Proceedings of the 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop

David Broman and Gabor Karsai (Eds.)

CPS 2013

AVICPS 2013

Vancouver, Canada, December 3, 2013

Proceedings of the 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop

December 3, Vancouver, Canada

Editors
David Broman and Gabor Karsai

Series: Linköping Electronic Conference Proceedings, No. 90
ISSN (print): 1650-3686
ISSN (online): 1650-3740
ISBN: 978-91-7519-451-6
DOI: http://dx.doi.org/10.3384/ecp13090

Table of Contents

*) Position paper (4 pages)
****)** Research paper (8 pages)

Message from the Program Co-chairs

We would like to welcome you to the 2013 Analytic Virtual Integration Cyber-Physical Systems (AVICPS) workshop. The workshop is focusing on analytic techniques that enable the early discovery of defects in CPS, before the system is integrated or its parts are built. The principal objective is to present and discuss novel ideas and results that help to discover and resolve problems early during the design and implementation phases. AVICPS 2013 aims at bringing together researchers, engineers, and application developers from both industry and academia to present their latest advances in this field. Our program is organized according to three themes: mathematical fundamentals, model integration, and model analysis. The program also includes time reserved for lively discussions; we hope that all attendees will benefit from these interactions.

We received 13 submissions, from which 6 were accepted; 4 as position papers and 2 as full research papers. Our 13 internationally known PC members came from academia and industry and they have worked very hard to review the papers; most papers have received three reviews. We would like to thank the program committee members for their excellent work and for their suggestions in the selection of papers.

We would like to thank all those who submitted papers for their efforts and for the quality of their submissions. Thank you for your active participation in AVICPS 2013. We hope you will find this event to be productive and enjoyable, and we look forward to seeing you next year at the next AVICPS.

David Broman, UC Berkeley, USA, and Linköping University, Sweden
Gabor Karsai, Vanderbilt University, USA

Workshop Organization

Program Chairs

David Broman (co-chair)	UC Berkeley, USA, and Linköping University, Sweden
Gabor Karsai (co-chair)	Vanderbilt University, USA

Program Committee

Peter Fritzson	Linköping University, Sweden
Jérôme Hugues	Institute for Space and Aeronautics Engineering (ISAE), France
Russell Kegley	Lockheed Martin, USA
Henrik Nilsson	University of Nottingham, UK
Roman Obermaisser	University of Siegen, Germany
Linh Thi Xuan Phan	University of Pennsylvania, USA
Andre Platzer	Carnegie Mellon, USA
Franz Rammig	Univerity of Paderborn, Germany
Walid Taha	Halmstad University, Sweden
Stavros Tripakis	University of California, Berkeley, USA
Shige Wang	GM Research, USA
Michael Whalen	University of Minnesota, USA
Dirk Zimmer	DLR Oberpfaffenhofen, Germany

Some Challenges for Model-Based Simulation [*] [†]

Walid Taha[1,2] Robert Cartwright[2,1]

[1]IDE, Halmstad University, Sweden, Walid.Taha@hh.se

[2]Computer Science, Rice University, USA, corky@rice.edu

Abstract

Comprehensive analytical modeling and simulation of cyber-physical systems is an integral part of the process that brings novel designs and products to life. But the effort needed to go from analytical models to running simulation code can impede or derail this process. Our thesis is that this process is amenable to automation, and that automating it will accelerate the pace of innovation. This paper reviews some basic concepts that we found interesting or thought-provoking, and articulates some questions that may help prove or disprove this thesis. While based on ideas drawn from different disciplines, we observe that all these questions pertain in a profound way to how we can reason and compute with real numbers.

1. Introduction

It is widely anticipated that much of tomorrow's innovation will be in the form of cyber-physical systems, that is, systems that include computing, communicating, and physically dynamic components. A vivid example of such a system is a team of robots playing soccer, or a fleet of vehicles functioning collectively as an intelligent transportation system. Because we need to reflect on, reason and communicate about designs, modeling is an integral part of conceiving and developing new products in such domains. Many important problems defined in terms of mathematical models do not have closed solutions. As a result, simulation must be an integral part of the innovation process. Unfortunately, the formidable time and effort needed to convert analytical models to running simulation code can impede or even derail the process. Our thesis is that this transformation process can be more reliably and predictably automated, and that such automation can play a crucial role in training the cadre of future innovators and making them more productive.

Although a vast range of modeling and simulation tools already exists, automating the mapping from models to simulation codes remains a challenging and elusive goal. This is the case even for seemingly elementary domains such as rigid-body dynamics, which is a fairly simplified type of mechanical models that can be used to develop basic models of robot dynamics [7]. While this first work succeeds in identifying some basic problems and showing how programming language techniques such as partial evaluation can play a role in addressing them, the automation problem is larger than this. It cannot be addressed with a handful of research papers, and success in demonstrating this thesis may have far reaching consequences on the CPS domain. At the same time, entering the domain of cyber physical systems can be challenging, primarily because of the vast diversity of technical disciplines that relate to different aspects of this domain. This diversity of sources is an obstacle not just for researchers but also for education.

In an attempt to reduce the effort needed to overcome this problem, this paper reviews basic concepts from several related areas that we found interesting or insightful, and articulates some questions that may assist in investigating this thesis. While drawn from different disciplines, all of these observations and questions pertain to how we can reason and compute with reals.

Contributions: Modeling systems that involve real-valued and time-varying quantities is somehow at odds with traditional approaches to computing. Furthermore, it is not obvious precisely what needs to be done to integrate real numbers with these approaches. With the aim of shedding some light on what is missing, this paper reviews basic concepts and puts forth some questions about

- Traditional floating-point methods, and
- Basic properties of real numbers.
- Interval arithmetic.

Software engineering in general, and in particular programming languages semantics, design, implementation, and engineering have a lot to contribute to the study of cyber-physical systems. We hope that the observations and ques-

[*] This manuscript is a reduced and edited revision of an invited paper entitled "The Trouble with Real Numbers" and presented at the WS4C workshop of INFORMATIK 2011 held in Berlin.

[†] This research was supported by Halmstad University, the Swedish Knowledge Foundation (KK) Centre CERES, the Swedish Knowledge Foundation (KK) Environment at Halmstad University, and US NSF CPS awards number 1136099 and 1136104.

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop
December 3, 2013, Vancouver, Canada.

tions presented here encourage the reader to share this view.

2. Traditional Floating-Point Methods

Traditional numerical methods and simulation technologies have carried us a long way, allowing us to build airplanes, space ships, and many other advanced, as well as mundane, innovations. However, part of the difficulty in going from analytical models to simulation codes lies in the nature of the codes, which are built using traditional numerical methods techniques. Numerical methods for solving virtually any type of problem are highly varied and yield qualitatively different results when solving the same problem. Yet the user of these methods still has to bear the responsibility of choosing the right method for dealing with each different kind of model that they formulate and wish to simulate. This is a huge distraction for a user whose concern is to study the system that is being modeled, rather than how to implement the solver for different components of the system being modeled. Because it requires making a choice between more than two options for each component, it is a problem that has work-hour cost with an order of complexity exponential in the number of components. This is an optimistic estimate, given that testing whether a certain method works well for a certain type of component is usually done using a certain data set, and this result does not necessarily imply that it will work equally well for other data sets. Even with this simplifying assumption, it is a serious impediment to productivity.

An important question from the software engineering point of view is whether there is a way to hide these implementation choices from the user in such a way that the right one is always chosen (if it exists). A key technical challenge in approaching this question is to determine whether or not there exists a single method (or semantics) that can be viewed as a gold standard, and which can be used for both statically proving or dynamically checking the correctness of any given method. Since dynamic checking will always allow us to validate more methods than static techniques (due to the reduction in the number of quantifiers in the problem), the natural and important question to ask becomes whether there are also *universal numerical methods* that can enable the automatic selection of the right method for the given problem dynamically. The existence of such methods could be most insightful if they end up being simpler than many of the other prevailing methods, as it is likely that they would be revealing of some deeper ideas that are today only implicit in such methods.

Thus, traditional numerical codes only work correctly if certain assumptions about their inputs are satisfied, and these assumptions are not usually checked by the code. Furthermore, these codes generally do not raise any errors to indicate that the output that they produce is meaningless, nor do they generally confirm the level of accuracy in the answer that they produce. Traditional methods do involve careful analysis of how the values in the model are represented in the computation. But this analysis is generally done at the meta-level and not in the code. In essence, this approach allows the programmer to bake into the implementation ad hoc assumptions about how they expect the code to be used. We believe that it is these restrictions that result in significant loss in usability and reusability of numerical codes developed using traditional methods.

A large part of the analysis that must be carried out, and of the problems that arise when trying to use such codes, relate directly to the question of how real numbers are represented. Today, the vast majority of numerical codes are based on floating-point arithmetic (FPA). Again, floating-point technology has been very successful in that it enabled the development of numerous highly useful numerical codes that have enabled many CPS innovations. However, it is reasonable to consider alternatives. In particular, while FPA is well suited for hardware implementation, it remains in essence a static data structure that can represent only a finite set of values. So, rounding must be done with almost every arithmetic operation, which can introduce enough error that it can be extremely difficult to evaluate simple polynomial expressions (An example in [6]).

The most noteworthy feature of floating-point numbers is that they do not explicitly tell us how many digits (or bits) of the result are truly valid or *representative* of the real answer that such a computation should produce, if we were computing with some idealized form of real numbers. When we consider this feature together with the fact that numerical computations usually involve an extremely large number of operations, putting aside the problem that this feature creates for users of such codes, it is really impressive that any large numeric codes can be built correctly. Clearly correctness can only be achieved with significant meta-level reasoning. However, it remains an important question to determine how we can express the precise conditions needed to guarantee that a particular result of numeric computation is truly valid up to a given number of significant digits. If this is achieved, it may be possible to consider more ambitious questions, such as how to formally prove these theorems, or how to generate code guaranteed to be correct. In contrast to the overarching goal of mapping models to simulation codes, the last question is focused specifically on the issue of producing code that implements a real arithmetic computation using floating points correctly.

Stepping back from the questions of understanding traditional numerical techniques, it is important to note that, from a software engineering point of view, floating-point numbers are a somewhat curious choice for implementing real numbers. Specifically, they are a completely static data structure being used to represent values that, in general, may need an unbounded number of digits to represents. Interestingly, much care is needed if we wish to address this problem. For example, seemingly plausible alternatives such as variable precision numbers may simultaneously achieve less than what we might expect, and also bring more complexity to the problem than we started off with. Before we consider promising alternatives to floating-point numbers, it will be useful to recall some basic mathematical properties of numbers.

3. Basic Properties of Real Numbers

Real numbers are a concept so widely used in science and engineering to model both abstract and concrete systems that it is hard to imagine the world without them. Examples of real numbers include: the distance between two points in two- or three-dimensional space, the ratio between features of simple geometric forms such as the circle, and the integral of $1/x$ between two positive values of x. Much of the analytical component of the engineering and science disciplines rely heavily on this concept.

Real numbers derive both their power and their troublesomeness from their ability to transcend simpler forms of numbers, such as natural numbers, integers, and rational (fractional) numbers. For example, none of these sets are big enough to include numbers like π or e. It is therefore remarkable that scientific computing has been able to achieve so much while using a finite representation for real numbers, namely the floating-point representation.

Keeping in mind some basic mathematical theoretic properties of real numbers can help us navigate the complex space of alternative representations for real numbers. A basic example of these types of properties is cardinality, or the size of a set (See for example [1]):

- The set of different values that a floating-point number can take is finite,
- The set of rational numbers is countably infinite, and
- The set of real numbers is uncountably infinite.

