Acumen: An Open-source Testbed for Cyber-Physical Systems Research

Walid Taha[1], Adam Duracz[1], Yingfu Zeng[2], Kevin Atkinson[2], Ferenc A. Bartha[2], Paul Brauner[2], Jan Duracz[1], Fei Xu[1], Robert Cartwright[2], Michal Konečný[3], Eugenio Moggi[4], Jawad Masood[2], Pererik Andreasson[1], Jun Inoue[2], Anita Sant'Anna[1], Roland Philippsen[1], Alexandre Chapoutot[5], Marcia O'Malley[6], Aaron Ames[7], Veronica Gaspes[1], Lise Hvatum[8], Shyam Mehta[8], Henrik Eriksson[9], and Christian Grante[10]

[1] School of Information Technology, Halmstad University, Halmstad, Sweden
`{name.surname}@hh.se`,
[2] Department of Computer Science, Rice University, Houston TX, USA
`{name.surname}@rice.edu`,
[3] Computer Science Group, Aston University, Birmingham, UK
`m.konecny@aston.ac.uk`,
[4] DIBRIS, University of Genova, Genoa, Italy
`moggi@unige.it`
[5] ENSTA ParisTech - U2IS, Paris, France `alexandre.chapoutot@ensta.fr`,
[6] Department of Mechanical Engineering, Rice University, Houston TX, USA
`omalleym@rice.edu`,
[7] School of Mechanical Eng., Georgia Institute of Technology, Atlanta GA, USA
`ames@gatech.edu`,
[8] Schlumberger, Houston TX, USA `{name.surname}@slb.com`,
[9] Dependable Systems, SP Technical Research Institute of Sweden, Borås, Sweden,
`henrik.eriksson@sp.se`,
[10] AB Volvo, Gothenburg, Sweden
`christian.grante@volvo.com`,

Abstract. Developing Cyber-Physical Systems requires methods and tools to support simulation and verification of hybrid (both continuous and discrete) models. The Acumen modeling and simulation language is an open source testbed for exploring the design space of what rigorous-but-practical next-generation tools can deliver to developers of Cyber-Physical Systems. Like verification tools, a design goal for Acumen is to provide rigorous results. Like simulation tools, it aims to be intuitive, practical, and scalable. However, it is far from evident whether these two goals can be achieved simultaneously. This paper explains the primary design goals for Acumen, the core challenges that must be addressed in order to achieve these goals, the "agile research method" taken by the project, the steps taken to realize these goals, the key lessons learned, and the emerging language design.

Key words: Testbed, Cyber-Physical Systems (CPS), Modeling, Simulation, Hybrid Systems, Open Source Software

This work was supported by US NSF award CPS-1136099, the Swedish Knowledge Foundation (KK), The CERES Center, and VINNOVA (Dnr. 2011-01819).

1 Introduction

Developing novel Cyber-Physical and IoT Systems requires methods and tools to support simulation and verification of hybrid systems models. Hybrid systems modeling languages are mathematical formalism that support the descriptions of dynamics that can be continuous in some parts and discontinuous or discrete in others. Acumen is an open source testbed for exploring the design space of what rigorous-but-practical next-generation tools can deliver to developers of Cyber-Physical Systems. Like verification tools, a design goal for Acumen is to provide correct and mathematically rigorous results. Like simulation tools, it aims to be intuitive, practical, and scalable. However, it is far from evident whether these two goals can be achieved simultaneously.

Contributions: The key contributions of this paper are to articulate and report on the results of a method for addressing a complex set of goals such as those put for the design of Acumen. We begin by presenting the goals set for Acumen (Section 2). Next, we analyze the challenges that face an undertaking of this scope (Section 3). We then describe the key features of the "agile research method" to advance towards these goals, the main milestones in applying this method to date, and some key lessons in the last five years of the project (Section 4). We briefly describe the emerging language design (Section 5), and conclude with a summary and an overview of current priorities for the development of Acumen.

We posit that the ambitious goals of the project as well as the challenges that face it are representative of the goals and challenges of interdisciplinary paradigms such as CPS and IoT, and see the gradual progress made by the project as giving assurance about the deep advances that can be expected from those disciplines. Through this exposition, we hope to interest other academic and industrial partners to become involved in developing of Acumen.

