks have a slightly different focus.
**Towards Component Based Design of Hybrid Systems: Safety and Stability (Damm et al. [16]).** The key achievement Damm et al. is the development of a compositional framework for component based design of hybrid controllers taking into account realistic assumptions about reaction times. They introduce the notion of *alarms*, which alert the environment of possible stability or safety issues that might arise after an *escape interval*, during which safety and stability are still guaranteed. This is necessary because of the decentralized setting—as relevant for Autosar[4] based automotive development—where modes are responsible for creating awareness for the need of possible mode-switching and no centralized control structure is available.

The behavior of the component is given in terms of runs of especially defined *hybrid automata with inputs* (i. e., a hybrid automaton is *admissible* for a basic component interface). These automata consist of sets of variables (local, input, output), modes and transitions, initial states, local invariants and differential inclusions, restricting the evolution of the inputs and continuous transitions. To describe the model of a plant, a single-mode hybrid automaton extended with safety and stability conditions (in the form of open first-order predicates) is used. Input variables model the set of actuators and additional variables are used to model was is typically called *disturbances*.

Before interconnecting components, connecting outports (propagating an alarm) to inports (which can handle incoming alarms), the validity of the connection has to be tested (i. e., check if the second component can handle the alarm raised by the first one). Resulting components themselves are handled as simple components again, which in turn allows for arbitrary hierarchical combinations of components.

To verify the stability of a system Lyapunov functions—functions that map each system state onto a non-negative energy value, under the restriction that the function must decrease along along every trajectory until it reaches its minimum—are used.

One of the major differences of this approach to our approach is, that we focus on centralized controllers, while Damm et al. consider distributed systems and thus require the introduction of additional concepts, such as

[4] `http://www.autosar.org`

alarms. Furthermore, their framework was designed for use in automotive control environments, which relates closely solely to our microscopic models as described in Section 2.5.

## 5.2 Domain Specific Languages (DSL)

DSLs are languages specialized to a specific application domain. They can be used to simplify creation of models by supporting easy combination of domain elements (i. e., components) to form larger models.

**A Framework for Unambigious and Extensible Specification of DSMLs for Cyber-Physical Systems (Simko et al. [62]).** Simko et al. state, that due to the heterogeneity of physical behavior and the diversity of models describing computational behavior employing heterogeneous languages, precise structural and behavioral specifications are necessary, in order to enable co-design, reusability and integrity of components. They further state, that this calls for *Model-Based Engineering* paradigms for CPS, which in turn rely on *Domain-Specific Modeling Languages* (DSMLs). Accordingly, they propose a formal framework based on mathematical logic, to support unambiguous specification and formal reasoning, that furthermore naturally supports reusability and extensibility.

For this purpose, they make use of a logic programming language called FORMULA (i. e., a constraint logic programming tool, based on fixed-point logic [30]) and illustrate their framework with the bond graph language (i. e., a multi-domain graphical representation of physical systems, describing the structure of energy flow in the system [32]). Furthermore, extensibility of the approach is demonstrated, by extending the bond graph language with physical domains, resulting in hybrid bond graphs.

Summing up, they introduce a framework for formal specification and simulation. Although they show the extensibility of the approach, the presented framework does not really represent a component framework. Furthermore, there is no focus on verification of the behavior of the underlying models.

**Modeling Cyper-Physical Systems: Model-Driven Specification of Energy Efficient Buildings (Kurpick et al. [38]).** In their paper, Kurpick et al. describe an approach to model buildings from an energy perspective using hybrid system models. They introduce a DSL for modeling buildings and technical facilities with focus on improving the energy efficiency. They demonstrate, that it is possible to use standard modeling techniques (e. g., UML) and adapt them as DSLs for CPS.

## 5.3 Hybrid System Verification

Especially on verification of microscopic models of single autonomous cars several related publications exists. A closely related approach is described in [42], where the verification of a distributed car control system

is presented in a modular hierarchical way. However, the approach is specifically fitted to the car control issue. Mitsch et al. [46] use formal verification techniques in order to verify properties for intelligent speed adaptation of autonomous cars on highways. Loos et al. [43] concentrate on efficiency analysis of formally verified cruise controllers. However, these approaches are not based on components, but model entire CPS at once.

## 5.4 Component-Based Software Engineering

Chen et al. [13] propose verification of component-based systems. Even though their approach does not deal with hybrid systems, it could provide valuable input for our proposed approach. Classical component-based software engineering has already been the focus of a considerable amount of research, like for example [15, 31]. However, the topics of verification or hybrid systems have not been the focus so far.

# 6 Research Focus

Pursuing the overall vision of a verification-driven engineering framework for CPS as presented above is in line with ongoing current research efforts at the involved institutions in a series of research projects. The CMU Pittsburgh and the JKU Linz already have established an intensive and successful cooperation in that area.

This report—in accordance with our research at CMU—focuses on the first goal discussed before, serving as an essential building block of the overall vision, whereas the other two are going to be focused by research of cooperating fellow researchers at both institutions, namely Prof. André Platzer and Dr. Stefan Mitsch. In particular, the research focus was on *modeling and verification of modular traffic nets*, as described in Section 6.1. Furthermore, first approaches for model transformations were implemented, namely in the form of a translation of KeYmaera models to the hybrid system interchange format CIF, as described in Section 6.2.

## 6.1 Modeling and Verification of Modular Traffic Nets

In accordance to the initial proposal, a component language for modeling of traffic situations was developed. This is a first step towards a generic component language for hybrid systems, as it shows its feasibility through application to the field of traffic control.

### 6.1.1 A Modular and Verified Traffic Network

The concept of a modular and verified traffic network is, to divide a traffic network into small thus simple parts, that can be verified and then connected to form a larger network of blocks. At first we define a generic block (Section 6.1.2) and its properties (Section 6.1.3). Then we define how

to connect two arbitrary blocks in a way that the resulting construct is again a (larger) block (Section 6.1.4). Finally, we introduce concrete traffic blocks (i. e., exemplary we introduce a traffic light) (Section 6.1.5) and show how to build larger traffic networks using them (Section 6.1.6). Section 6.1.7 concludes this section.

### 6.1.2 A generic Block

**Definition 6.1 (Block)** *A* Block $B$ *is defined as a tuple consisting of the following elements:*

$$B = (I, O, in, cap, maxout, otime, pre, load, out, oflow)$$

*Two sets representing the in- and outflows of the block ($I$, $O$), functions assigning values to the basic properties of these flows ($in$, $capacity$, $maxout$,$overtime$), preconditions that have to hold for them ($pre$) and functions returning calculation rules for advanced properties that depended on the basic ones ($load$, $out$, $overflow$).*

Fig. 3 shows an overview of a generic block. A detailed description of the elements can be found below.