It may seem that because rational numbers closer in cardinality to real numbers than floating-point numbers ("at least rational numbers are infinite") that they could make a more natural approximation of real numbers. But closer inspection suggests that they make it too easy to introduce unnecessary ad hoc decisions when trying to devise representations of real numbers. Indeed, rational numbers can be easily represented exactly on a computer through the use of dynamic data-structures. The basic arithmetic operators of addition, multiplication, and division for rational numbers can all be computed exactly on a computer. However, rational numbers are still insufficient for representing real numbers (a fact long known in mathematics [2]). When programming, this mathematical fact is reflected by the absence of an obvious way to compute with rational numbers in place of real numbers. This difficulty arises when we try to carefully explain why we cannot extend our set of operators to trigonometric functions, if we limit ourselves to rational numbers as a representation. Mathematically, the problem can be seen as the absence of a best rational number approximation for the result of the trigonometric function. It is tempting to take a pragmatic approach and make an ad hoc choice and choose *some* rational number to return when the result is not really rational, but then we begin to fall in the same traps that make it easy to abuse FPA to produce completely incorrect results. From the software engineering point of view, such choices are baking magic numbers into our code, which will eventually make it brittle and hard to maintain. Thus, rational numbers do not seem very well suited as a direct representation for real numbers, without building significant additional machinery around them. What is interesting here is that the consideration of the cardinality of the sets seems to provide the clearest indication of this mismatch, which is then echoed by a need for making ad hoc choices, when we try to build implementations. It is useful here also to be aware of the cardinality of various types of irrational real numbers, such as algebraic and transcendental numbers.

4. Interval Arithmetic

Interval arithmetic [5] is a powerful method for addressing one of the most basic problems in working with plain floating-point numbers: We can get information about the actual precision of the answer. Interval arithmetic uses two floating-point numbers to bound the exact answer of a real-valued computation from above and from below. Thus, interval arithmetic provides us with an immediate warning if rounding error has grown too large, rendering the result useless. Qualitatively, this additional information about precision is a huge improvement over allowing results to be silently corrupted by rounding. With this kind of information, the programmer or user can redo the full computation with a higher precision, and hope that this produces an answer with an acceptable level of precision.

Interval arithmetic is not a plug-and-play replacement for floating-point arithmetic. In fact, conceptually, interval arithmetic provides us with great concrete examples of why the floating-point way of doing business may in fact not be the way we will ultimately want to compute with real numbers. For example, the elementary operation of comparison on float-point numbers will have to behave differently when we move to intervals. How do we answer the question of whether the interval [1, 2] is less than [1.5, 2.5]? The answer cannot be yes or no, but rather, that the two are incomparable. As a result, it may not always be possible to expect that numerical algorithms can work without modification, using intervals rather than arithmetic.

It is interesting to note that a kind of abstract interpretation is almost built into interval arithmetic. It is not clear that this view has been fully developed (exceptions include [4] and [3]). Interval arithmetic gives rise to a beautiful theory that can teach us a lot about how we can compute effectively with real numbers.

A basic result of interval analysis is that performing a computation (that consists of the basic arithmetic operators) with more information about its inputs will always lead to a result that has no less information. Denotational semantics experts and domain-theorists will recognize this property as a notion of monotonicity. This property provides an elegant way to characterize well-behaved operators that we may or may not want to introduce into the language being interpreted using the primitives of interval arithmetic. An elegant observation from interval analysis is that monotonicity can occasionally provide a nice method for producing an answer with higher-precision without necessarily increasing the precision of the intermediate results. For example, because of monotonicity, we can always split the input interval into two overlapping parts, compute two results for the two parts, and merge them together. This can often produce an improved result, especially in cases where the computation suffers from what is often referred to as the dependence problem. Evaluating an expression such as $x-x$ with x equal to [1,3] does not produce [0,0] but rather [-2,2]. Operators such as addition have no way of knowing that there is a special relation between their two inputs. Splitting the input into smaller parts, however, helps us get closer to the most precise answer. For example, if we compute the expression with x equal to [1,2] we get [-1,1], and with x equal to [2,3] we get [-1,1], which is clearly more precise than [-2,2].

An unexpected feature several recent treatments of interval arithmetic is that they often resort to opening up the interval, computing with both endpoints, and putting them back in again. This is an example of breaking abstraction boundaries, and can easily lead to breaking the monotonicity property mentioned earlier. We may have encountered a related problem, which is finding a simple definitions of transcendental values and trigonometric functions that do not involve separately computing an expected value and an error term and then adding them together (that is, without breaking the interval abstraction). It also seems that algorithms such as an interval version of Euler's forward method for integration (which is needed for doing integration in the context of solving an initial value problem, for example) are things that the authors have been told do exist, but do not quite know how they work, or whether or not they are defined without breaking the interval abstraction.

While it may be counter-intuitive, interval arithmetic implementations usually do not use rational bounds. Instead, they typically use floating point numbers. The intrinsic nature of this observation can be illustrated by considering the computation $sin([0.9, 1.1])$. If we want the result to be a pair of rational numbers, then the only "right answer" would be the pair of rationals that are closest to the exact answer for $sin(0.9)$ and $sin(1.1)$ from the outside. But there are not two such best approximations of real numbers in general, and therefore, there is no ideal rational candidate. This observation is important for realizing that floating-point bounds are not just an implementation convenience for interval arithmetic, but rather a necessity. Because floating-point numbers have maximal degree of precision, the result of the above computation is well-defined, because there is always a best floating-point approximation to any real number.

We conclude this section with three questions. The first question is whether demand-driven iteration or incremental evaluation is inherent to the way we use interval arithmetic. For example, the most basic (albeit not the only) method for improving a result that we attain with interval arithmetic is to repeat the computation with a higher-precision floating-point representation for the bounds. What does this really tell us about the idea of interval arithmetic? It could mean that doing interval arithmetic forward is inherently iterative. Would it be useful to do it backwards, that is, in a demand-driven way?

The second question is what would constitute a well engineered and conceptually clear way to build an interval arithmetic library? Real analysis is usually not computationally effective; constructive analysis is, in a sense, effective, but not aimed at computing efficiently. How can we build up such a library in a way that would allow us to explain clearly and easily to students how they are expected to program with interval arithmetic? What is the right way to approach the problem of re-computing with higher precision? It is reasonable to expect that essentially the same question will also need to be answered for more sophisticated representations of real numbers.

The third question is whether there are high-level formulations for the computational problems and solution techniques that arise in this domain. In particular, it seems that important techniques such as interval arithmetic are typically constructed in ways that isolate some underlying floating-point infrastructure. While this appears perfectly sensible from the point of view of efficient implementation, it means that interval arithmetic is typically not built "from the ground up". Instead, it depends critically on the idea that floating-point arithmetic is the most efficient approach to implementing stream-based computation. It seems plausible that this could be the case in current architectures, but not for future ones. In particular, it is less obvious why this approach should be the most appropriate for other emerging architectures such as GPU, FPGAs, many-core systems, or for future microprocessor designs.

Acknowledgement

Michal Konecny and Alexandre Chapoutot provided valuable comments on drafts of this paper.

References

[1] Richard Beals. *Analysis: An Introduction*. Cambridge University Press, 2004.

[2] Georg Cantor. Über eine eigenschaft des inbegriffes aller reellen algebraischen zahlen. *Journal für die Reine und Angewandte Mathematik*, 1874.

[3] Alexandre Chapoutot. Interval slopes as a numerical abstract domain for floating-point variables. In *Static Analysis Symposium*, 2010.

[4] Eric Goubault and Sylvie Putot. Under-approximations of computations in real numbers based on generalized affine arithmetic. In *Static Analysis Symposium*, 2007.

[5] Moore, Kearfott, and Cloud. *Introduction to Interval Analysis*. SIAM, 2009.

[6] Warwick Tucker. *Validated Numerics: A Short Introduction to Rigorous Computations*. Princton University Press, 2011.

[7] Angela Yun Zhu, Edwin Westbrook, Jun Inoue, Alexandre Chapoutot, Cherif Salama, Marisa Peralta, Travis Martin, Walid Taha, Marcia O'Malley, Robert Cartwright, Aaron Ames, and Raktim Bhattacharya. Mathematical equations as executable models of mechanical systems. In *International Conference on Cyber-Physical Systems*, 2010.

Operational Semantics for a Modular Equation Language

Christoph Höger
Department of Software Engineering and Theoretical Computer Science, Technische Universität Berlin, Germany,
christoph.hoeger@tu-berlin.de

Abstract

Current equation–based object–oriented modeling languages offer great means for composition of models and source code reuse. Composition is limited to the source level, though: There is currently no way to compose pre-compiled model fragments. In this work we present $t^{(n,p)}$, a language which aims to overcome this deficiency. By using automatic differentiation directly in the language semantics, $t^{(n,p)}$offers the ability to implement index-reduction and causalisation of equation-terms without knowing their source-level representation. The semantics of $t^{(n,p)}$allow for calculation of arbitrary-order partial and total derivatives of pre-compiled terms.

Keywords Composition, Equation, DAE, Separate Compilation, Automatic Differentiation

1. Introduction

Equation based modeling languages offer great means for model composition and reuse. Unfortunately, this feature vanishes during classical model instantiation. There is no such thing as a Functional Mockup Unit ([1], a standard for composing compiled ordinary differential equations) for a differential algebraic equation (DAE).

The reason for this seems to be an implementation detail in the most common implementations: Usually e.g. a Modelica implementation will *interpret* the global model once and afterwards *compile* the resulting DAE into efficient executable code.

This "one-time instantiation" approach has some downsides:

- It favors a whole-model approach for static analysis: As there is no necessity for any interface abstraction, errors caused by wrong composition are only detected just prior to simulation.
- The code generation tends to *scale* badly in the case of large, uniform models [5, 13], because the symbolic manipulation of every equation leads to huge amounts of generated code.
- It prevents highly dynamic structural models: If a model *computes* (i.e. by a Turing-complete language) its succeeding mode, it is in general impossible to predict all modes of operation.

One way to overcome all those limitations is to find a way to compile equations *separately* into a form that can be instantiated arbitrarily often. Yet the attempt to do so reveals that the one-time approach is not just an implementation detail, since simulation often depends on the manipulation of the global system of equations. Thus, separate compilation needs to preserve the possibility to apply these manipulations.

As we will show, the main operation of index-reduction –differentiation of terms– can be implemented by automatic differentiation. Additionally, our method allows for sorting the compiled equations and solving them efficiently.

The rest of this paper is organized as follows: First we motivate the need for arbitrary-order differentiation and parameter selection of equations in a compiled setting. Afterwards we formally define $t^{(n,p)}$, a family of languages that allows for arbitrary order differentiation of mathematical terms, which we have implemented in a a general purpose DAE library called jdae[1]. We prove that the evaluation of this language indeed yields correct partial and total derivatives of a function. Finally, we show how $t^{(n,p)}$ can be implemented recursion-free.

1.1 Notation

In this paper we will use some short cuts to enhance the readability of formulas:

First, as we are talking about continuous functions, we will usually name them with the letters f, g, h. Domain and codomain are written in a blackboard style, i.e. $f : \mathbb{R} \to \mathbb{R}$ means that f is a function mapping real numbers to real numbers. Arguments are usually vectors ($\bar{x}$) or scalars (v). When the context is clear, we extend primitive operations over continuous functions: $(f + g)(v, \bar{x}) \equiv f(v, \bar{x}) + g(v, \bar{x})$. For function application we use a "flat" format: If $f : \mathbb{R}^{n+1} \to \mathbb{R}$, $v \in \mathbb{R}$ and $\bar{x} \in \mathbb{R}^n$, then we write $f(v, \bar{x})$ to denote the application of f to the concatenation of v to $\bar{x}$.

[1] https://github.com/choeger/jdae

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop
December 3, 2013, Vancouver, Canada.

2. Motivation

To understand the two most important operations on systems of differential and algebraic equations (namely index-reduction and causalisation), we resort to the classic higher index example, the cartesian pendulum:

$$x^2 + y^2 = 1 \quad (1)$$
$$\ddot{x} = Fx \quad (2)$$
$$\ddot{y} = Fy - 9.81 \quad (3)$$

As always, x and y denote the pendulum's coordinates while F is the tension force. In this example, we set the length of the pendulum and its mass to 1 to enhance readability.

2.1 Index Reduction

It is well known, that the above model cannot be solved directly by an ODE solver. The reason is obviously that both x and y appear differentiated but only one of them can be solved by equations 2 or 3, as one these equations is required to solve for F.