Related Work: In terms of "final product", the most closely related tools to our work are hybrid systems verification tools, such as CHARON [2], KeyMaera [18], and SpaceEx [10], as well as equational modeling languages such as Modelica [4]. At the highest level, the work described here can be viewed as an effort to bridge the gap between the first class of (rigorous) tools and the second class of widely popular tools that are generally viewed as being much more accessible and broadly applicable (but provide no guarantees of correctness). Technical comparisons between the tools and Acumen on technical grounds can be found in the papers on Acumen cited in this paper. The focus of this paper is the process through which a rigorous modeling language aimed specifically at the CPS domain is being developed. Unfortunately, the literature on the development process for such tools in particular, or domain-specific languages in general, is relatively sparse. Related work includes commonality and variability analysis [7], but the CPS domain is much broader than the intended types of problems for this analysis. The survey by Mernik, Heering, and Sloane [17] suggests that existing methods require that the domain is much more clearly defined than is possible for a large and evolving area such as CPS.

2 Inception and Design Goals

The primary goal for Acumen is to serve as a testbed that facilitates research into rigorous-but-practical tools for CPS. A secondary goal, which stems from a necessity for achieving the primary goal in the absence of standard methods for studying such tools, is to serve as a testbed for programming languages and software engineering research. A tertiary goal, which stems in part from the need for developing evaluation methods for the primary goal, is accessibility to users. This goal entails that Acumen had to prove to be an effective device for teaching. The rest of this section elaborates these three goals.

Testbed for CPS research: At the very outset of the project the motivation was simply to develop controllers for horizontal drilling tractors (robots) for oil wells. Quickly, this concrete problem pointed to a much broader problem of the need for coherent toolchains for CPS design. Discussions with domain experts suggested and eventually confirmed that a wide range of different tools are used to design and analyze different components of products such as horizontal drilling tractors, or any of a wide range of non-trivial robotic systems. The problem of "coherence" of the existing toolchains is largely due to:

- The need to use different formalisms and tools for different subsystems,
- The absence of mechanized methods for checking consistency between the results of these tools, and
- The absence of methods that ensure the correctness of the results of individual tools, even for the smallest problems.

Overcoming the "coherence" problem is a central goal for Acumen [24].

Toolchains consist of software applications that support modeling, simulation, and verification of various subsystems. Such tools constitute infrastructure for supporting virtual prototyping and testing of these subsystems, as well as tools for visualizations. We were inspired by the power of specialized tools such as CarSim [3] (for vehicles) or Gazebo [11] (for robotics), but also surprised by the lack of transparency and user control over underlying dynamical models, as well as the need for such radically different tools to support these closely related domains. Access to such models is essential for allowing users to control the computational cost (and accuracy) of the models being used for virtual testing, and to interpret the results of such tests. Out of these observations emerged the goal of transparency with respect to models, and ensuring that the user has full access to (and control over) such models, as well as the goal of being a unifying language that can be used across a wide range of CPS domains.

A practical criterion for an effective testbed is accessibility. We define accessibility as the ease with which new users from a wide range of backgrounds can acquire the software and use it to solve a problem that interests them, or use it to learn something of value to them. First, it should be self-evident that convenience to users is an integral part of testbeds success, and that this has a direct impact on the size of audience the tool can reach. But it is also the kind of goal that is essential to spell out and stress throughout the process of developing it. As we will see in the rest of this paper, accessibility is both a source of stringent requirement on design choices and an easy criteria for members of the research team to forget as they become engaged in solving much more specific technical problems. Second, our emphasis on accessibility was driven in part out of the idea that the ability to model and simulate the world around us is important to individual and societal well-being. This partly came from Lessigs "Code is Law" [13] and, in particular, the idea that our society is increasingly being shaped by codes that only few of us understand and that even fewer truly consent to incorporating the socially-significant values induced by these codes.