- $I$ is a finite, ordered set $\{I_1, ..., I_n\}$ of $n$ inflows into the block $B$. An inflow represents any kind of road a vehicle can use to enter the current block.
- $O$ is a finite, ordered set $\{O_1, ..., O_m\}$ of $m$ outflows from the block $B$. Similar as with inflows, an outflow can be any kind of road, through which a vehicle can leave the current block.
- $in$ is a function over the set of inflows $I$, assigning an actual inflow rate to each inflow: $I \rightarrow \mathbb{R}^+$. The resulting value represents the number of vehicles that pass through the inflow in a given time (number of vehicles per unit time).
- $cap$ is a function over the set of inflows $I$, assigning a maximal capacity to each inflow: $I \rightarrow \mathbb{R}^+$. The capacity is the number of vehicles, that can be on an inflow at any point in time (number of vehicles). The value usually depends on environmental characteristics (e. g., a long straight road with many lanes can contain a lot of cars).
- $maxout$ is a function over the set of outflows $O$, assigning a maximal outflow rate to each outflow: $O \rightarrow \mathbb{R}^+$. The values represent the flow capacity of the outflow, i. e., the respective road. The maximum flow usually depends on the characteristics of the road (e. g., more lanes usually means more capacity and thus a higher maximum outflow).
- $otime$ is a function over the set of outflows $O$, assigning a maximum time span for which the outflow may produce an overflow: $O \rightarrow \mathbb{R}^+$.
- $pre$ is a set of predicates over $B$ setting conditions, that have to hold initially.

- $load$ is a function over the set of inflows $I$, returning a calculation rule for the actual capacity used as a function of the inflows $in$: $I \rightarrow \left(\left(I \rightarrow \mathbb{R}^+\right)^n \rightarrow \mathbb{R}^+\right)$. The calculated value represents the maximum number of cars, that might be present on the corresponding inflow at any one point in time (number of vehicles).
- $out$ is a function over the set of outflows $O$, returning a calculation rule for the average outflow rates as a function of the inflows $in$: $O \rightarrow \left(\left(I \rightarrow \mathbb{R}^+\right)^n \rightarrow \mathbb{R}^+\right)$. The calculated value represents the average number of cars, that will actually leave the block over the respective outflow in a given time (number of vehicles per unit time).
- $oflow$ is a function over the set of outflows $O$, returning a calculation rule for the maximum overflow produced by each outflow as a function of the inflows $in$ and the outflows $out$: $O \rightarrow \left(\left(\left(I \rightarrow \mathbb{R}^+\right)^n, \left(O \rightarrow \mathbb{R}^+\right)^n\right) \rightarrow \mathbb{R}^+\right)$. While the functions returned by $out$ calculate the *average* flow through an outflow, average means that higher and lower flows might occur over shorter periods of time. The maximum overflow functions as returned by $overflow$, calculates the upper bound of vehicles that can be released through an outflow, which can be way above the value calculated by $out$.

Furthermore, we define the number of vehicles actually produced as overflow from an output as the product of the number of vehicles per units of time, times the duration of the overflow (cf. (1)).

$$\forall j \in \{1, ..., m\}\, over\left(O_j\right) = oflow\left(O_j\right) * otime\left(O_j\right) \tag{1}$$

### 6.1.3 Well-formed Blocks

Before we can talk about safe traffic networks, we have to define what we mean by *safety* for our model. For our model, we define that the *safety* of a system is guaranteed, as long as no traffic breakdown occurs at any point in time. In literature, a breakdown is often defined as a traffic density larger than some predefined border. As we do not consider densities, but simple flow models, we consider a slightly different definition of a traffic breakdown. We define a breakdown, as any kind of congestion that is not restricted to a single part of the road network, but propagates through it. This means that, as soon as the load at any input to any block is greater than the capacity of this input, a breakdown occurs, since the block is not able to handle his load which again would mean, that the congestion would propagate to the next block and so on.

**Definition 6.2 (Safe)** *A Block $B$ is said to be* safe*, if and only if its actual load is never greater than the capacity:*

$$\forall i \in \{1, ..., n\} : load\left(I_i\right) \leq capacity\left(I_i\right) \tag{2}$$

Besides the definition of safety, it is also important for the model to work with respect to the physical boundaries as enforced by the real-world environment. While values like $capacity$ or $maxout$ can be inferred from the properties of a road, parameters like $inflow$ or $load$ depend on other blocks and must be calculated using the provided functions. Nevertheless, these parameters have to obey the real-world boundaries.

**Definition 6.3 (Valid)** *A Block $B$ is said to be* safe, *if and only if the actual outflow never exceeds the maximum outflow:*

$$\forall j \in \{1, ..., m\} : out\,(O_j) + oflow\,(O_j) \leq maxout\,(O_j) \tag{3}$$

Def. 6.3 states, that the all actual outflows must always be less or equal than the respective maximum outflow. This includes the regular outflow and the additional overflow that might be created by the block. This condition is necessary to ensure that the model does not violate the physical bounds that restrict the real world road.

Ultimately, we want to have blocks which comply to the real-world environment and are safe. We call these kind of blocks *well-formed* (cf. Def. 6.4).

**Definition 6.4 (Well-formed)** *A Block $B$ is said to be* well-formed, *if and only if it is* safe *and* valid.

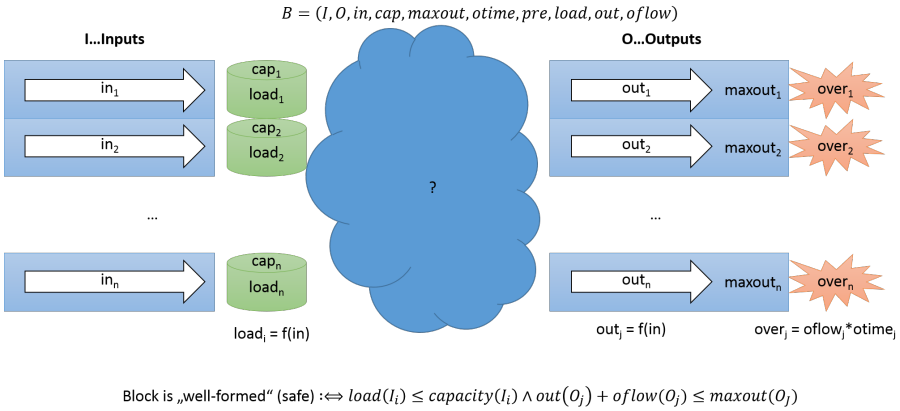



Figure 3: Visualization of a Generic Block

### 6.1.4 Connecting Blocks

Connecting two traffic blocks means linking one output of the predecessor block with one input of the successor block. In real-world term, this

would mean that the two blocks represent consecutive parts of the traffic network and are connected by an actual road. This of course means, that the cars leaving the first block move on as inputs to the second block. Finally, we want the construct resulting from connecting two blocks to be again a block. Accordingly, we define the connection of two blocks as follows:

**Definition 6.5 (Connection)** *Let* $B^1 = \left(I^1, O^1, in^1, cap^1, maxout^1, otime^1, pre^1, load^1, out^1, oflow^1\right)$ *and* $B^2 = \left(I^2, O^2, in^2, cap^2, maxout^2, otime^2, pre^2, l\right.$ *be blocks. We define* $B^3 = B^1 \xrightarrow{i,j} B^2$ *as the connection of the output* $i$ *of block* $B^1$ *(i.e.,* $O_i^1$*) to the input* $j$ *of block* $B^2$ *(i.e.,* $I_j^2$*), defined as* $B^3 = \left(I^3, O^3, in^3, cap^3, maxout^3, otime^3, pre^3, load^3, out^3, oflow^3\right)$*, with*

- $I^3 = (I^1 \backslash I_j^1) \cup I^2$,
- $O^3 = O^1 \cup (O^2 \backslash O_i^2)$,
- $in^3 : I^3 \to \mathbb{R}^+$, *with* $in^3(I^1) = in^1(I^1)$ *and* $in^3(I^2) = in^2(I^2)$,
- $cap^3 : I^3 \to \mathbb{R}^+$, *with* $cap^3(I^1) = cap^1(I^1)$ *and* $cap^3(I^2) = cap^2(I^2)$,
- $maxout^3 : O^3 \to \mathbb{R}^+$, *with* $maxout^3(O^1) = maxout^1(O^1)$ *and* $maxout^3(OI^2) = maxout^2(O^2)$,
- $otime^3 : I^3 \to \mathbb{R}^+$, *with* $otime^3(I^1) = otime^1(I^1)$ *and* $otime^3(I^2) = otime^2(I^2)$,
- $pre^3 = pre^1 \cup pre^2$,
- $load^3 : I^3 \to \left(\left(I^3 \to \mathbb{R}^+\right)^n \to \mathbb{R}^+\right)$, *with* $load^3(I^1) = load^1(I^1)$ *and* $load^3(I^2) = load^2(I^2)$,
- $out^3 : O^3 \to \left(\left(I^3 \to \mathbb{R}^+\right)^n \to \mathbb{R}^+\right)$, *with* $out^3(I^1) = out^1(I^1)$ *and* $out^3(I^2) = out^2(I^2)$,
- $oflow^3 : O^3 \to \left(\left(\left(I^3 \to \mathbb{R}^+\right)^n, \left(O^3 \to \mathbb{R}^+\right)^n\right) \to \mathbb{R}^+\right)$, *with* $oflow^3(I^1) = oflow^1(I^1)$ *and* $oflow^3(I^2) = oflow^2(I^2)$.

**Theorem 6.6** *If* $B^1$ *and* $B^2$ *are blocks, their connection* $B^3 = B^1 \xrightarrow{i,j} B^2$ *is again a block.*

According to Def. 6.5 and following Theorem 6.6, the block resulting from connection of two other blocks contains the union of the inputs of the first block and the inputs of the second block, except the one used for the connection. Furthermore, the outputs of the resulting block are the union of the outputs of the second block and the outputs of the first block, again except the one used for the connection. An example for the connection of two block with three inputs and outputs each, is shown in Fig. 4. The green and yellow arrows show the inputs and outputs of the resulting block. The two blocks are connected through the red arrow.

**Theorem 6.7** *Let* $B^1$*,* $B^2$ *and* $B^3$ *be blocks, and* $B^3 = B^1 \xrightarrow{i,j} B^2$*. Then, if* $B^1$ *and* $B^2$ *are well-formed,* $B^3$ *is also well-formed.*

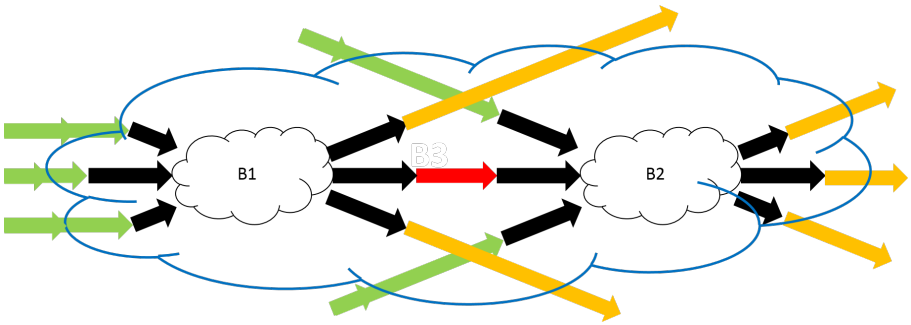



Figure 4: Connection of two blocks

### 6.1.5 Traffic Components - Traffic Light Block

In order to reach our goal of building a modular and verified traffic networks, we need to define what components this networks should be built of. The implementation of the traffic network will make use of our previously defined Blocks, which need to be instantiated with actual models. We define what is required of an actual KeYmaera model in order to be a valid traffic component.

**Definition 6.8 (Traffic Component)** *Let $B$ be a well-formed Block and $K$ a KeYmaera model. A tuple*

$$TC = (B, K)$$

*is a Traffic Component, if $K$ at least defines all properties of $B$ (i.e., $in$, $cap$, $maxout$, $otime$, $load$, $out$ and $oflow$) according to their definition and following the conditions in $pre$, and if $K$ fulfills the well-formedness property of $B$ (i.e., load and outflow are restricted by the according bounds, cf. Def. 6.4).*

**Traffic Light Block.** We define a traffic light as a straight piece of road, having two states, namely *red* and *green*. If the traffic light is red, no flow along the road is possible and vehicles start to pile up in front of the traffic light. As soon as the traffic light is green, the vehicles start flow away from the traffic light with a constant rate. This represents a simplification of the actual behavior of vehicles in front of a traffic light, which usually brake and accelerate slowly, resulting in so called shock-waves [59]. However, this abstraction it is necessary in order to ensure a hybrid systems model that can be verified using KeYmaera.

**The KeYmaera Model.** A KeYmaera model of a traffic light can be found in the appendix in A.1 and is in the following referred to as $TLK$. The controller switches the traffic light from red to green and vice versa in a fixed interval. The continuous part of the model adapts the *load* in front of the traffic light according to the current in- and outflows on the one hand, and calculates the *number of vehicles* actually leaving the traffic light. The safety condition states, that the *load* will never exceed a certain value and the *average flow* through the traffic light is limited by an upper bound.