The solution to this problem is naturally to differentiate equation 1. By application of simple arithmetic laws the above model also implies:

$$\dot{x}x + \dot{y}y = 0 \quad (4)$$
$$\dot{x}^2 + \ddot{x}x + \dot{y}^2 + \ddot{y}y = 0 \quad (5)$$

As one can easily see, the augmented system of equations can be solved as an ODE (by choosing either x or y as state).

This result can be obtained by using an index-reduction algorithm like e.g. Pantelides' method [10], the dummy derivative method [8] or Pryce' method [11]. Any such method will result in the number of times a given equation or variable needs to be differentiated (explicitly as vectors d and c in Pryce' algorithm). Thus a compiled equation needs to be able to, given an arbitrary n, compute the n-th total derivative of itself.

2.2 Causalisation

In addition to the total derivative, it is usually necessary to compute the partial derivatives of an equation. This is due to the fact that most models will require to solve some algebraic parts iteratively. To do so efficiently, it is a common requirement to calculate the Jacobian of the equations.

In our example (assuming we choose x as state), we might solve y and $\dot{y}$ directly by equations 1 and 4 (as x and $\dot{x}$ are known by numerical integration), but finding a valid solution for F, $\ddot{x}$ and $\ddot{y}$ requires the iterative solution of equations 2, 3 and 5.

The process of finding such a partitioning of the system is called causalisation or equation-sorting. Given an index-1 DAE, the process is equivalent to the contraction of all strongly connected components in the dependency graph of equations.

This process raises an interesting problem in the setting of compiled equations: As the causalisation is applied after compile time, it is unknown for a given equation, which of the occurring variables are true iteration variables (e.g. F in the algebraic loop above) during simulation and which become constants (e.g. y).

2.3 Automatic Differentiation

In summary, a compiled equation needs to be able to compute for any order of total derivation and any subset of its variables the value and the partial derivatives of its residual. As we do not know neither the subset of variables nor the final degree of derivation, the only practical solution to this requirement is automatic (or algorithmic) differentiation. This technique is based on the fact that the derivation rules for primitive operations like addition or sine are well known (up to an arbitrary degree) and the chain rule shows how any composition of those primitive operations can be derived.

In this work we present a novel approach that embeds AD into a small term-language and prove its correctness. Our approach is hand-tailored to compute exactly the derivatives needed by a DAE solver. For any further reading about automatic differentiation we refer to section 7.3.

3. The term language t

In this section we define the simple term language t. We define language syntax in form of an EBNF-grammar. t is defined as:

$$\begin{array}{rcl} \tau & ::= & \mathtt{u}_{n,d} \\ & | & \tau \oplus \tau \\ & | & \tau \otimes \tau \\ & | & \phi\ \tau \\ & | & r \in \mathbb{R} \end{array}$$

t is a rather simple language: Terms (in the following sections abbreviated by variables τ_i) are either multiplication (written $\otimes$ to distinguish multiplication-terms from multiplication on real numbers which we will write as $\times$ further down), addition (written as $\oplus$), real values (r) and primitive functions $\phi \in \mathtt{P}$, where $\mathtt{P}$ is a not further defined set of n-times continuously differentiable (i.e. of class C^n) single-argument real-valued functions. Terms describing the application of a primitive function ϕ on a term τ are written as juxtaposition $\phi\ \tau$ while parentheses (i.e. $\phi(r)$ indicate the result of the actual application on real-numbers.

A notable feature of t is the availability of unknowns $\mathtt{u}_{n,d}$. Informally an unknown, written by $\mathtt{u}$ subscripted by two natural numbers, e.g. $\mathtt{u}_{n,d}$, represents the d-th total derivative of the n-th variable of a system of equations.

3.1 Semantics

The operational semantics of t is given below in the form of natural semantics. For an open interval $\mathbb{D} \subseteq \mathbb{R}$, a vector $\bar{x} = (x_1, \dots, x_p) \in (\mathbb{D} \to \mathbb{R})^p$ of functions of class C^n and a free variable $v \in \mathbb{D}$, the evaluation relation $\Downarrow$ is defined. We write $(\mathbb{D}, \bar{x}, v) \vdash \tau \Downarrow r$ to denote that under $(\mathbb{D}, \bar{x}, v)$ τ evaluates to r. Intuitively, $(\mathbb{D}, \bar{x}, v)$ denotes the domain of a ideal residual computation: $\bar{x}$ is the vector of unknowns, v is the independent variable and $\mathbb{D}$ is the interval upon which the model is well-posed (i.e. n-times continuously differentiable).

$\Downarrow$ is defined inductively by rules in form of a natural (or big-step) semantics. Each rule contains zero or more premises (the part above the bar) and one conclusion (the part below). The whole rule forms an implication: If the premises are true, the same holds for the conclusion. Multiple premises form an implicit conjunction. We also write side-conditions as part of the premises to save some space.

$$\frac{}{(\mathbb{D}, \bar{x}, v) \vdash r \Downarrow r} \quad (\textsc{Real})$$

$$\frac{(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \Downarrow r_1 \qquad (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \Downarrow r_2}{(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \oplus \tau_2 \Downarrow r_1 + r_2} \quad (\textsc{Add})$$

$$\frac{(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \Downarrow r_1 \qquad (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \Downarrow r_2}{(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \otimes \tau_2 \Downarrow r_1 \times r_2} \quad (\textsc{Mul})$$

$$\frac{r \in \mathbf{dom}(\phi) \qquad (\mathbb{D}, \bar{x}, v) \vdash \tau \Downarrow r}{(\mathbb{D}, \bar{x}, v) \vdash \phi\ \tau \Downarrow \phi(r)} \quad (\textsc{Prim})$$

$$\frac{n \in 0 \dots p}{(\mathbb{D}, \bar{x}, v) \vdash \mathtt{u}_{n,d} \Downarrow x_n^{(d)}(v)} \quad (\textsc{Unk})$$

Definition 1. *Let $f : (\mathbb{D} \to \mathbb{R})^p \times \mathbb{D} \to \mathbb{R}$ be an n-times differentiable function, then:*

$$(\mathbb{D}, \bar{x}, v) \vdash \tau \rightsquigarrow f :\Leftrightarrow (\mathbb{D}, \bar{x}, v) \vdash \tau \Downarrow f(\bar{x}, v)$$

(τ computes f)

To conclude this section, we state an observation about the well-formedness of terms of t. We call a term well formed (under $(\bar{x}, v)$) if there is an $r \in \mathbb{R}$ such that $(\mathbb{D}, \bar{x}, v) \vdash \tau \Downarrow r$. It should be obvious that the only form of non-reducing terms can occur due to E-Prim-Real or E-Unk. For the proofs further below, we eliminate this possibility by the following lemma (which could be easily shown, since t is a strict language):

Lemma 1. *Well formed terms always compute a function and terms that compute a function are well-formed.*

4. The $t^{(n,p)}$Family

To implement automatic differentiation, we lift the simple language t into a family of languages $t^{(n,p)}$.

$t^{(n,p)}$is syntactically very similar to t. Again, we see addition, multiplication, primitive functions and unknowns:

$$\begin{array}{lcl} \tau^{(n,p)} & ::= & \mathtt{u}_{m,d} \\ & | & \tau^{(n,p)} \oplus \tau^{(n,p)} \\ & | & \tau^{(n,p)} \otimes \tau^{(n,p)} \\ & | & \phi\ \tau^{(n,p)} \\ & | & \mathtt{A}\ \in \mathbb{R}^{(n+1,p+1)} \end{array}$$

The only visible difference to t is that the domain of real values is exchanged with a domain of real-valued matrices (which we abbreviate with capital latin letters). ϕ still denotes real-valued functions. The intuitive explanation is that a $t^{(n,p)}$-term calculates not only a value, but also the n total derivatives and the p partial derivatives (of every total derivative) of a function.

Two terms of different instances of $t^{(n,p)}$can only differ in the shape of the constants. It is important to note that if a term does not contain any constants, it is a valid $t^{(n,p)}$-term for any given concrete n and p.

4.1 Fundamental Definitions

Before we can explain the semantics of $t^{(n,p)}$, we need to introduce some helper functions. First, we define a lifting operator $\lceil\rceil_p^n$ that allows us to map a t-term into a $t^{(n,p)}$-term:

$$\begin{aligned} \lceil r \rceil^{(n,p)} &= \begin{vmatrix} r & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{vmatrix} \\ \lceil \tau_1 \oplus \tau_2 \rceil^{(n,p)} &= \lceil \tau_1 \rceil^{(n,p)} \oplus \lceil \tau_2 \rceil^{(n,p)} \\ \lceil \tau_1 \otimes \tau_2 \rceil^{(n,p)} &= \lceil \tau_1 \rceil^{(n,p)} \otimes \lceil \tau_2 \rceil^{(n,p)} \\ \lceil \mathtt{u}_{m,d} \rceil^{(n,p)} &= \mathtt{u}_{m,d} \\ \lceil \phi\ \tau \rceil^{(n,p)} &= \phi\ \lceil \tau \rceil^{(n,p)} \end{aligned}$$

Additionally we introduce two reduction operations on real-matrices called Δ (for differentiation) and I (for integration). The naming will be obvious once we define the operational semantics of $t^{(n,p)}$, but for now they are defined by elimination of the first and last row respectively:

$$I \begin{vmatrix} r_{0,0} & \cdots & r_{0,p} \\ \vdots & \ddots & \vdots \\ r_{n,0} & \cdots & r_{n,p} \end{vmatrix} = \begin{vmatrix} r_{0,0} & \cdots & r_{0,p} \\ \vdots & \ddots & \vdots \\ r_{n-1,0} & \cdots & r_{n-1,p} \end{vmatrix}$$

$$\Delta \begin{vmatrix} r_{0,0} & \cdots & r_{0,p} \\ \vdots & \ddots & \vdots \\ r_{n,0} & \cdots & r_{n,p} \end{vmatrix} = \begin{vmatrix} r_{1,0} & \cdots & r_{1,p} \\ \vdots & \ddots & \vdots \\ r_{n,0} & \cdots & r_{n,p} \end{vmatrix}$$

For the definition of the semantics as well as the proof of correctness, we will require a notion of mixed total and partial derivation. To reduce syntactical noise we will stick with the well-known d and ∂ notations, but omit any fractions:

$$\begin{aligned} \partial_0 f &= f \\ \partial_j f &= \frac{\partial f}{\partial x_j}, \quad i \in 1 \dots p \\ \mathtt{d}^{(i)} f &= \frac{\mathtt{d}^i f}{\mathtt{d} v^i} \end{aligned}$$

This notation ignores the partial derivative $\frac{\partial f}{\partial x_0}$. The reason for this decision is the matrix layout used further down: It is convenient to use the same index-set for the delta operator as in the matrix. To reconcile this restriction with our general mathematical model, we introduce the convention that $x_0(v) = v$, thus the missing partial derivative is identical to the first total derivative.

Matrix values will be abbreviated with capital latin letters, so $A \in \mathbb{R}^{n+1,p+1}$ in the context of $t^{(n,p)}$. When we index a matrix with one number, we select the corresponding row-vector, i.e. A_0 is the vector containing the elements of the first row of A.

To store partial and total derivatives, we introduce the n, p-dimensional derivative-matrix $\mathcal{D}^{(n,p)}(f, \bar{x}, v)$ of a function f.

$$\mathcal{D}^{(n,p)}(f, \bar{x}, v) \in \mathbb{R}^{n+1,p+1}$$
$$\mathcal{D}^{(n,p)}(f, \bar{x}, v)_{(i,j)} = \partial_j \mathtt{d}^{(i)} f(\bar{x}, v)$$

In case the context is clear, we omit the $\bar{x}, v$ arguments for brevity.

This definition enables the formulation of a correctness theorem on the evaluation of $t^{(n,p)}$-terms. It needs to be based on the fundamental assumption that a term actually computes a function. If this was not the case, the output of the reduction might still yield values but they are hardly meaningful in the sense of derivation.