Testbed for language and software engineering research: At the time of starting the development of Acumen, in the mainstream programming languages community, the formalism that came closest to providing a modeling language for Cyber-Physical Systems was Functional Reactive Programming [23] (FRP). It had been used successfully in a wide range of domains, including robotics and computer animation. However, its denotational semantics was presented in complete partial orders (CPOs) without giving real numbers detailed treatment. It therefore remained an open question whether this language could be extended to incorporate such treatment. Ideas presented in work by Edalat and Pattinson on the semantics of hybrid systems [9] appeared to provide some or all of the necessary foundations, but the connection between these two lines of work was not obvious. Developing a semantic foundation that unified these lines of work is a key goal of the Acumen project. Once such a unifying foundation is established, it can be used to elaborate various notions of computation for Cyber-Physical Systems as well as their inter-relations.

There are also more practical goals relating to investigating better methods for the design and implementation of domain-specific languages. These include the use of staging [19] (both for implementing the language and as a mechanism supported by the language) and property-based testing [22] for ensuring the quality of the code base. A related goal is the use of Acumen to investigate the possibility of exploiting parallel resources without introducing concurrency problems [6].

Tool for teaching CPS: Tools that can be used in practice must be accessible to their users, and waiting to evaluate accessibility at the end of the project entails unreasonably high risk of failure. There are two natural proxies to evaluating such accessibility. The first is usability by domain-experts, which we have already addressed in the discussion of the primary goal of serving as a testbed for CPS. The second is usability by novice users. Fortunately, this goal is highly synergistic with well-recognized need for better content, methods, and tools for teaching the emerging topic of CPS.

Because it is a new and high interdisciplinary area, CPS students come from a highly diverse set of backgrounds, and with different goals. We identified from the outset three different groups of students: The first group is college-level CPS students. It includes embedded systems and mechatronics students, who expect to have a career developing CPSs. The second group is college-level students specializing in related areas, which could be computer-related, engineering-related, science-related, or arts and humanities. Clearly, students in each of these sub-areas have different interests. They are seeking a professional degree and will spend their careers in the context of a world populated by Cyber-Physical Systems, and may well contribute to solving problems relating to such systems. The third group is high-school students that have a general interest in science, technology, engineering, and math (STEM) disciplines. A tool that can address the toolchain coherence problem should make a positive contribution to the education of these three categories of students, but it is far from obvious that one tool can cater to such different audiences.

In the next section we discuss the issues that make these goals challenging.

3 Challenges

In this section we describe some of the key challenges, especially in relation to the CPS testbed goal (and in tool chain coherence in particular). For reasons of space, we address the programming languages and teaching goals only briefly.

To address the question of how to best support the engineering process we must understand how engineers carry out their work in practice. This includes both how engineers go about designing a new product, and the more specialized skills of how they think about models of the systems they design. Even the latter is challenging for someone interested in its computational mechanization. For example, research papers are a natural source for understanding the notation and calculations used to reason about models of Cyber-Physical Systems. However, such papers rarely focus on the mechanics of derivation, and assume significant knowledge about mathematics, control, and the domain. Developing a formalism for modeling such systems entails acquiring a deep understanding of such domains. A somewhat more practical challenge is that research papers are often incomplete in their specification of concrete examples and rely instead on the intuition of experts. This complication means that even testing theories about correctly understanding the mechanics of the computations used in these papers can be challenging.

In terms of specific simulation tools, it is clear that MATLAB/Simulink is one of the main tools used in industrial practice. If MATLAB/Simulink was not just the most popular tool but the only tool, the research question would be simpler: “How do we improve MATLAB/Simulink?” For better or for worse, there is a myriad of other specialized tools also used by practitioners. The upside of this multiplicity is that it can be a source of inspiration. The downside is that it makes the search space of prior work vast and highly fragmented. If we are interested only in discrete-time simulation we may be able to focus on computer science and operations research venues. For continuous time simulation, almost every discipline of science has its own literature. For hybrid continuous/discrete systems, the literature becomes somewhat sparse, but locating related work remains challenging.

For the goals of transparency of models and user control it is natural to focus on mathematical notation as the syntax for models. After all, mathematics is the de facto lingua franca across many technical disciplines. But to make a lingua franca of human discourse into a mechanical formalism is a significant challenge. It helps, of course, that mathematics is a rigorous domain, but mechanical formalization rests not just on meaning but also on syntax. Mathematical notation includes multiple syntactic notations for the same concept, as well assignments of multiple different meanings to the same notation. Navigating the space of possibilities to identify what is intuitive for novices and acceptable across domains is a key language design challenge for this project. Another is defining mechanical interpretations (semantics) for such notations. Often, especially at the boundary of integrating continuous and discrete mathematics, we are able to define an interpretation for two constructs independently, but it is not obvious how to define an interpretation that allows both to be used together. One of the most profound challenges is to understand the mathematical space of meanings (or solutions) for mathematical expressions (problems).