**The Block.** A traffic light block has a single inflow ($n = 1$) and a single outflow ($m = 1$) which are separated through a traffic light and is in the following refereed to as $TLB$. We consider a simple traffic light, that has equally long red and green cycles and refer to this cycle length as $Trg \in \mathbb{R}^+$. The functions for calculating $load$, $out$ and $overflow$ are defined in (4).

$$\begin{aligned} load\,(I_1) &= Trg * in\,(I_1) \\ out\,(I_1) &= in\,(I_1) \\ over\,(I_1) = otime\,(I_1) * oflow\,(I_1) &= Trg * in\,(I_1) \end{aligned} \tag{4}$$

If the traffic light is red, vehicles start to accumulate in front of it. As the traffic light is red for exactly $Trg$ units of time and cars arrive at a rate of $in\,(I_1)$ vehicles per time unit the maximum amount of cars that might be in front of the traffic light (i. e., on the single inflow) is the product of these two values. As we assume, that no vehicles can disappear at a traffic light, the average outflow ($out$) equals the actual inflow. However, as the traffic light is red half of the time, resulting in no outflow, the outflow in the green phase has to be significantly higher than average. In fact, an inflow of $in\,(I_1)$ ($overflow$) over a time $Trg$ ($overtime$) must be compensated, thus resulting in an overflow as defined by $overflow$.

**Corollary 6.9** *The block $TLB$ is* well-formed, *if and only if*

1. $Trg * in\,(I_1) \leq cap\,(I_1)$
2. $2 * in\,(I_1) \leq maxout\,(O_j)$

**The Traffic Component.** As the KeYmaera model $TLK$ defines all properties of $TLB$ according to Corollary 6.9, the tuple $TLTC = (TLB, TLK)$ is a traffic component, since the model actually behaves as defined in (4). We prove this behavior with the help of KeYmaera.

### 6.1.6 Forming the network

Thus far, we have defined single traffic blocks and how to connect them. Provided with a library of generic traffic blocks, this knowledge can now be used to build arbitrary traffic networks from these blocks. For each block, several properties, such as maximum flow, have to be defined. For the blocks at the beginning of the network, the initial values, such as inflow, have to be fixed. Everything else can be calculated using the rules defined above and the well-formedness can be checked accordingly.

### 6.1.7 Conclusion

We have presented a method to model smaller parts of a traffic network as generic flow models, whose behavior can be verified using the hybrid system verification tool KeYmaera. Furthermore, we introduced a method to connect these blocks in order to form larger networks, which are then

again verified because each of their parts and the connection relation is verified. These method can be used to easily verify the behavior of arbitrary traffic networks and is a first step towards a generic component language for hybrid systems.

## 6.2 Model Transformation

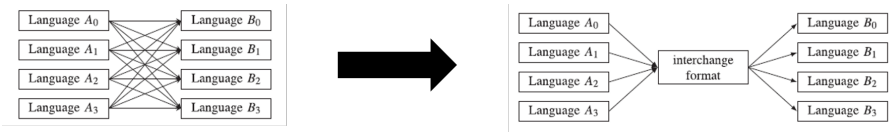



Figure 5: Interchange Formats

As described in Section 4.3, hybrid system interchange formats can be used, to transfer hybrid system models between various tools and formalisms. Fig. 5 shows the advantage of using interchange formats in contrast to direct pairwise translation. The *Compositional Interchange Format* (CIF), which was also introduced in Section 4.3, is one of these formats. CIF is supported by numerous tools and formalisms, comes with an integrated development environment (by means of an Eclipse plug-in) and supports simulation of models. As a first step towards a bi-directional translation to and from CIF, we implemented a translator from KeYmaera to the interchange format. This of course entails the possibility, to transform the resulting CIF model to all connected formalisms, as shown in Fig. 6.

### 6.2.1 The Translation

The transformation of KeYmaera models to CIF works in several steps. At first a KeYmaera model is translated to a hybrid automaton [27]. This automaton is only kept in working memory and used as an intermediate step. An external tool (i. e., Wolfram Mathematica) is then used, to determine initial values for the variables defined in the model. This step is necessary in order to allow simulation of the resulting CIF models in the integrated simulation environment. Finally, the hybrid automaton together with its initial values is translated to CIF syntax and stored as a file. The entire export process is visualized in Fig. 7. The graphic also shows that due to the very generic hybrid automata memory representation of the model, translations to further automata based syntaxes would be possible.

### 6.2.2 CIF Simulation

As mentioned above, the implemented translator from KeYmaera to CIF finds valid initial values for all fields defined in the model and thus, allows simulation of the model. The CIF simulator takes a simple vector graphic file (SVG) and animates it according to a so called *hook*-file. This hook-file

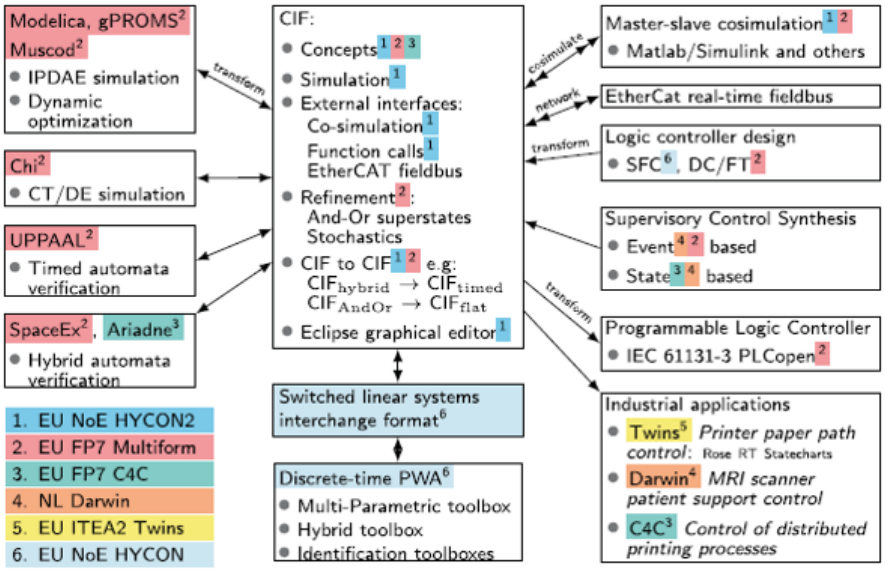



Figure 6: Compositional Interchange Format

describes how the model elements of the SVG should be animated, i. e., it links variable of the CIF model to graphical elements in the SVG. If a model describes a robot that moves along an x and y axis, the animation would move the robot in SVG according to the values of the modeled variables. 8 shows a screenshot of the simulation. A robot (green) moves around in a 2D space and should never hit the apple. The original model was created and verified using KeYmaera, then translated to CIF and simulated using the integrated simulator.

In Appendix A.2 the original KeYmaera model of the example can be found. Furthermore, Appendix A.3 contains the translated CIF model.

# 7 Future Work

In this section we first give a short overview of the ongoing research regarding our overall vision (cf. Section 7.1), before we describe possible future research directions regarding the work conducted as focus of the Marshall Plan funded research at CMU (cf. Section 7.2). Finally, the last section contains the envisioned future plans of the applicant (cf. 7.3).