Theorem 1 (correctness)**.**

$$(\mathbb{D}, \bar{x}, v) \vdash \tau \rightsquigarrow f \Rightarrow (\mathbb{D}, \bar{x}, v) \vdash \lceil \tau \rceil^n_p \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(f)$$

(the result of computing a function is the matrix of its derivatives)

The formal definition of $\Downarrow^{(n,p)}$ (the evaluation semantics of $t^{(n,p)}$as a binary relation between $t^{(n,p)}$-terms) follows below. As we will show, it fulfills this theorem.

We also introduce a special vector operator $\star : \mathbb{R}^{p+1} \times \mathbb{R}^{p+1} \to \mathbb{R}^{p+1}$, which is defined as follows:

$$(\bar{a} \star \bar{b})_0 = a_0 \times b_0$$
$$(\bar{a} \star \bar{b})_i = a_i \times b_0 + b_i \times a_0 \qquad i \in 1 \ldots p$$

The motivation of this definition lies in the following fact:

Lemma 2. *Let $g, h : (\mathbb{D} \to \mathbb{R})^p \times \mathbb{D} \to \mathbb{R}$ be differentiable functions with p parameters each, then:*

$$\mathcal{D}^{(0,p)}(g, \bar{x}, v) \star \mathcal{D}^{(0,p)}(h, \bar{x}, v) = \mathcal{D}^{(0,p)}(g \times h, \bar{x}, v)$$

A second operator, $\bullet : ((\mathbb{R} \to \mathbb{R}) \times \mathbb{R}^{p+1}) \to \mathbb{R}^{p+1}$, takes a differentiable function and a $p+1$ vector and returns a $p+1$ vector:

$$(\phi \bullet \bar{a})_0 = \phi(a_0)$$
$$(\phi \bullet \bar{a})_i = \phi'(a_0) \times a_i \qquad i \in 1 \ldots p$$

This composition operator is also motivated by an associated corollary:

Lemma 3. *Let $\phi : \mathbb{R} \to \mathbb{R}, g : (\mathbb{D} \to \mathbb{R})^p \times \mathbb{D} \to \mathbb{R}$ be differentiable functions, then:*

$$\phi \bullet \mathcal{D}^{(0,p)}(g, \bar{x}, v) = \mathcal{D}^{(0,p)}(\phi \circ g, \bar{x}, v)$$

4.2 Semantics

The operational semantics of $t^{(n,p)}$is also parametric over n and p. It is defined by the evaluation relation $\Downarrow^{(n,p)}$. Although the defining rules of $\Downarrow^{(n,p)}$ might seem fairly complex, the reader should notice how their structure reflects the derivation rules for addition, multiplication and composition of real valued functions.

$$\frac{}{(\mathbb{D}, \bar{x}, v) \vdash A \Downarrow^{(n,p)} A} \quad (\text{AD-REAL})$$

The evaluation of unknowns is now extended to also evaluate derivatives accordingly:

$$\frac{}{(\mathbb{D}, \bar{x}, v) \vdash \mathtt{u}_{m,k} \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(\mathtt{d}^{(k)} x_m, \bar{x}, v)} \quad (\text{AD-UNK})$$

Addition in $t^{(n,p)}$is simply defined as matrix addition:

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \Downarrow^{(n,p)} A \\ (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \Downarrow^{(n,p)} B\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \oplus \tau_2 \Downarrow^{(n,p)} A + B} \quad (\text{AD-ADD})$$

Multiplication in $t^{(n,p)}$is defined recursively over the parameter n.

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \Downarrow^{(0,p)} \bar{a} \\ (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \Downarrow^{(0,p)} \bar{b}\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \otimes \tau_2 \Downarrow^{(0,p)} \bar{a} \star \bar{b}} \quad (\text{AD-MULT-0})$$

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \Downarrow^{(n+1,p)} A \\ (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \Downarrow^{(n+1,p)} B \\ (\mathbb{D}, \bar{x}, v) \vdash \Delta(A) \otimes I(B) \Downarrow^{(n,p)} C \\ (\mathbb{D}, \bar{x}, v) \vdash \Delta(B) \otimes I(A) \Downarrow^{(n,p)} D\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \otimes \tau_2 \Downarrow^{(n+1,p)} \begin{vmatrix} A_0 \star B_0 \\ C + D \end{vmatrix}} \quad (\text{AD-MULT-N})$$

Composition (the application of primitive functions) is also defined recursively:

$$\frac{(\mathbb{D}, \bar{x}, v) \vdash \tau \Downarrow^{(0,p)} \bar{a} \qquad a_0 \in \mathbf{dom}(\phi)}{(\mathbb{D}, \bar{x}, v) \vdash \phi\ \tau \Downarrow^{(0,p)} \phi \bullet \bar{a}} \quad (\text{AD-COMP-0})$$

The general rule also relies on the definition of multiplication:

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \tau \Downarrow^{(n+1,p)} A \qquad A_{0,0} \in \mathbf{dom}(\phi) \\ (\mathbb{D}, \bar{x}, v) \vdash (\phi'\ I(A)) \otimes \Delta(A) \Downarrow^{(n,p)} B\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \phi\ \tau \Downarrow^{(n+1,p)} \begin{vmatrix} \phi \bullet A_0 \\ B \end{vmatrix}} \quad (\text{AD-COMP-N})$$

It remains to show that these rules are actually meaningful and compute the derivatives of a function *correctly*.

5. Correctness of $t^{(n,p)}$

Recall, that theorem 1 is defined about a t-term that is lifted into a $t^{(n,p)}$-term. Thus it can be proven by structural induction over t. To do so, we show that if the property holds for every sub-term of a term it also holds for the term itself.

Case 1 ($\tau \equiv r$). *By definition of lifting, we know that* $\lceil r \rceil_p^n = \left|a_{i,j}\right|$ *with* $a_{0,0} = r$ *and* $a_{i,j} = 0$ *if* $(i,j) \neq (0,0)$. *By definition of* $\rightsquigarrow$*, it holds that* f *must be the constant function* $f(\bar{x}, v) = r$ *and thus* $\partial_j \mathtt{d}^{(i)} f(\bar{x}, v) = 0$ *if* $(i,j) \neq (0,0)$.

Therefore: $a_{i,j} = \mathcal{D}^{(n,p)}(f, \bar{x}, v)_{i,j}$ *for any* $\bar{x}, v$ □

Case 2 ($\tau \equiv \mathtt{u}_{m,k}$). *We know by rule* AD-UNK *that:*

$$(\mathbb{D}, \bar{x}, v) \vdash \lceil \mathtt{u}_{m,k} \rceil_p^n \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(\mathtt{d}^{(k)} x_m, \bar{x}, v)$$

By definition of $\rightsquigarrow$ *it follows that* $f(\bar{x}, v) = \mathtt{d}^{(k)} x_m(v)$.

Therefore: $\mathcal{D}^{(n,p)}(\mathtt{d}^{(k)} x_m, \bar{x}, v) = \mathcal{D}^{(n,p)}(f, \bar{x}, v)$ □

Case 3 ($\tau \equiv \tau_1 \oplus \tau_2$). *We know by Lemma 1 and* $(\mathbb{D}, \bar{x}, v) \vdash \tau \rightsquigarrow f$ *that both* τ_1 *and* τ_2 *are well-formed (and compute a function each):*

$$(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \rightsquigarrow g \wedge (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \rightsquigarrow h$$

Application of the induction hypothesis to τ_1 *and* τ_2 *and insertion into* AD-ADD *yields:*

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \rceil_p^n \Downarrow^{(n,p)} G \\ (\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_2 \rceil_p^n \Downarrow^{(n,p)} H\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \oplus \tau_2 \rceil_p^n \Downarrow^{(n,p)} G + H}$$

$$\text{Where} \qquad \begin{aligned} G &= \mathcal{D}^{(n,p)}(g, \bar{x}, v) \\ H &= \mathcal{D}^{(n,p)}(h, \bar{x}, v) \end{aligned}$$

Since $G + H = \mathcal{D}^{(n,p)}(g + h, \bar{x}, v)$ *and (by definition of t)* $f = g + h$*:*

$G + H = \mathcal{D}^{(n,p)}(g + h, \bar{x}, v) = \mathcal{D}^{(n,p)}(f, \bar{x}, v)$ □

To prove the correctness of multiplication, we need to take two steps. First, we will show the correctness in case of $n = 0$. Afterwards we apply natural induction to show the general case.

Case 4 ($\tau \equiv \tau_1 \otimes \tau_2 \quad n = 0$). *As for the addition, we notice that by Lemma 1* $(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \otimes \tau_2 \rightsquigarrow f$ *implies that:*

$$(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \rightsquigarrow g \wedge (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \rightsquigarrow h \wedge f = g \times h$$

Therefore, because both terms are well-formed, we can insert the induction hypothesis into rule AD-MULT-0 *and get:*

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \rceil_p^0 \Downarrow^{(0,p)} \mathcal{D}^{(0,p)}(g, \bar{x}, v) \\ (\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_2 \rceil_p^0 \Downarrow^{(0,p)} \mathcal{D}^{(0,p)}(h, \bar{x}, v)\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \otimes \tau_2 \rceil_p^0 \Downarrow^{(0,p)} \mathcal{D}^{(0,p)}(g) \star \mathcal{D}^{(0,p)}(h)}$$

And by Lemma 2 and $f = g \times h$*:*

$(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \otimes \tau_2 \rceil_p^0 \Downarrow^{(0,p)} \mathcal{D}^{(0,p)}(f, \bar{x}, v)$ □

Case 5 ($\tau \equiv \tau_1 \otimes \tau_2$, general case). *In the general case, we make the same observation about* f, g *and* h*:*

$$(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \rightsquigarrow g \wedge (\mathbb{D}, \bar{x}, v) \vdash \tau_2 \rightsquigarrow h \wedge f = g \times h$$

Additionally, we know by the application of the structural induction hypothesis:

$$\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \rceil_p^{n+1} \Downarrow^{(n+1,p)} \mathcal{D}^{(n+1,p)}(g, \bar{x}, v) \\ (\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_2 \rceil_p^{n+1} \Downarrow^{(n+1,p)} \mathcal{D}^{(n+1,p)}(h, \bar{x}, v)\end{array}$$

As we have seen earlier, Δ *and* I *map matrices from* $t^{(n+1,p)}$ *into* $t^{(n,p)}$*:*

$$\begin{aligned}\Delta(\mathcal{D}^{(n+1,p)}(g, \bar{x}, v)) &= \mathcal{D}^{(n,p)}(g', \bar{x}, v) \\ I(\mathcal{D}^{(n+1,p)}(g, \bar{x}, v)) &= \mathcal{D}^{(n,p)}(g, \bar{x}, v)\end{aligned}$$

Application of the structural induction to τ_1 *and* τ_2 *yields:*

$$\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \rceil_p^{n+1} \Downarrow^{(n+1,p)} \mathcal{D}^{(n+1,p)}(g) \\ (\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_2 \rceil_p^{n+1} \Downarrow^{(n+1,p)} \mathcal{D}^{(n+1,p)}(h)\end{array}$$

When we set $G = \mathcal{D}^{(n+1,p)}(g)$ *and* $H = \mathcal{D}^{(n+1,p)}(h)$*, the natural induction hypothesis yields:*

$$\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \Delta(G) \otimes I(H) \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(g' \times h) \\ (\mathbb{D}, \bar{x}, v) \vdash \Delta(H) \otimes I(G) \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(h' \times g)\end{array}$$

If we apply these results to rule AD-MULT-N *we can see that:*

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \rceil_p^{n+1} \Downarrow^{(n+1,p)} G \\ (\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_2 \rceil_p^{n+1} \Downarrow^{(n+1,p)} H \\ (\mathbb{D}, \bar{x}, v) \vdash \Delta(G) \otimes I(H) \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(g' \times h) \\ (\mathbb{D}, \bar{x}, v) \vdash \Delta(H) \otimes I(G) \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(h' \times g)\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \otimes \tau_2 \rceil_p^n \Downarrow^{(n+1,p)} \begin{vmatrix} H_0 \star G_0 \\ \mathcal{D}^{(n,p)}(h \times g' + g \times h') \end{vmatrix}}$$