With respect to the correctness of simulation tools (part of the toolchain coherence problem), a fundamental question is whether it is at all possible to find or develop rigorous methods for all aspects of simulation. In particular, working rigorously with just real numbers (not to mention functions over reals) introduces known computability and decidability issues [20]. This means that even computability is a fundamental challenge for the development of Acumen. A more nuanced (but equally important) question is whether, when such methods exist, they are precise enough, fast enough, and have all the computational properties that are needed to make for practical tools.

From the point of view of programming languages research, a core challenge is how to manage the high-dimensional problem of language design. In particular, a DSL that has not yet been fully specified has a large number of degrees of freedom. Basic examples include: syntax, semantics, user interface, documentation and tutorial materials, intended user base, language design and development team, and intellectual property and licensing issues. While there is significant literature, tools, and advice on many of these aspects individually or in combination with some others, literature addressing all these aspects simultaneously is sparse. Programming languages methods are readily available to identify which part of a language is broken and how to fix that part; but the more profound question of what language should exist and how to create it does not seem to have been sufficiently investigated.

In terms of teaching, the practical challenge is that, as programming languages researchers, we have limited contact and direct and regular access to students in CPS-related domains. Of course, at the time the project started, there were no CPS programs as such. Practical methods need to be found to address this challenge, so that it is possible to develop a concrete understanding of the needs of different types of students and potential users, as well as to have a basis for evaluating and quantifying the success of Acumen.

4 Approach and Implementation

In this section we summarize the approach, milestones in the effort to implement it, and key lessons learned in the process.

An agile research method: Given the complexity of the task of realizing the goals of Acumen, it was accepted from the outset that decomposing the process into clearly isolated technical problems may not be effective. For example, separating the process into selecting a semantics, a type system, and a syntax or even the reverse order would not be practical, and may not even be possible. There is a strong interdependence between these choices, as well as in other aspects of the design goals. It was therefore accepted that the development of Acumen would be a highly iterative and adaptive process that involves creating prototypes that would allow us as designers to gradually understand the space of technical design choices as well as to gain a better understanding of user needs and abilities.

Key features of the approach include frequent interaction between design (or design-critique) and implementation and close collaboration with potential users, especially domain-experts in domains that intersect with an evolving notion of the user base and novice users (often students). These continual activities allow us as the designers to gradually develop:

A clear understanding of the CPS domain in terms of technical needs from modeling and simulation tools, A portfolio of concrete example models for evaluation of the language and validation of hypotheses about actual engineering processes, A clear notion of a user base including size, interests, and expertise, A clear understanding of the semantic foundations and technically feasible functions that modeling and simulation tools can provide, A model of how to plan and manage a research-oriented, open-source effort that is manned primarily by researchers, students, and volunteers.

For lack of a better term, this approach can be described as an "agile research method", borrowing from the "agile" software development [5] literature. There is a similar emphasis on maintaining enough structure or "scaffolding" to enable reasonable testing of new ideas at all times, as well as on understanding the "customer" of the software application and engaging continually in the gradual development of the technical requirements for the final product.

Given the complexity in simultaneously developing these expertise, an obvious risk is spreading resources too thin. The key mechanisms for mitigating this risk have been careful consideration of all the possible language development initiatives (exploration or even addition of seemingly trivial but attractive features), and keeping the language and its implementations as small and simple as possible to reduce the effort of maintaining it.

Milestones and lessons: We followed the above method from the beginning of work on Acumen (in 2007) to the present. For reasons of space, we focus here on the second and current prototype of Acumen.

In 2010, and within three to six months, the first version of the current, Scala-based, implementation was created. It included a GUI, a simple editor, a reference interpreter and an optimized interpreter, and support for automatic plotting. Work on Acumen then took a brief break due to a gap between the end of one research project and the start of a new one. In 2011, work by a masters student introduced support for 3D visualization [24]. Students taking the first instance of our CPS course [21], which used Acumen, showed a strong preference to using the implementation that supported 3D visualization, as it made working with virtual CPS design problems noticeable easier than looking only at plots of individual signals.