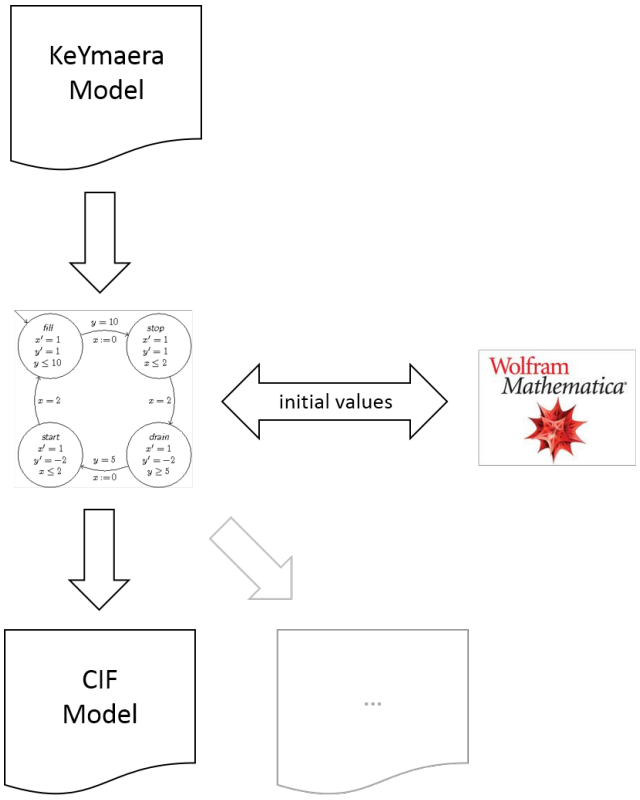



Figure 7: CIF Export

## 7.1 Overall Vision

Regarding the overall vision, presented in Section 3, the work will be continued towards design, development and implementation of a comprehensive framework for verification-driven engineering. A first step towards the realization of the first part of our proposed framework on *Modular Traffic Nets*, has been targeted as described in Section 6. Still this work has to be further improved as described in Section 7.2. While *Refactorings for Hybrid System Models* is currently target of the research of Dr. Stefan Mitsch, the last part on *Verified Model Transformation* is currently still at a very early stage.

## 7.2 Modular Traffic Nets

The initial approach was employing flow models that could be decomposed into specifically restricted components with fixed structure for addressing the macroscopic view on traffic models. Although, these first

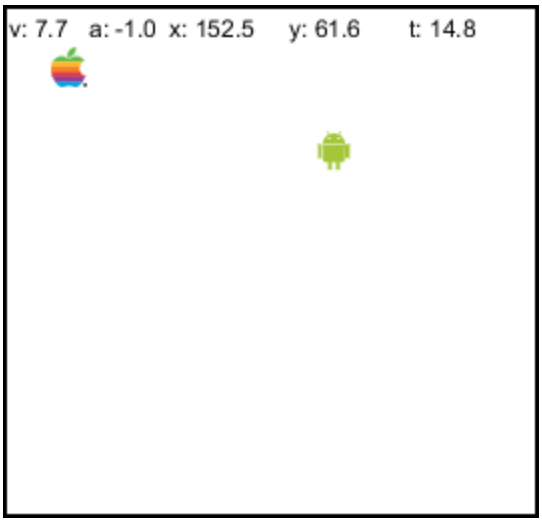



Figure 8: Simulation in CIF

results give a strong indication of the feasibility of the pursued approach, further improvements are of course necessary in order to allow modeling of more general components. Thus, a further research directions is to extend beyond flow models, as in the field of road traffic control, actions are not necessarily bound to the macroscopic view. Although, a bulk of cars as a whole may be modeled as flow, the actions of each car (e. g., considering an autonomous car) and its interactions with other cars (e. g., considering car-to-car communication) result in smaller scale microscopic models of single cars. These views allow analysis of different aspects of traffic control, like analyzing impacts of a speed limit on traffic flow (i. e., macroscopic) to see how the entire traffic network is affected and on single cars (i. e., microscopic), to verify that each car can safely abide to the limit. An example for decomposition of macroscopic traffic models is cutting a traffic net into smaller pieces, like crossroads or traffic lights. Decomposition of microscopic traffic models deals with isolating the various actions of a car or driver, like braking or accelerating. In both cases the decomposed parts of the traffic model should be as minimal as possible, in order to allow for elevated reuse in a large number of combinations when building further and larger models from them.

Ultimately, the goal of this proposal is to create concepts for a generic framework to decompose CPS into minimal verified components, which can then be verified in isolation and (re-)used to (re-)build the entire verified model. For this, the architecture and properties of an abstract component have to be specified along with the safety conditions that must hold. The definition of such a component thereby shall be generic in order to support a wide range of concrete application scenarios. The architecture

allowing to compose components into larger models must include an exact specification of the interfaces used to connect the different components to a network along with explicating the safety conditions inter-relationships, in order to avoid further inherent dependencies. If the behavior, i.e. the satisfaction of the safety condition of an abstract component, can be verified, an actual component configured within the limits of the safely condition is guaranteed to posses verified behavior. Given an appropriate method to connect components along their safety condition to a network for the CPS and provided that each component's model is verified, it is then possible to draw conclusion the safeness of the overall behavior of the CPS.

## 7.3 Further Plans

In the future, the applicant strives to enhance the CSI expertise of hybrid systems modeling and verification in Linz. With the help of an Austrian colleague from CMU (Dr. Stefan Mitsch), who will return to JKU in 2015, the work will help to establish a model-driven CPS research group at JKU, based on the EU *Marie Curie* funded project SPHINX[5]. In the course of this Marie Curie grant and overall two Marshall Plan scholarships, a close cooperation between JKU and CMU has been established.