Thus we can apply calculus to the derivation of products: $\mathcal{D}^{(n,p)}(h \times g' + g \times h') = \mathcal{D}^{(n,p)}((h \times g)') = \mathcal{D}^{(n,p)}(f')$ □

A similar technique can be used to proof correctness over composition. Again, we start with the basic case $n = 0$:

Case 6 ($\tau \equiv \phi\ \tau_1 \quad n = 0$). *Again, Lemma 1 provides us with the guarantee that* τ *and thus* τ_1 *is well-formed:*

$$(\mathbb{D}, \bar{x}, v) \vdash \phi\ \tau_1 \rightsquigarrow f \Rightarrow (\mathbb{D}, \bar{x}, v) \vdash \tau_1 \rightsquigarrow g \wedge f = \phi \circ g$$

Therefore, we can apply the structural hypothesis and Lemma 3 to rule AD-COMP-0*:*

$$\frac{(\mathbb{D}, \bar{x}, v) \vdash \lceil \tau_1 \rceil_p^0 \Downarrow^{(0,p)} \mathcal{D}^{(0,p)}(g, \bar{x}, v)}{(\mathbb{D}, \bar{x}, v) \vdash \lceil \phi\ \tau_1 \rceil_p^0 \Downarrow^{(0,p)} \mathcal{D}^{(0,p)}(\phi \circ g, \bar{x}, v)}$$

and since: $\mathcal{D}^{(0,p)}(f) = \mathcal{D}^{(0,p)}(\phi \circ g)$ □

Case 7 ($\tau \equiv \phi\ \tau_1$, general case). *As usual we start with our well-formedness observation:*

$$(\mathbb{D}, \bar{x}, v) \vdash \phi\ \tau_1 \rightsquigarrow f \Rightarrow (\mathbb{D}, \bar{x}, v) \vdash \tau_1 \rightsquigarrow g \wedge f = \phi \circ g$$

This allows us to apply the structural induction hypothesis:

$$\begin{aligned} \tau_1 &\Downarrow^{(n+1,p)} G \\ G &= \mathcal{D}^{(n+1,p)}(g) \end{aligned}$$

Also, by definition of Δ and I:

$$\begin{aligned} \Delta(G) &= \mathcal{D}^{(n,p)}(g') \\ I(G) &= \mathcal{D}^{(n,p)}(g) \end{aligned}$$

If we apply our observations about Δ and I from case 5 to our natural induction hypothesis, we see:

$$(\mathbb{D}, \bar{x}, v) \vdash \phi'(I(G)) \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}(\phi' \circ g)$$

As we have already shown, multiplication is also correct. Thus (by application of the chain-rule, since ϕ is a single-argument function):

$$(\mathbb{D}, \bar{x}, v) \vdash \mathcal{D}^{(n,p)}(\phi' \circ g) \otimes \Delta(G) \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}((\phi \circ g)')$$

Applying both results to AD-COMP-N, *we can conclude that:*

$$\frac{\begin{array}{c}(\mathbb{D}, \bar{x}, v) \vdash \tau_1 \Downarrow^{(n+1,p)} G \\ (\mathbb{D}, \bar{x}, v) \vdash \phi'\ I(G) \otimes \Delta(G) \Downarrow^{(n,p)} \mathcal{D}^{(n,p)}((\phi \circ g)')\end{array}}{(\mathbb{D}, \bar{x}, v) \vdash \phi\ \tau \Downarrow^{(n+1,p)} \begin{vmatrix} \phi \bullet G_0 \\ \mathcal{D}^{(n,p)}((\phi \circ g)') \end{vmatrix}}$$

$\square$

With this proof completed, we can state that a $t^{(n,p)}$term can be evaluated to yield any wanted total derivative and all partial derivatives of a function. Naturally, the programmer remains responsible to ensure the correctness' premise, the correspondence between the implied (differentiable) function f and the given term t for the used interval $\mathbb{D}$.

6. Implementation

In this section we will assume that equations have to be solved in a concrete host language (like Java, C, Haskell etc.) and t-terms are compiled into this host-language before lifting.

6.1 jdae

$t^{(n,p)}$has been implemented in a general purpose DAE library called jdae[2]. jdae offers the means to implement models directly in form of Java-classes. Those classes can be composed in any possible way to define a system of equations at runtime. jdae then also offers facility to index-reduce and simulate the global model.

The choice fell to Java mostly because of its bytecode access that allows for a simple implementation of the runtime specialization. We assume that $t^{(n,p)}$can be implemented quite naturally in any other language like ANSI-C or C++ as well.

[2] `https://github.com/choeger/jdae`

6.2 Basic Operations

Lifting into $t^{(n,p)}$can be implemented directly rather easily: Most languages provide the means to dynamically create and modify a matrix of real-values at runtime (either as an array or a list of floating point numbers). Lifting a t-term can thus be implemented by introducing n and p as object-level parameters into the built-in operations addition, multiplication, composition, unknown loading and constant creation. So before one evaluates a $t^{(n,p)}$term for a given n and p in the host language, one simply passes these arguments towards the term.

Neither addition nor constants render any problems: Filling a matrix with zeros but for the top-most, left-most element should be as simple to implement as matrix addition.

Loading an unknown is a slightly more complex case. Here, the implementation carefully needs to provide the different derivatives, depending on the context. First, one needs to take into account that higher order derivatives might be present implicitly in a term: If for example, one has compiled the equation $a(v)^2 + b(v)^2 = 1$ and wants to evaluate it as a $t^{(1,1)}$ term, then the implicit presence of a' yields $\bar{x} = (a, a')$. So in that case, the result of loading a by $\mathtt{u}_{1,0}$ should yield:

$$\begin{vmatrix} a(v) & 1 & 0 \\ a'(v) & 0 & 1 \end{vmatrix}$$

Since $t^{(n,p)}$is a language for DAE-simulation, the values of a and a' are either already known or part of an iterative solution process. It is only important to correctly reflect the dependencies between different unknowns.

Implementing multiplication and composition is a different story, though. The definition of their semantics hints towards a recursive implementation strategy quite directly: If one already has to implement multiplication and composition as functions of n, then it is obviously not a problem to implement them recursively. So multiplication of a $t^{(n,p)}$term relies on the implementation of the multiplication of $t^{(n-1,p)}$ and so on.

Unfortunately, this approach is hardly efficient. Not only may it involve a large amount of data copying between the recursive calls, but it also avoids an elementary optimization:

If we take a look at the multiplication, we see that every direct application of AD-MULT-N, requires *two* recursive invocations. This would yield $\mathcal{O}(2^n)$ applications. On the other hand, we know by the General Leibniz rule that we should be able to compute the n-th total derivative of a composed function by means of a sum of n elements:

$$(f \cdot g)^{(n)} = \sum_{k=0}^{n} \binom{n}{k} f^{(k)} g^{(n-k)}$$

This can be achieved for $t^{(n,p)}$-terms as well: It is easy to observe that the *pattern* of primitive calculations $(+, \times)$ does not change depending on the numbers being multiplied:

Every field of the resulting matrix of a multiplication is calculated by a sum of products of the form:

$$(A \otimes B)_{i,j} = \sum_{(q,r,s,t) \in F_{i,j}} a_{q,r} \times b_{s,t}$$

$$i \in 0 \dots n, \in 0 \dots p$$

F can be computed recursively:

$$\begin{aligned}
F_{0,0} &= \{(0,0,0,0)\} \\
F_{0,j} &= \{(0,j,0,0),(0,0,0,j)\} \\
F_{i+1,j} &= A \cup B \\
&\text{where} \\
&\quad A = \{(q+1,r,s,t) | (q,r,s,t) \in F_{i,j}\} \\
&\quad B = \{(q,r,s+1,t) | (q,r,s,t) \in F_{i,j}\}
\end{aligned}$$

We omit a proof of correctness for this iterative implementation for brevity. It should suffice to state that F_0 is an implementation of $\star$ and the union of A and B reflects the addition in the conclusion of AD-MULT-N. The increase of the first and third indices are the iterative version of Δ.

If one *precomputes* F once for every combination of n and p, the implementation of $t^{(n,p)}$-multiplication can be handled inside a tight loop, increasing performance and decreasing memory usage. Note, that this precomputation does not need to occur prior to compilation, but can as well be done on link- or runtime. Additionally, algebraic optimization (e.g. merging of summands with the same factors) can be applied to the precomputed elements of F, finally yielding a result similar to Leibniz' formula.

A similar approach (leading to a variant of Faà di Bruno's formula) can be implemented for the composition case.

6.3 Runtime Specialization

A way to optimize the AD-operations even further is to think of the precomputed patterns F as a form of a tiny language that is interpreted at runtime to calculate the element of the result matrix. This view yields an interesting result: As every partial derivative in the result matrix is computed in the same way (only differing in the column), there are essentially only two different methods for every operation and every concrete n (namely to compute the n-th total derivative and any given partial derivative of it).

For any concrete n, the operations remain constant for any evaluation. So *after* index reduction we can create specialized methods for every required n (in fact, we can even create some beforehand, e.g. $n = 0$). This kind of *runtime specialization* allows us to eliminate the interpretative overhead generated by the application of the precomputed operations. So instead of calculating a sum e.g. by means of a for-loop, we can directly issue a sum-expression that contains all summands, etc.

7. Conclusion

We have shown that differential algebraic equations can be given a compositional operational semantics. This semantics can be used to compile equations (and thus models) separately and still simulate them efficiently.

7.1 Example

The Cartesian pendulum model from section 2 can be compiled to t as follows (using residuals and setting $x \equiv \mathtt{u}_{1,0}, y \equiv \mathtt{u}_{2,0}, F \equiv \mathtt{u}_{3,0}$) :

$$\mathtt{u}_{1,0} \otimes \mathtt{u}_{1,0} \oplus \mathtt{u}_{2,0} \otimes \mathtt{u}_{2,0} \oplus -1 \tag{6}$$

$$\mathtt{u}_{1,2} \oplus \mathtt{u}_{1,0} \otimes \mathtt{u}_{3,0} \otimes -1 \tag{7}$$

$$\mathtt{u}_{2,2} \oplus (\mathtt{u}_{2,0} \otimes \mathtt{u}_{3,0} \oplus -9.81) \otimes -1 \tag{8}$$

As we have seen, index-reduction of the model requires us to differentiate equation 6 two times. To do so, we simply replace it by the following $t^{2,p}$ equation:

$$\lceil \mathtt{u}_{1,0} \otimes \mathtt{u}_{1,0} \oplus \mathtt{u}_{2,0} \otimes \mathtt{u}_{2,0} \oplus -1 \rceil_3^2 \tag{9}$$

Using this method, all equations of the model can be compiled separately and instantiated according to index reduction, without using symbolic or numeric differentiation.

7.2 Consequences

$t^{(n,p)}$allows for modular semantics of compiled models: There is no need to fallback to symbolic differentiation at any time for the computation of derivatives. This allows to instantiate compiled equations in any context and greatly reduces the size of the generated code (avoiding the scalability problem of traditional implementations).

It also enhances the expressiveness of DAE modeling by including models with fully variable structure. At the same time it allows for a clean separation of modules during development of models, increasing safety and reliability.

7.3 Related Work

The principle of n-th order automatic derivation in this work is based on the multivariate automatic differentiation by Kalman [6], which in turn is a generalization of Rall's numbers [12]. Kalman's operators V, D and L inspired Δ and I in this work. An early prototype of this work was developed using an implementation (by the apache commons project [3]) of Kalman's Derivative Structures. Also the idea of precomputed basic operations for multiplication and composition is based on that implementation.

Yet the fact that Kalman computes *all* mixed partial derivatives induces a large inefficiency for our application case: Inevitably, the containing Derivative Structure would also compute $\partial_1^2 \partial_i, \partial_1 \partial_2 \partial_i$ and so on (which are all unneeded for the semi-explicit solution of a DAE). So from the perspective of automatic differentiation, this work is a specialized implementation of Kalman's technique to a certain problem-domain.