In 2012, work began on one of the most significant technical results of the effort to date: developing a method for the correct simulation of Zeno systems [12]. This included the implementation of the first rigorous, enclosure-based simulator for a subset of Acumen. A problem that faced the implementation, and which was addressed around the same time, was the responsiveness and intermittent crashing issues with the GUI. Although the fix was seemingly minor, substantial effort went into understanding, diagnosing, and designing the fix. As programming languages researchers from a more "theoretical" background, it was startling to realize that concurrency problems in GUIs are not a solved problem, even in Javas SWING library. Difficulties in maintaining the integration between the core language interpreters and the interactive GUI proved to be a constant sink of engineering effort throughout the project.

As the activity under new funding was ramping up, there was a concerted effort to find ways to support more concurrent development on the code base. To avoid dependency on network connectivity and a centralized repository, a decision was made to move from svn to git. After some initial experience with github, we found that it had only limited support for external/internal visibility and "issue management", a decision was made to move to paid services by Atlassian (bitbucket and JIRA). These tools have played an essential role in facilitating development by a growing team, and in documenting key open challenges and the rationale for resolutions made.

In 2013 there was a concerted effort to expand the portfolio of case studies. This included developing low-order models of vehicle dynamics [16] by a domain-expert (with a recent PhD in mechanical engineering, specializing in robotics). Improvements visible to the user were made in the GUI and included syntax highlighting and code completion. Internally, with the version management and issue management systems in place, regression testing and continuous integration were introduced. This paved the way to exploring the use of property-based testing. In particular, a generator for random Acumen programs was built. It was not based solely on syntax, but on trying to exhibit interesting behaviour in order to catch problems when using the generated program to compare two semantics. Due to performance issues, it has so far only been used on a small scale. Building this infrastructure did draw our attention to the importance of carefully planning property-based testing in the context of computationally intensive codes.

There were also early efforts to explore the development of a compiled implementation; to study the mapping of Acumen to hybrid automata; and to allow the user to manipulate the textual models indirectly by manipulating the 3D rendering of objects. Such explorations were seen as premature by the project leadership, but it was nevertheless approved as it was in strong alignment with the interests of the researcher who wished to pursue it. It remains in the code base but is not actively supported. For a variety of reasons, such investment in activities with uncertain outcomes are a necessary part of any team effort, and we hope that there will be opportunities for capitalizing on the experience gained from these efforts at a later time.

A significant effort was made in 2014 to research licensing options, and the feasibility of shifting from GPL to BSD. The Acumen development team was fortunate to have a researcher with expertise in open source licensing. The effort included understanding both the needs/expectations of different contributing institutions, as well as the libraries available to enable such a migration. Activities directly visible to users included developing a new optimized implementation, more accurate integrators, including line numbers in error messages, and completing support for vectors and matrices. Development of the second-generation enclosure-based interpreter also began the same year. A practical problem with ease of installation was solved, namely, the migration from Java3D/OpenGL to jPCT [1]. In particular, the former requires the separate installation of a non-standard library - an additional step that was not easy for novice users, and can be seen as time consuming by potential expert users. The quality of 3D rendering was affected, but some interesting new possibilities were introduced. On the administrative front, a command line interface was introduced to provide researchers with the ability to access/configure most of Acumens features through command line, making Acumen scriptable.

In 2015, the development of two generations of enclosure interpreters had enabled the formalization of the enclosure semantics. This was done using the techniques of both denotational semantics (which formalized the notion of a solution to a hybrid systems model in the first place) and operational semantics (which provided a concrete way to compute rigorous over-approximations for these solutions). Contemporaneously, the methods for processing partial derivatives and bounded quantification (needed for the Euler-Lagrange equation) were formalized. There were also two key engineering efforts, namely, support for real-time 3D animation and for real-time input from external devices such as smart phones.

5 Emerging Design

As noted earlier, the purpose of the iterative/agile approach taken is to accumulate knowledge relating to multiple complex questions. For reasons of space it is not possible to cover all that has been learned about these different questions. However, we can briefly describe the central concrete artifact of the project, which is the language design emerging from this process.