# References


[1] Altman, E., Gaitsgory, V.: Asymptotic Optimization of a Nonlinear Hybrid System Governed by a Markov Decision Process. SIAM J. Control Optim. 35(6), 2070–2085 (1997), \url{http://dx.doi.org/10.1137/S0363012995279985}

[2] Alur, R.: Formal verification of hybrid systems. In: Proceedings of the ninth ACM international conference on Embedded software. pp. 273–278. EMSOFT '11, ACM, New York and NY and USA (2011), \url{http://doi.acm.org/10.1145/2038642.2038685}

[3] Alur, R., Dang, T., Ivančić, F.: Reachability Analysis of Hybrid Systems via Predicate Abstraction. In: Tomlin, C., Greenstreet, M. (eds.) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 2289, pp. 35–48. Springer Berlin Heidelberg (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, Lecture Notes in Computer Science, vol. 600, pp. 45–73. Springer Berlin Heidelberg (1992), \url{http://dx.doi.org/10.1007/BFb0031987}


[5] Sphinx is an extensible verification-driven engineering toolkit based on the Eclipse platform. (cf. http://www.cs.cmu.edu/afs/cs/Web/People/smitsch/tools.html)

[5] Amin, S., Abate, A., Prandini, M., Lygeros, J., Sastry, S.: Reachability Analysis for Controlled Discrete Time Stochastic Hybrid Systems. In: Hespanha, J.P., Tiwari, A. (eds.) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 3927, pp. 49–63. Springer Berlin Heidelberg (2006), \url{http://dx.doi.org/10.1007/11730637_7}

[6] 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.) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 4416, pp. 587–600. Springer Berlin Heidelberg (2007)

[7] Berkenkötter, K., Bisanz, S., Hannemann, U., Peleska, J.: The HybridUML profile for UML 2.0. International Journal on Software Tools for Technology Transfer (STTT) 8(2), 167–176 (2006), \url{http://dx.doi.org/10.1007/s10009-005-0211-z}

[8] 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)

[9] Brillout, R.: Using Theorem Provers as Preprocessors for Hybrid Systems Model Checking. Ph.D. thesis, Carnegie Mellon University, Pittsburgh and PA (2012)

[10] Broman, D., Lee, E.A., Tripakis, S., Törngren, M.: Viewpoints, Formalisms, Languages, and Tools for Cyber-Physical Systems (preprint). In: Proceedings of the 6th International Workshop on Multi-Paradigm Modeling (MPM 2012) (2012)

[11] 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)

[12] Cellier, F.: Continuous System Modeling. Springer (1991), \url{http://books.google.at/books?id=c8p0DAtyEUAC}

[13] Chen, Z., Liu, Z., Ravn, A.P., Stolz, V., Zhan, N.: Refinement and Verification in Component-based Model-driven Design. Sci. Comput. Program. 74(4), 168–196 (2009), \url{http://dx.doi.org/10.1016/j.scico.2008.08.003}

[14] Clarke, E.M., Wing, J.M.: Formal Methods: State of the Art and Future Directions. ACM Comput. Surv. 28(4), 626–643 (1996), \url{http://doi.acm.org/10.1145/242223.242257}

[15] Crnkovic, I.: Component-based software engineering — new challenges in software development. Software Focus 2(4), 127–133 (2001), \url{http://dx.doi.org/10.1002/swf.45}

[16] Damm, W., Dierks, H., Oehlerking, J., Pnueli, A.: Towards Component Based Design of Hybrid Systems: Safety and Stability. In: Manna, Z., Peled, D. (eds.) Time for Verification,

Lecture Notes in Computer Science, vol. 6200, pp. 96–143. Springer Berlin Heidelberg (2010), \url{http://dx.doi.org/10.1007/978-3-642-13754-9_6}

[17] Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. 5(1-2), 29–35 (1988)

[18] David, R., Alla, H.: On Hybrid Petri Nets. Discrete Event Dynamic Systems 11(1-2), 9–40 (2001), \url{http://dx.doi.org/10.1023/A:1008330914786}

[19] Fowler, M., Beck, K.: Refactoring: improving the design of existing code. Object Technology Series, Addison-Wesley, 8 edn. (1999), \url{http://books.google.at/books?id=1MsETFPD3I0C}

[20] 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: Ganesh Gopalakrishnan, S.Q. (ed.) Proc. 23rd International Conference on Computer Aided Verification (CAV). LNCS, Springer (2011)

[21] Fritzson, P., Engelson, V.: Modelica — A unified object-oriented language for system modeling and simulation. In: Jul, E. (ed.) ECOOP'98 — Object-Oriented Programming, Lecture Notes in Computer Science, vol. 1445, pp. 67–90. Springer Berlin Heidelberg (1998), \url{http://dx.doi.org/10.1007/BFb0054087}

[22] Geisberger, E., Broy, M.: Integrierte Forschungsagenda Cyber-Physical Systems (2012)

[23] 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), \url{http://www.sciencedirect.com/science/article/pii/S1045926X06000589}

[24] Gupta, A.: Formal Hardware Verification Methods: A Survey. In: Kurshan, R. (ed.) Computer-Aided Verification, pp. 5–92. Springer US (1993), \url{http://dx.doi.org/10.1007/978-1-4615-3556-0_2}

[25] Harel, D., Kupferman, O., Vardi, M.: On the complexity of verifying concurrent transition systems. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR '97: Concurrency Theory, Lecture Notes in Computer Science, vol. 1243, pp. 258–272. Springer Berlin Heidelberg (1997), \url{http://dx.doi.org/10.1007/3-540-63141-0_18}

[26] Helbing, D.: From microscopic to macroscopic traffic models. In: Parisi, J., Müller, S., Zimmermann, W. (eds.) A Perspective Look at Nonlinear Media, Lecture Notes in Physics, vol. 503, pp. 122–139. Springer Berlin Heidelberg (1998), \url{http://dx.doi.org/10.1007/BFb0104959}

[27] Henzinger, T.A.: The Theory of Hybrid Automata. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96). pp. 278–292. IEEE Computer Society Press (1996)

[28] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Longman Publishing Co., Inc., Boston and MA and USA, 3 edn. (2006)

[29] Hu, J., Lygeros, J., Sastry, S.: Towards a Theory of Stochastic Hybrid Systems. In: Lynch, N., Krogh, B. (eds.) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 1790, pp. 160–173. Springer Berlin Heidelberg (2000), \url{http://dx.doi.org/10.1007/3-540-46430-1_16}

[30] Jackson, E.K., Kang, E., Dahlweid, M., Seifert, D., Santen, T.: Components, Platforms and Possibilities: Towards Generic Automation for MDA. In: Proceedings of the Tenth ACM International Conference on Embedded Software. pp. 39–48. EMSOFT '10, ACM, New York and NY and USA (2010), \url{http://doi.acm.org/10.1145/1879021.1879027}

[31] Jifeng, H., Li, X., Liu, Z.: Component-Based Software Engineering. In: Hung, D., Wirsing, M. (eds.) Theoretical Aspects of Computing – ICTAC 2005, Lecture Notes in Computer Science, vol. 3722, pp. 70–95. Springer Berlin Heidelberg (2005), \url{http://dx.doi.org/10.1007/11560647_5}

[32] Karnopp, D.C., Margolis, D.L., Rosenberg, R.C.: System Dynamics: Modeling and Simulation of Mechatronic Systems. John Wiley & Sons, Inc, New York and NY and USA (2006)

[33] Kern, C., Greenstreet, M.R.: Formal Verification in Hardware Design: A Survey. ACM Trans. Des. Autom. Electron. Syst. 4(2), 123–193 (1999), \url{http://doi.acm.org/10.1145/307988.307989}