The opposite approach of the implementation of n-th order automatic differentiation is the infinite computation of total derivatives as Karczmarczuk demonstrated it in [7]. The elegant, recursive style in this work inspired the formulation of the operational semantics of $t^{(n,p)}$. Albeit, Karczmarczuk does neither handle partial derivatives nor provide an iterative implementation.

Modularity of modeling languages has been researched e.g. by Zimmer [13] and Furic [3]. The former preferred an

[3] `http://commons.apache.org/math`

after-compilation approach to "rescue" as much modularity as possible, while the latter restricts the modeling formalism itself in a way that renders most symbolic manipulation needless. Both approaches differ from this work, since $t^{(n,p)}$ allows to retain *complete* modularity without sacrificing certain modeling techniques.

Another strategy is to embed a modeling language into an established general purpose language as shown by Giorgidze and Nilsson in [4]. This can even go so far as to formally define a complete host language specialized for this task, as Broman's Modelyze [2]. Both lead to modular, formally defined modeling semantics (either by inheriting from an established language or by formally defining the host language).

The main difference to our work lies in the treatment of equations: There, they are handled as data structures in the host language and interpreted (or JIT-compiled) for simulation, while in our case they can be directly translated into terms of the target language.

Finally, it should be noted that variable structure systems have been researched for a while now. In [9], Mehlhase presents an approach for systems with finite (in fact small) amounts of modes, where the symbolic manipulation can be applied to each mode separately (yielding efficient simulation code for every mode). In [14], Zimmer avoids compilation completely and shows, how runtime index reduction can concisely express certain simulation scenarios.

8. Future Work

As simulation performance is paramount, it is an obvious research topic for $t^{(n,p)}$. Simulating a set of real-world models and comparing the performance to existing implementations should be an interesting field of study.

It is an open question, how efficient index-reduction can be applied in the case of structurally variable models. As this is a non-trivial problem, any efficient solution will probably make use of an incremental approach. But until now it remains unclear, how such an approach might be implemented.

t is obviously (due to the lack of recursion) not Turing-complete. Neither is any member of $t^{(n,p)}$. To overcome this limitation, one could extend $t^{(n,p)}$ to a Turing-complete language $e^{(n,p)}$, containing all elements of $t^{(n,p)}$ plus the simple lambda calculus, general recursion (e.g. via a fixed-point operator), non-strict conditionals etc.

For any such extension of $t^{(n,p)}$, evaluation would be defined as usual (i.e. in non-AD languages). Intuitively, the automatic differentiation still "works" as in the case of $t^{(n,p)}$. Yet, it remains an interesting question how one would formulate a corresponding proof.

A second possible extension is to think in terms of *modeling*: Here it would be interesting to see, how a model *computes* different $t^{(n,p)}$ terms. For instance one could easily define the derivation operator used in modeling languages by operating on $t^{(n,p)}$-unknowns.

Another interesting aspect of $t^{(n,p)}$ is that it opens the door for a deeper use of precompiled models in the style of FMI. Using $t^{(n,p)}$ it should even be possible to introduce the concept of external *equations* into a language like Modelica (i.e. equations that are completely hidden in a precompiled library without any limitations to their usage).

References

[1] Torsten Blochwitz, M Otter, M Arnold, C Bausch, C Clauß, H Elmqvist, A Junghanns, J Mauss, M Monteiro, T Neidhold, et al. The functional mockup interface for tool independent exchange of simulation models. In *Modelica'2011 Conference, March*, pages 20–22, 2011.

[2] David Broman and Jeremy G. Siek. Modelyze: a gradually typed host language for embedding equation-based modeling languages. Technical Report UCB/EECS-2012-173, EECS Department, University of California, Berkeley, Jun 2012.

[3] Sébastien Furic. Enforcing model composability in modelica. In *Proceedings of the 7th International Modelica Conference, Como, Italy*, pages 868–879, 2009.

[4] George Giorgidze and Henrik Nilsson. Mixed-level embedding and jit compilation for an iteratively staged dsl. In *Proceedings of the 19th international conference on Functional and constraint logic programming*, WFLP'10, pages 48–65, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Christoph Höger. Separate compilation of causalized equations -work in progress. In François E. Cellier, David Broman, Peter Fritzson, and Edward A. Lee, editors, *EOOLT*, volume 56 of *Linköping Electronic Conference Proceedings*, pages 113–120. Linköping University Electronic Press, 2011.

[6] Dan Kalman. Doubly recursive multivariate automatic differentiation. *Mathematics magazine*, 75(3):187–202, 2002.

[7] Jerzy Karczmarczuk. Functional differentiation of computer programs. In *ACM SIGPLAN Notices*, volume 34, pages 195–203. ACM, 1998.

[8] Sven Erik Mattsson and Gustaf Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal on Scientific Computing*, 14(3):677–692, 1993.

[9] A. Mehlhase. A Python Package for Simulating Variable-Structure Models with Dymola. In Inge Troch, editor, *Proceedings of MATHMOD 2012*, Vienna, Austria, feb 2012. IFAC. submitted.

[10] Constantinos C Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988.

[11] John D Pryce. A simple structural analysis method for daes. *BIT Numerical Mathematics*, 41(2):364–394, 2001.

[12] L.B. Rall. *The Arithmetic of Differentiation*. MRC TSR. Defense Technical Information Center, 1984.

[13] Dirk Zimmer. Module-preserving compilation of modelica models. In *Proceedings of the 7th International Modelica Conference, Como, Italy, 20-22 September 2009*, Linköping Electronic Conference Proceedings, pages 880–889. Linköping University Electronic Press, Linköpings universitet, 2009.

[14] Dirk Zimmer. *Equation-based Modeling of Variable-structure Systems*. PhD thesis, ETH Zürich, 2010.

Verifying Consistency Between Models

August Schwerdfeger Hazel Shackleton Steve Vestal
Adventium Labs, USA,
{august.schwerdfeger,hazel.shackleton,steve.vestal@adventiumlabs.com}

Abstract

Numerous aircraft development programs have suffered cost and schedule delays due in part to unplanned rework that occurred during integration and acceptance testing. Many of the errors that required rework can be traced back to inconsistencies between different specifications and models developed by or for different disciplines and suppliers early in the development process. We describe a novel method for specifying and verifying complex consistency properties between different kinds of models. This method makes use of a gray-box model integration framework and an SMT verification tool. We report on the application of this method to one specific challenge problem, verifying that a logical computer system architecture specified in AADL and a solid model specified in Creo together satisfy a particular consistency property.

Keywords model consistency, virtual integration, model integration, SMT, verification, defect detection

1. Introduction

Several aircraft development programs have suffered from cost and schedule overruns due to unplanned rework late in the project. Among the reasons cited for the A380 were problems with design configuration management between the different suppliers [1]. For the B787, delays during software and system integration were cited [3]. Delays in software and system integration were also cited as a problem for the F-35 [13]. Studies of software failures have reported that a significant proportion of defects are associated with interfaces between modules or between requirements and implementation rather than a design or coding error within a single module [11, 18, 19]. The System Architecture Virtual Integration (SAVI) project being conducted by a consortium of civil aircraft manufacturers and suppliers has identified verification of model consistency as a priority need [12]. A key goal of the SAVI program is analysis of models in early program phases to detect defects that currently remain latent until integration or acceptance testing, which they have estimated could save $400M in an aircraft development program [24].

One approach to assure consistency between different models is to define a Domain-Specific Language (DSL) and then generate different kinds of models for different kinds of analyses from a common specification. An example of this is the SAE standard Architecture Analysis and Design Language (AADL) for embedded computer system architectures. A variety of safety, security, performance, and behavioral models (and system integration and configuration data) can be generated from a common AADL specification. These models can then be analyzed by appropriate back-end tools. This avoids the need to manually generate these various models, which is what is done in the absence of an appropriate DSL and tool set. More importantly, consistency between these various models (and the integration code and data) is built-in to the generators. All are generated from a common input AADL specification. A correctness property of the tools is that the generated models (and integration code and data) are consistent with the input AADL specification and with each other.

A DSL approach only works if a domain is sufficiently bounded and understood to define such a DSL and widely enough used to merit the investment. In contrast, vehicle development is a multi-domain problem that involves computer architecture, solid, control, fluid dynamics, electrical, hydraulic, and many other kinds of models. In a study of ground vehicle development, BAE identified dozens of different modeling languages and tools and a hundred different developer roles [4]. The investment in legacy training, tools, and models is enormous. It has been estimated that the DoD has been spending billions of dollars annually on modeling and simulation [15]. Autodesk alone reports investing about $250M annually in R&D in their tool domain [2]. This paper deals with verifying consistency between different models from different domains, and our approach is based on the use of a model integration framework rather than a new DSL.

One technology that can detect a class of inconsistencies between models is type checking. Vehicles have been lost due to a simple units mismatch [6]. We earlier reported on a multi-view gray-box model integration framework called FUSED that includes a powerful, extensible, abstract type system [7]. In our own experience with a set of UAV models, we found one case of units mismatch and one case of frame-of-reference mismatch between different models.

However, type checking only detects one class of defects. In this paper we report on an approach that allows us to specify and verify consistency properties over the structures of different models. By model structure, we mean the objects and relationships between objects declared by the user in a model. Using the capabilities of our multi-view

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop
December 3, 2013, Vancouver, Canada.

gray-box model integration framework, we provide the system engineer with an abstract model structure type. This is a graph structure in which nodes and edges are typed. A structure for almost every kind of model can be represented in this format at some level of abstraction. The mapping from a specific model in a specific language to a graph that abstractly represents the structure of that model is defined when the modeling environment is initially integrated into the framework.

The overall goal is illustrated in Figure 1. Given two models that provide distinct views of a cyber-physical system that are used for different engineering purposes, the model integration framework can provide the system engineer with a gray-box view of each as an abstract structure graph. These abstract structure graphs are then imported into a Satisfiability Modulo Theories (SMT) environment. The SMT specification has "subscribe" statements that direct the framework to import the two model structure abstractions as a set of SMT declarations. The desired consistency property is manually specified in the standard SMT-LIB language. The class of consistency properties that we explore in this paper has the form that there exists a mapping from one model structure to another model structure that satisfies a set of typing and connectivity assertions. Intuitively, the consistency property we verified is that the processors, buses, devices, etc. specified in the avionics model map to corresponding solid objects with a corresponding connection topology. However, our long-term goal is general methods for specifying and verifying a broad class of complex consistency properties.

Examples of previous work to define and verify consistency between different models are synchronization between UML models [9], treating multiple models as extended projections of a central common model [22], consistency between different views and a base architecture [5], and traceability between models[16]. All of these share certain similarities. The mappings between models are partial, each may contain information that does not appear in and cannot be generated from any of the other models. Most use annotated graph representations of the static structure of models and define mappings between such graphs. Our approach does not require an expert in meta-modeling or formal languages or the integration framework to explicitly specify particular transformations or mappings or trace links between models. It uses concise specifications of desired properties and applies verification technology that automatically checks for the existence of a satisfying mapping between models. Our approach does not require an explicit single base architecture or common central view. It assumes models are written in existing languages (nine were used in an earlier series of UAV demonstrations[7]).

In the remainder of this paper, we will present the two models in our challenge problem: an AADL model of the logical software and hardware architecture for an aircraft mission system; and a solid model of the boards, enclosures, and cables for that system. We then present the SMT-LIB specification for a desired consistency property and a FUSED workflow specification that orchestrates the overall verification task. We conclude with lessons learned from this exercise and the results of some synthetic benchmarking experiments to assess scalability.

2. Models and Workflow

The two models that are to be verified consistent with each other are a 3D solid model specified using the commercial Creo (formerly ProE) tool and a logical avionics architecture model specified in AADL. Consistency means the two models satisfy a consistency property that we specify in the SMT-LIB language, which is a third model. Finally, there is a FUSED workflow specification that composes these three models to automate the sequence of actions necessary to perform the actual verification. FUSED workflows are themselves models, and this workflow specification is a fourth model. All of these models have an associated modeling environment, which is a particular modeling language supported by a particular toolset. These four environments are integrated into the FUSED multi-view gray-box model integration framework, which is necessary to execute a workflow automatically.