Syntax: As noted earlier, the emerging core syntax includes guarded equations, where equations can specify either behaviors continuous over time or discrete (discontinuous) transitions over time. Expressions in equations can include arithmetic operations and derivatives. This way, the language can express ordinary differential equations with discontinuities. Partial derivatives are allowed, but only if they can be eliminated through symbolic differentiation at compile time. Until recently, equations had to be directed. Recently, through the introduction of symbolic Gaussian elimination, it is possible to relax this requirement and allow users to express models as undirected equations as long as the system can automatically direct them through Gaussian elimination at compile time.

Semantics: Values in Acumen are all functions of real-valued time, and their co-domain can be atomic constants (strings), reals, vectors, and matrices. In addition, the language supports an object-like notion of a sub-model [6], which can be instantiated to model the creation of an object dynamically at a certain point in simulation (logical) time, and which can also be terminated. Models are hierarchical in that any sub-model instantiated dynamically by a particular model is considered a "child" model. Communication only occurs between "parent" and "child" models. Models can, however, be moved from one parent model to another. These choices where made in part to facilitate the automatic parallelization of models.

Support for 3D visualization is provided by allowing any model to contain a special variable _3D, which can be equated to special constructors that result in the display of 3D objects in a particular pane in the GUI.

The implementation supports a "traditional" semantics which uses traditional (non-validated) numerical methods. This is the most complete and most widely used semantics, and suffices for basic educational uses. This implementation has played a crucial role in allowing us to explore the design space and to converge on an expressive, minimal syntax for modeling hybrid systems, as well as semantics for solutions. For example, it allowed us to make an early decision to support the notion of "super-dense" time, first introduced in the verification literature [15], and later advocated by Edward Lee [14].

The implementation also supports an "enclosure" semantics, which is intended to produce rigorous over-approximations (guaranteed upper and lower bounds) for all simulations [12]. This is the semantics that we would like Acumen, ultimately, to provide. While the current implementation of this semantics is not perfect, our recent work on the theoretical foundations for this semantics provides evidence that it is, at least in principle, feasible. The enclosure interpreter supports the use of intervals (closed, compact, connected sets on the reals) in source programs. As representations of sets (or, in computer science terminology, non-determinism), enclosures and intervals enabled rigorous analysis of systems with uncertain parameters. This was particularly important for the collaboration with partners from the automotive industry [16, 8].

So far, the introduction of a static type system in the implementation has been actively avoided. This choice is only a temporary one, to facilitate results on implementation techniques and semantic foundations, and to maintain a low entry barrier to the language. The language is seen as being in a phase where requirements for the type system are still being gathered.

6 Conclusions

This paper presented the design goals for the Acumen, explained the intrinsic challenges in achieving such ambitious goals, and articulated the research method taken to advance towards these goals. The complex interdependence between the individual goals is representative of the challenges that face the ambitious efforts to advance the state of the art by both the CPS and IoT communities. At the same time, the progress made to date gives us hope that this approach, which is characteristic of the philosophy of both communities, can be effective at enabling advances that would not possible through more narrowly focused efforts. The paper also presented several lessons learned in the context of specific activities in the development of Acumen.

One insight that cannot be placed in the context of a particular milestone relates to the gap between CPS and IoT. It appears that there is a gap in foundations between CPS (especially based on computer science) and IoT (especially based on information and communication theory) that is echoed in a disconnect in tools that support non-deterministic uncertainty and stochastic uncertainty, respectively. Our awareness of this gap developed gradually as we made several efforts to model networking system and found non-deterministic modeling of uncertainty insufficient for typical problems in this domain. Stochastic methods seem necessary for many of those problems. At the tool level, stochastic models (such as Markov chains or queueing systems) can seem simpler or independent from non-deterministic methods. Semantically, however, stochastic models are more naturally built after non-determinism has been fully treated and notions of distributions can be overlayed on notions of sets. With this insight in mind, we anticipate that when distributions can be introduced into Acumen that this will help bring closer the disciplines of CPS and IoT.