[34] Kim, K.D., Kumar, P.: Cyber-Physical Systems: A Perspective at the Centennial. Proceedings of the IEEE 100(Special Centennial Issue), 1287–1308 (2012)

[35] Klir, G.J.: An Approach to General Systems Theory. Van Nostrand Reinhold Co (1969)

[36] Kordon, F., Hugues, J., Renault, X.: From Model Driven Engineering to Verification Driven Engineering. In: Proceedings of the 6th IFIP WG 10.2 international workshop on Software Technologies for Embedded and Ubiquitous Systems. pp. 381–393. SEUS '08, Springer-Verlag, Berlin and Heidelberg (2008)

[37] 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), \url{http://drops.dagstuhl.de/opus/volltexte/2011/3212}

[38] Kurpick, T., Pinkernell, C., Look, M., Rumpe, B.: Modeling Cyber-physical Systems: Model-driven Specification of Energy Efficient Buildings. In: Proceedings of the Modelling of the Physical World Workshop. pp. 2:1–2:6. MOTPW '12, ACM, New York and

NY and USA (2012), \url{http://doi.acm.org/10.1145/2491617.2491619}

[39] Lamport, L.: Proving the Correctness of Multiprocess Programs. Software Engineering, IEEE Transactions on Proving the Correctness of Multiprocess Programs 3(2), 125–143 (1977)

[40] Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997), \url{http://dx.doi.org/10.1007/s100090050010}

[41] Lee, E.: Cyber Physical Systems: Design Challenges. In: 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC 2008). pp. 363–369 (2008)

[42] Loos, S.M., Platzer, A., Nistor, L.: Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified. In: Butler, M., Schulte, W. (eds.) FM : FM. LNCS, vol. 6664, pp. 42–56. Springer (2011)

[43] Loos, S.M., Witmer, D., Steenkiste, P., Platzer, A.: Efficiency Analysis of Formally Verified Adaptive Cruise Controllers. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013) (2013)

[44] Mens, T., Taentzer, G.: Model-driven Software Refactoring. In: Dig, D. (ed.) 1st Workshop on Refactoring Tools, WRT 2007, in conjunction with 21st European Conference on Object-Oriented Programming, July 30 - August 03, 2007, Berlin, Proceedings. pp. 25–27 (2007)

[45] Meyer, B.: Applying Design by Contract. Computer 25(10), 40–51 (1992), \url{http://dx.doi.org/10.1109/2.161279}

[46] Mitsch, S., Loos, S.M., Platzer, A.: Towards Formal Verification of Freeway Traffic Control. In: Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems. pp. 171–180. ICCPS '12, IEEE Computer Society, Washington and DC and USA (2012), \url{http://dx.doi.org/10.1109/ICCPS.2012.25}

[47] Mitsch, S., Passmore, G.O., Platzer, A.: A Vision of Collaborative Verification-Driven Engineering of Hybrid Systems. In: Proceedings of Enabling Domain Experts to use Formalised Reasoning - Symposium AISB, Do-Form. pp. 8–17 (2013)

[48] MoBIES team: HSIF semantics (version 3, synchronous edition): Technical Report (2002)

[49] Nadales Agut, D. E., van Beek, D.A., Rooda, J.E.: Syntax and semantics of the compositional interchange format for hybrid systems. The Journal of Logic and Algebraic Programming 82(1), 1–52 (2013)

[50] Ouimet, M., Lundqvist, K.: Formal Software Verification: Model Checking and Theorem Proving (2007), \url{http://www.mrtc.mdh.se/index.php?choice=publications\&id=1436}

[51] 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)

[52] Pinto, A., Carloni, L.P., Passerone, R., Sangiovanni-Vincentelli, A.: Interchange Format for Hybrid Systems: Abstract Semantics. In: Hespanha, J.P., Tiwari, A. (eds.) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 3927, pp. 491–506. Springer Berlin Heidelberg (2006)

[53] Pinto, A., Sangiovanni-Vincentelli, A.L., Carloni, L.P., Passerone, R.: Interchange formats for hybrid systems: review and proposal. In: Proceedings of the 8th international conference on Hybrid Systems: computation and control. pp. 526–541. HSCC'05, Springer-Verlag, Berlin and Heidelberg (2005)

[54] Platzer, A.: Differential Dynamic Logic for Hybrid Systems. Journal of Automated Reasoning 41(2), 143–189 (2008), \url{http://dx.doi.org/10.1007/s10817-008-9103-8}

[55] Platzer, A.: Differential-algebraic Dynamic Logic for Differential-algebraic Programs. J. Log. Comput. 20(1), 309–352 (2010)

[56] Platzer, A.: Logic and Compositional Verification of Hybrid Systems (Invited Tutorial). In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference (CAV 2011). LNCS, vol. 6806, pp. 28–43. Springer (2011)

[57] 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)

[58] Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science. pp. 46–57. SFCS '77, IEEE Computer Society, Washington and DC and USA (1977), \url{http://dx.doi.org/10.1109/SFCS.1977.32}

[59] Richards, P.I.: Shock Waves on the Highway. Operations Research 4(1), 42–51 (1956)

[60] Sanislav, T., Miclea, L.: Cyber-Physical Systems - Concept, Challenges and Research Areas. Journal of Control Engineering and Applied Informatics 14(2) (2012)

[61] Shi, J., Wan, J., Yan, H., Suo, H.: A survey of Cyber-Physical Systems. In: International Conference on Wireless Communications and Signal Processing (WCSP 2011). pp. 1–6 (2011)

[62] Simko, G., Lindecker, D., Levendovszky, T., Jackson, E.K., Neema, S., Sztipanovits, J.: A Framework for Unambiguous and Extensible Specification of DSMLs for Cyber-Physical Systems. In: Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the. pp. 30–39 (2013)

[63] Teschl, G.: Ordinary differential equations and dynamical systems, Graduate studies in mathematics, vol. v. 140. American Mathematical Society, Providence and R.I (2012)

[64] Völzer, H., Varacca, D.: Defining Fairness in Reactive and Concurrent Systems. Journal of the ACM (JACM) 59(3), 13:1–13:37 (2012), \url{http://doi.acm.org/10.1145/2220357.2220360}

[65] Wan, K., Hughes, D., Man, K.L., Krilavicius, T., Zou, S.: Investigation on Composition Mechanisms for Cyber Physical Systems. International Journal of Design, Analysis and Tools for Integrated Circuits and Systems 2(1), 30–40 (2011)

[66] 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), \url{http://doi.acm.org/10.1145/1978802.1978807}

[67] Zhang, L., She, Z., Ratschan, S., Hermanns, H., Hahn, E.: Safety Verification for Probabilistic Hybrid Systems. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 6174, pp. 196–211. Springer Berlin Heidelberg (2010), \url{http://dx.doi.org/10.1007/978-3-642-14295-6_21}