2.1 Equipment Solid Model

Solid models are three dimensional geometric models created by specifying 2D and 3D shapes and combining them to form more complex shapes, parts, and assemblies. Shapes (closed regions in 3D space) can be created by applying operations such as extrusion and rotation to 2D shapes. More complex shapes and parts can be specified by applying constructive solid geometry operations such as union, intersection and difference to simpler shapes. Geometric constraints can be specified to combine parts into assemblies and to combine parts and assemblies into increasingly more complex assemblies. A variety of properties can be specified for objects in the models, such as material and surface appearance properties. Most tools allow user-defined properties to be associated with solid objects.

Almost all the tools of which we are aware impose a hierarchical containment structure on the solid objects in a model. Every shape, part, and assembly (other than the root) is uniquely contained in one other shape, part or assembly. A containment tree view is provided to users to facilitate model navigation and object selection. Geometric constraints define relationships that may cut across this tree structure to form general graphs of relationships between solid objects. For example, parts in different subtrees can be aligned or mated using geometric constraints. Geometric constraints can create relationships between almost any pair of nodes in the containment tree structure.

There are standard exchange formats for solid models such as STEP and IGES, but model creation is done using a GUI unique to each tool vendor. Most commercial vendors provide a variety of solid model analysis capabilities, for example mass properties and structural analyses. Most tools provide an API for third-party tool developers.

Figure 2 is a rendering of the solid model for the simple avionics system used in this study. We developed our own solid model based on data sheets that could be downloaded from vendors of ruggedized embedded electronics modules and associated standards. Actual production models would contain more detail than we were able to include in our model, a subject to which we will return in the remarks section. (At least one vendor we talked to could supply detailed solid models, but only for purchased equipment under an NDA.)

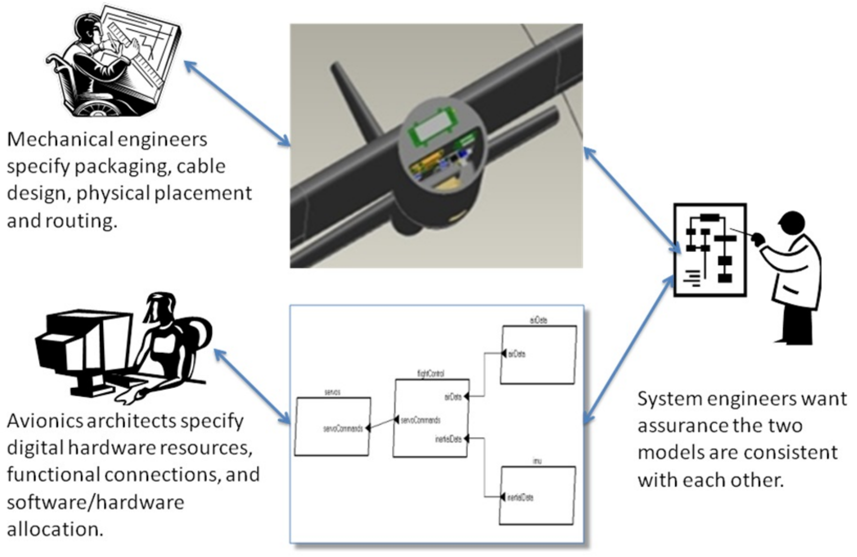

Mechanical engineers specify packaging, cable design, physical placement and routing.
Avionics architects specify digital hardware resources, functional connections, and software/hardware allocation.
System engineers want assurance the two models are consistent with each other.
airData
airData
flightControl
servos
airData
servoCommands
servoCommands
inertialData
imu
inertialData

Figure 1. The goal is to verify consistency properties between different kinds of models.

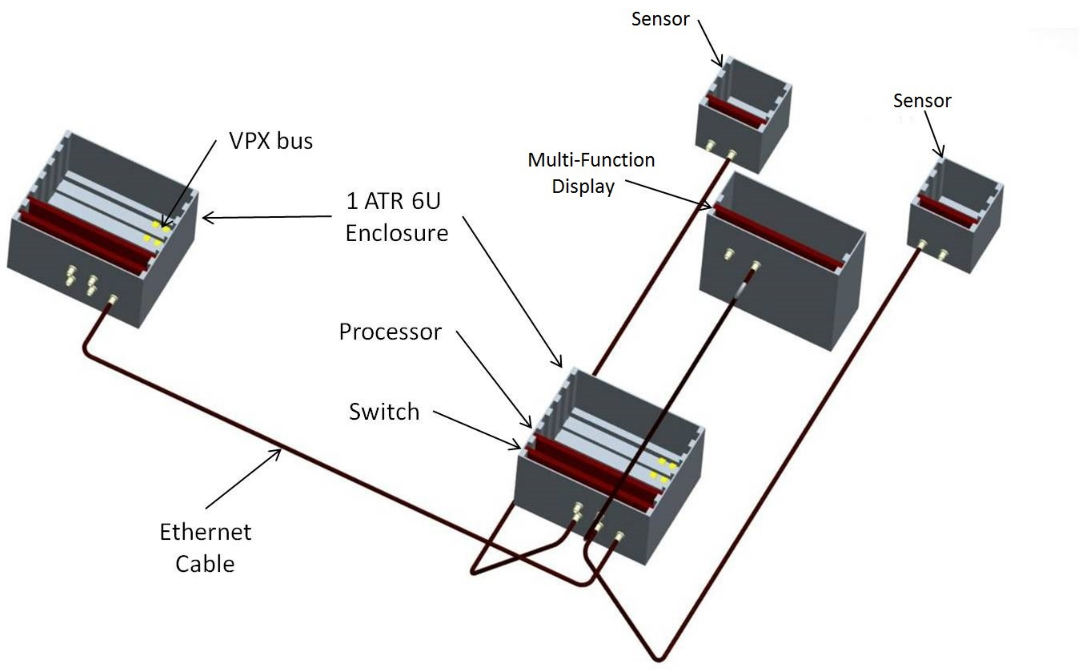


Figure 2. The 3D solid model shows enclosures, boards, and cables. Boards are inserted and cables connected by declaring geometric constraints. Special tool plug-ins collect electrical and type data.

These assemblies would normally be located (using additional geometric constraints) within an airframe solid model. Given the effort involved, placing these assemblies within an outer airframe model was not deemed significant enough for this exercise because the consistency property we used only applies to solid objects of particular types. We specified types for the objects in our model using a user-specified property, and only objects typed as devices, processors, switches, enclosures, cables, etc. were referenced by our consistency property.

Wiring harness design is a complex problem, e.g. routing, electromagnetics, reliability, installation, maintainability. Special add-in tools are available for most commercial solid modeling tools for the sub-domain of wiring harness design and placement. These specialized wiring harness add-ins capture electrical connectivity data and also add geometric constraints to the model. There are thus relationships between solid objects in the solid model that are typed as electrical connections.

We did not have a wiring harness add-in tool. We manually drew cabling using our own geometric constraint pattern to specify connections, shown in Figure 3. We added user-defined properties to explicitly identify electrical connections between solid objects. In our experiment we used the latter to identify electrical connection relations between solid objects (i.e. to strongly type certain relations between solid objects as electrical connections). In practice this could and should be done using only data that is already available from a wiring harness add-in without any additional user input.

2.2 Avionics Architecture Model

AADL is an SAE standard language used to model software and hardware architectures for embedded computer systems [20]. Software objects such as subprograms, objects, threads, processes, partitions, etc. and hardware objects such as memories, devices, processors, buses, etc. can be connected and bound to each other. There is a sophisticated typing system with extension, generic, and refinement capabilities. The language has a well-defined semantics and a large set of standard properties to enable a variety of analyses for performance, safety, security, behavior, etc. Tools also exist to generate system integration code and data (detailed specifications for individual components that are being integrated are usually written in some other suitable language, e.g. C, Ada, VHDL, SimuLink). The language includes packaging and hierarchical structuring features for components, data types, and connections. The standard defines textual, graphical, and tool interchange formats for AADL specifications.

We wrote our specification using the Open Source AADL Tool Environment (OSATE) [21]. Figure 4 shows three of the many diagrams from the graphical representation of the model. Hierarchical relationships are indicated in this figure by dotted lines that show which of the lower diagrams provides internal design detail for which boxes in the upper diagram. (In the actual tool, buttons and clicks are used to navigate up and down the design hierarchy between diagrams.) AADL specifications, in the graphical view, have the boxes-and-arrows look-and-feel of many modeling languages used in the computer field (but with well-defined semantics that enable a variety of analysis and generation tasks to be automated).

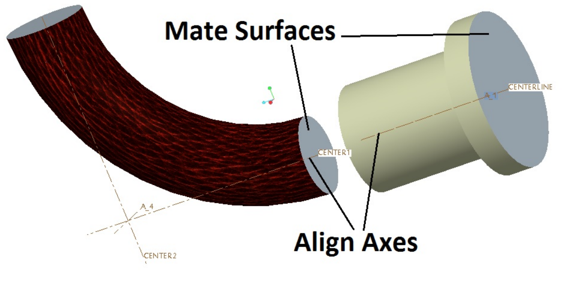


Figure 3. Connections are declared in the solid model using a geometric constraint pattern and properties on solids.

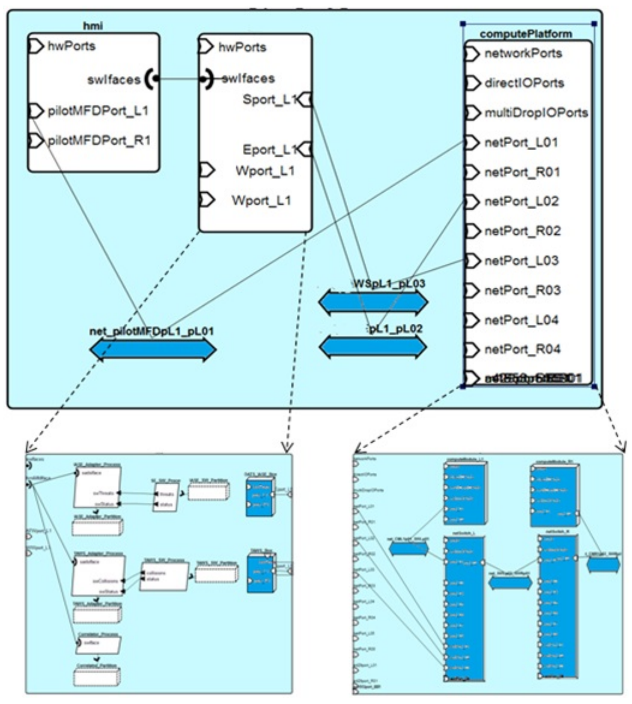


Figure 4. Three of the several diagrams in the graphical view of the AADL model. The dotted lines indicate where the lower diagrams show internal detail for boxes in the upper diagram.

The boxes that represent hardware objects have been shaded blue in the figure. AADL is a system language, and software and hardware objects are mixed in various places in a complex model. For example, in our model there is a common hardware computing platform shared by multiple hosted functions. Each hosted function consists of software subsystems together with special hardware equipment for that particular function, such as sensors. AADL is used to specify an integrated system in which multiple hosted functions are integrated onto the common computing platform both by binding software elements and by connecting specialized hardware. The overall hardware architecture is typically not contained in a single diagram. The hardware components are scattered among multiple specifications (typically developed by multiple suppliers) plus integration specifications developed by the system integrator.

An AADL specification is analogous to a set of class declarations. For example, a declaration of a processor called PowerPC is actually a specification for a class of processors. Multiple instances can be created from this declaration, for example by declaring multiple components of this class inside another declaration. A model for a specific system instance is obtained by "instantiating" a designated root system declaration in the AADL specification. This produces a data structure that has a specific set of instance objects that represent a specific system.

2.3 FUSED Models and Workflows

One way to better coordinate multidisciplinary modeling is to use an environment that supports capture of traceability links between different models in an environment [17]. Some engineering workflows that involve different kinds of models can be automated using a black-box model integration framework [25, 10]. In a black-box model integration framework, models are abstracted as functions from design parameters to quality metrics that can be evaluated at design time. Model integration frameworks allow the specification of "workflows" that automate engineering tasks that involve multiple different kinds of models. For example, a solid model can be "evaluated" to map choices of wingspan to vehicle weight and moment of inertia, which can in turn be used as input parameters of a dynamical systems model, which can then be solved or simulated to obtain performance metrics of interest to the engineer.