At a more practical level, Acumen is at a stage where performance issues should be given priority. There are two distinct concerns relating to performance. The first is with enabling casual users to experiment with larger models that can be simulated easily (and for many applications, usefully) using the traditional (non-validated) semantics. The second is with enabling rigorous simulation in reasonable time. We expect that the latter will be essential for user acceptability of the ultimate results of the Acumen development effort. A particular problem for the enclosure semantics that will also need to be addressed is to improve error messages. They are currently less intuitive compared to the traditional semantics, mainly because of the less familiar (and maybe less intuitive) process used for computing enclosures. Finally, we are also working on applying well-understood mathematical techniques for producing more precise over-approximations.

Acknowledgement. Numerous colleagues have contributed to the development of Acumen. We thank in particular the students of the CPS course at Halmstad University.

7 References

1. The jPCT web page. `http://www.jpct.net`. Accessed: 2015-08-10.
2. Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar, and Insup Lee. Modular specification of hybrid systems in charon. In *Hybrid Systems: Computation and Control*, pages 6–19. Springer, 2000.

3. Ann Arbor. CarSim reference manual, ver. 6.03, 2005.
4. Modelica Association et al. Modelica–a unified object-oriented language for physical systems modeling. *Language Specification, Version*, 2, 2005.
5. Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.
6. Paul Brauner and Walid Mohamed Taha. Globally parallel, locally sequential: a preliminary proposal for acumen objects. In *Proc. of the 9th Workshop on Parallel/High-Perf. Object-Oriented Scientific Computing*, page 2. ACM, 2010.
7. James Coplien, Daniel Hoffman, and David Weiss. Commonality and variability in software engineering. *Software, IEEE*, 15(6):37–45, 1998.
8. Adam Duracz, Henrik Eriksson, Ferenc Ágoston Bartha, Yingfu Zeng, Fei Xu, and Walid Taha. Using rigorous simulation to support ISO 26262 hazard analysis and risk assessment. In *12th IEEE Intl. Conf. on Embedded Software and Sys.*, 2015.
9. Abbas Edalat and Dirk Pattinson. Denotational semantics of hybrid automata. *The Journal of Logic and Algebraic Programming*, 73(1):3–21, 2007.
10. Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*.
11. Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems. Proceedings. IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.
12. Michal Konečný, Walid Taha, Jan Duracz, Adam Duracz, and Aaron Ames. Enclosing the behavior of a hybrid system up to and beyond a zeno point. In *Cyber-Physical Systems, Networks, and Applications, IEEE 1st Intl. Conference on*, 2013.
13. Lawrence Lessig. Code is law: On liberty in cyberspace. *Harvard Magazine*, 2000.
14. Xiaojun Liu, Eleftherios Matsikoudis, and Edward A Lee. Modeling timed concurrent systems. In *Concurrency Theory*, pages 1–15. Springer, 2006.
15. Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Real-time: theory in practice*, pages 447–484. Springer, 1992.
16. Jawad Masood, Roland Philippsen, Jan Duracz, Walid Taha, Henrik Eriksson, and Christian Grante. Domain analysis for standardised functional safety: a case study on design-time verification of automatic emergency braking.
17. Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
18. André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In *Automated Reasoning*, pages 171–178. Springer, 2008.
19. Walid Taha. A gentle introduction to multi-stage programming. In *Domain-Specific Program Generation*, pages 30–50. Springer, 2004.
20. Walid Taha and Robert Cartwright. The trouble with real numbers. In *INFORMATIK 2011*, page 325. Bonner Köllen Verlag, 2011.
21. Walid Taha, Robert Cartwright, Roland Philippsen, and Yingfu Zeng. A first course on cyber physical systems. In *Workshop on Cyber-Physical Sys. Edu.*, 2013.
22. Walid Taha, Veronica Gaspes, and Rex Page. Accurate programming: Thinking about programs in terms of properties. *arXiv preprint arXiv:1109.0786*, 2011.
23. Zhanyong Wan and Paul Hudak. Functional reactive programming from first principles. In *ACM SIGPLAN Notices*, volume 35, pages 242–252. ACM, 2000.
24. Yingfu Zeng, Chad Rose, Paul Brauner, Walid Taha, Jawad Masood, Roland Philippsen, Marcia O'Malley, and Robert Cartwright. Modeling basic aspects of cyber-physical systems, part ii. In *IEEE 11th Intl. Conf. on Embedded Software and Sys.*, 2014.