[68] Zwillinger, D.: Handbook of differential equations. ACADEMIC PressINC (1998), \url{http://books.google.at/books?id=YD4PL0GZg0cC}

# A Appendix

## A.1 Traffic Light - KeYmaera Model

```
\functions {
}

\programVariables {
 R t; //system time
 R tt; //traffic light change time
 R Trg; //red-green interval
 R rg; //traffic light status, r=0,g=1
 R in;   //in(I_1)...actual inflow
 R capacity;  //capacity(I_1)...capacity of inflow
 R load;   //load(I_1)...used load of inflow
 R out;   //out(O_1)...actual outflow
 R l;   //current load
}

\problem {
 // -- initial state description --
 //times
 t=0 & tt=0 & Trg>0
 //actual load
 & l=0
 //rg can only be 0 or 1
 & rg=0 //let traffic light be red initially
 //value ranges (R+)
 & in>=0 & capacity>=0 & load>=0 & out>=0 //& over>=0 & maxout>=0
 //advanced properties
  & load=Trg*in
 //pre
 & load<=capacity & out>=2*in
 -> \[
   (
    /* if the traffic light has not changed for Trg,
    change from red to green or vice-versa */
    if(tt=Trg) then
     tt:=0;
     /* if the traffic light is red, rg=0 implying no flow,
     if its green rg=1 implying flow*/
     if(rg=1) then
      rg:=0
     else
      rg:=1
     fi
    fi;
```

```
    {l'=in-rg*out, t'=1,tt'=1,tt<=Trg}
   )*
   @invariant(t>=0 & tt>=0 & tt<=t & tt<=Trg
    & rg*(rg-1) = 0
    & l<=load
    & ( (rg=0) -> l + (Trg-tt)*(in) + (Trg)*(in-out) <= load )
    & ( (rg=1) -> l + (Trg-tt)*(in-out) <= load )
   )
 \]
 (l<=load)
}
```

## A.2 Transformer Example - KeYmaera Model

```
\functions {
  R A;// @func("MAX_ACCELERATION"); /*robot's maximum acceleration*/;
  R B;// @func("MAX_BRAKING"); /*robot's maximum braking*/
  R T;// @func("MAX_REACTION_TIME"); /* Time-trigger limit on evolution */
  R r; /* Radius of the robot */
  R obsr; /* Radius of the obstacle */
  R buffer; /* Required distance between center of robot and obstacle */
}
\programVariables{
  R t; /* time */
  R trackr; /* Radius of track */
  R x  @sensor("MY_POS_D1"); /* Position of robot in x direction */
  R y  @sensor("MY_POS_D2"); /* Position of robot in y direction */
  R v  @sensor("MY_LVEL"); /* Linear velocity of robot */
  R a  @actuator("MY_LACC"); /* Linear acceleration of robot */
  R dirx; /* Unit vector in direction of travel, x direction */
  R diry; /* Unit vector in direction of travel, y direction */
  R obsx  @sensor("OBS_POS_D1"); /* x Position of obstacle */
  R obsy  @sensor("OBS_POS_D2"); /* y Position of obstacle */
}

\problem{
(
   v >= 0
 & (
    Abs(x - obsx) > v^2 / (2*B) + buffer
  |
   Abs(y - obsy) > v^2 / (2*B) + buffer
  )
 & dirx^2 + diry^2 = 1
 & trackr != 0
 & A >= 0
 & B > 0
```

```
 & T > 0
 & r > 0
 & obsr > 0
 & buffer >= r + obsr
)
->
\[(
      (a := -B)
   ++ (?v = 0; a := 0)
   ++ (
  ?(Abs(x - obsx) > v^2/(2*B) + (A/B + 1) * (A/2 * T^2 + T*v) + buffer
  | Abs(y - obsy) > v^2/(2*B) + (A/B + 1) * (A/2 * T^2 + T*v) + buffer);
     /* control acceleration a */
     a := *; ?-B <= a & a <= A;
     /* guarded non-deterministic assignment of radius ("steer") */
     trackr := *; ?trackr != 0
  );
   t := 0;
   {x'=v*dirx, y'=v*diry, dirx'=-a/trackr*diry, diry'=a/trackr*dirx,
    v'=a, t' = 1, t <= T, v >= 0}
     @invariant( t >= 0,
        dirx^2 + diry^2 = 1
      ) /* Differential Invariant Proof Annnoation */
   )*@invariant(
      v >= 0
 & trackr != 0
 & dirx^2 + diry^2 = 1
 & (
    Abs(x - obsx) > v^2 / (2*B) + buffer
  |
   Abs(y - obsy) > v^2 / (2*B) + buffer
  )
 ) /* Loop Invariant Proof Annotation */
\](x-obsx)^2 + (y-obsy)^2 > buffer^2) /* Safety Condition*/
}
```

## A.3 Transformer Example - CIF Model

```
model Test() =
|[
disc control real obsy = 0.0
cont control real y = 0.0
disc control real obsx = 0.0
const real r = 0.25
const real B = 2.0
cont control real diry = 0.0
cont control real x = -1.0
```

```
const real buffer = 0.5
cont control real t = 0.0
const real A = 0.0
cont control real dirx = 1.0
const real T = 1.0
cont control real v = 1.0
const real obsr = 0.25
disc control real a = 0.0
disc control real trackr = 1.0
::
TestAutomaton:
|(
mode V21 =
initial;
tcp false;
when true do a := -B goto V2;
when (v = 0.0) goto V4;
when (((abs((x - obsx)) > ((((v ^ 2.0) /
  ((2.0 * B))) + ((((A / B) + 1.0)) * ((((A / 2.0) *
  (T ^ 2.0)) + (T * v))))) + buffer))
 or (abs((y - obsy)) > ((((v ^ 2.0) /
  ((2.0 * B))) + ((((A / B) + 1.0)) * ((((A / 2.0) *
  (T ^ 2.0)) + (T * v))))) + buffer))))
 goto V8

mode V2 =
tcp false;
when true goto V18

mode V4 =
tcp false;
when true do a := 0.0 goto V6

mode V8 =
tcp false;
when true do a :=  goto V10

mode V18 =
tcp false;
when true do t := 0.0 goto V19

mode V6 =
tcp false;
when true goto V18

mode V10 =
tcp false;
```

```
when ((-B <= a) and (a <= A)) goto V12

mode V19 =
tcp false;
when true goto V20

mode V12 =
tcp false;
when true do trackr :=  goto V14

mode V20 =
tcp (t <= T), (v >= 0.0);
inv (x' = (v * dirx)), (y' = (v * diry)),
 (dirx' = ((-a / trackr) * diry)),
 (diry' = ((a / trackr) * dirx)),
 (v' = a), (t' = 1.0);
when true goto V21

mode V14 =
tcp false;
when (trackr /= 0.0) goto V16

mode V16 =
tcp false;
when true goto V18

)|
]|
```