For our challenge problem, which requires a higher degree of visibility into the models being integrated, we used a prototype gray-box model integration framework called FUSED [7]. Among the goals for FUSED are improved specification and management of system (multimodel) configurations and design trade spaces, automated inference of traceability data from higher-level declarations of dependencies between models, interoperation with multiple model repositories and servers across multiple organizations, and more complex workflows such as combining multiple design automation environments (e.g. trade space visualization, multidisciplinary design optimization) with system models that are themselves compositions of many design models. Another example of a complex workflow is the subject of this paper, the use of a gray-box structural abstraction viewpoint into models to support automated verification of complex consistency properties.

The type system we are developing in FUSED combines concepts from programming language type systems (e.g. type constructors, multiple inheritance, interfaces, generics) and ontologies (e.g. description logics). All of this is done with the perspective that any single FUSED type is just one possible abstract viewpoint of some aspect or element of a model that has a more detailed and precise meaning and structure in the semantics and type system of its own language and modeling environment.

In the exercise reported here, several types play several roles. Both the solid modeling and the logical architecture modeling languages have their own type systems (e.g. primitive types of surfaces and shapes, types of relations such as part-of or electrical connection). At the FUSED level in this exercise, these domain-specific type systems are abstracted down to sets of enumeration literals (think of abstracting all types in a software program down to a list of class names). At the FUSED level in this exercise, we also use a graph type whose nodes and elements are labeled with these domain-specific type names. For example, the solid model is abstractly represented as a graph whose node labels name types of solid shapes and whose edge labels name types of solid relationships. (Inheritance in the domain-specific type systems is currently abstractly represented by allowing nodes and edges to have multiple type labels.)

The gray-box viewpoint used in this exercise could be graphically examined and navigated using a FUSED GUI by the engineer (who is developing the verification workflow specification). The nodes of the abstract model graph (a FUSED type of thing) represent objects statically declared in the model specification (e.g. component declarations but not objects that would be dynamically created during an execution or simulation of that model – those might appear in a different viewpoint). The edges of the graph represent relationships between objects that exist in a model. The nodes and edges are labeled with the names of domain-specific types declared in the detailed models. Figure 5 shows a graphical view of the abstract structure graph for the AADL model used in this exercise in which hardware (versus software and system) nodes have been shaded blue and connection (versus containment) edges have been shaded green (shading is used here in liu of displaying lists of domain-specific type labels).

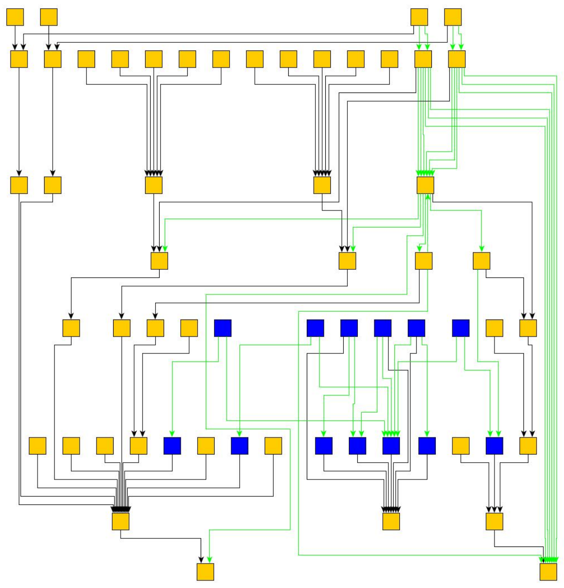


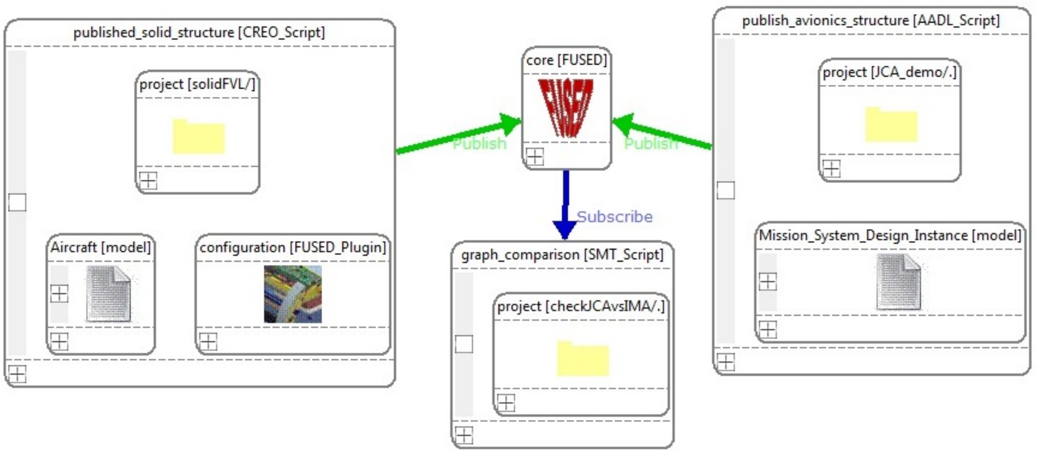
Figure 5. One layout for the abstract architecture view. Hardware objects are shaded blue, connection relations are shaded green.

The FUSED workflow used in this exercise is shown in Figure 6. This specification identifies the desired solid and architecture models and their configurations, specifies that abstract structural views are published for these two models and used to satisfy subscriptions (dependencies on external data) within a specified SMT model, and specifies that the SMT model is verified within the SMT modeling environment. The SMT model contains the specification of the consistency property to be verified.

2.4 SMT Consistency Property

The overall "proof structure" is to verify that a particular consistency relationship exists between two abstractions of the two models. The abstraction steps were done conventionally and without any formal definition or verification beyond strong typing of the data structures. The consistency relation is typed subgraph isomorphism. This is captured as an SMT-LIB specification and verified using an SMT checker.

Satisfiability Modulo Theories (SMT) tools can determine whether a model, as the term "model" is used by formal logicians in mathematics, exists for a specified set of many-sorted first-order logic predicates over an available set of theories [8]. The set of theories over which predicates may range can vary from tool to tool, but almost all support theories of integers, reals and arrays. SMT has been widely applied to formally verify properties for software. In this exercise SMT technology is applied for a novel purpose, to



published_solid_structure [CREO_Script]
project [solidFVL/]
Aircraft [model]
configuration [FUSED_Plugin]
core [FUSED]
Publish
Publish
Subscribe
graph_comparison [SMT_Script]
project [checkJCAvsIMA/.]
publish_avionics_structure [AADL_Script]
project [JCA_demo/.]
Mission_System_Design_Instance [model]

Figure 6. The FUSED workflow specification directs that abstract structural views be published by the solid and architecture model and provided to the SMT model, which is then executed to verify the consistency property.

formally verify a consistency property between the abstract structures of two different kinds of models. The consistency property desired is written in the SMT-LIB language by a subject matter expert familiar with this modeling and verification technology.

Figure 7 shows a redacted version of the consistency property specification used in this exercise. This can be broadly divided into two sections, the declarations for the abstract model structures to be checked and the declarations for the consistency property itself.

The declarations for the two abstract model structures are trivial for the SMT user. FUSED was built using language extension technology from the University of Minnesota [23, 14]. This makes it easy to integrate new modeling environments at a FUSED site. It also makes it easy to layer carefully defined new features on top of any modeling language at the time that modeling environment is integrated into FUSED. For example, this can allow type specification and checking not available in the original language (e.g. units, frames of reference). In this exercise, we layered a "subscribe" declaration on top of the SMT-LIB language. This new declaration allows the SMT specifier to identify data that is to be provided from an external source as specified by the system engineer in a FUSED workflow specification.

The FUSED framework will automatically perform type checking and representation conversion as required and expand the subscribe declarations into standard SMT-LIB declarations before invoking SMT tools. In the SMT-LIB language, nodes and types are declared as enumeration literals that are satisfied by an assignment of unique integers to them. Nodes are typed by a predicate over nodes and types. Edges are a predicate over pairs of nodes and an edge type. Nodes and edges may have multiple types, e.g. inheritance or casting in the original modeling language.

The second part of the specification must be manually written. This can be divided into two subparts. The first subpart consists of predicates that identify the nodes and edges (by type) for which a mapping must exist and specify type compatibility rules between nodes and edges that are mapped from one model to the other. The second subpart is the declaration of an array that maps nodes from one model to another plus a declaration that says this map must be a subgraph isomorphism where the nodes and edges satisfy the type compatibility rules.

The predicates that identify the types of nodes and edges that must be mapped and the type compatibility rules need to be modified for each new pair of models, but their structure is simple. The subgraph isomorphism declarations can probably be reused for a different pair of models with minor modifications.

The consistency relation between the two abstractions is typed subgraph isomorphism. Although theoretically a computationally challenging problem, graph isomorphism has been well-studied, and reasonably scalable algorithms exist. Adding the type compatibility constraints, pre-filtering the abstract graphs to nodes and edges having types of interest, and seeding with known mappings (e.g. "assert avionics leftCabinet maps to solid leftEnclosure") should further simplify the problem.

Our choice of SMT for this specific consistency property could reasonably be challenged as overkill. However, our hypothesis is that a variety of complex properties may emerge, and the number and nature of such properties is still a research issue. We selected SMT because it is a powerful general-purpose verification technology. A research question is whether a many-sorted first-order logic over a few fixed theories can easily express a wide class of useful properties.

3. Results

SMT performance was surprisingly good. Our challenge problem required about three dozen nodes and edges to be mapped. Verifying satisfiability or unsatisfiability required only a few seconds.

Most SMT tools provide their own languages and support unique features. There are multiple ways to encode a problem, especially when the set of alternatives considered include tool-specific language features. For example, some languages support special forms of declaration for enumerations. We conducted some preliminary synthetic benchmarking exercises, summarized in Figure 8, that both scaled the size of the problem and experimented with different formulations. There were some surprises, e.g. the

```
;; Subscribe to SMT declarations for elements of the abstract model graph type.
;; FUSED expands these to enumerations and functions for nodes, edges, and types.
(FUSED-subscribe Solid)
(FUSED-subscribe AADL)

(define-fun mapAADL ((ao AADL.Object)) Bool
    ;; return true if AADL.Object must have a mapping to a solid model object
))

(define-fun typeCompatible ((ao AADL.Object) (se Solid.Object)) Bool
    ;; return true if the avionics and solid object types are compatible
))

;; The model consistency property is true if a satisfying value for "map" exists.
(declare-fun map ()(Array AADL.Object Solid.Object))

;; The typed graph isomorphism property the map must satisfy.
;; For all avionics objects that must map to solid objects...
(assert (forall ((ao AADL.Object)) (=>  (mapAADL ao)
     (and
         ;; the map from avionics to solid model is 1-to-1
         (forall ((aoB AADL.Object))
              (=> (and (mapAADL aoB) (= (select map ao) (select map aoB))) (= ao aoB)))
         ;; if <ao,aoB> is an avionics edge that must map to a solid edge
         ;; then solid edge <map(ao), map(aoB)> with compatible type must exist
         (let ((so (select map ao)))
             (forall ((aoB AADL.Object))
                 (=> (and (AADL.attached ao aoB AADL.connection) (mapAADL aoB))
                      (Solid.attached so (select map aoB) Solid.Mate))))
         ;; avionics type of ao is compatible with solid type of map(ao)
         (typeCompatible ao (select map ao))))))
```

Figure 7. A redacted SMT consistency property specification used in this exercise.

use of special enumeration declaration forms could be less tractable than a more primitive declaration of literals plus a series of uniqueness assertions. We hypothesize that further experimentation in collaboration with practitioners skilled in the art could significantly expand the size of problem solvable. It is still an open question whether this can scale to problems of real-world size, but even if not, SMT may be useful to explore for consistency properties for which efficient tailored verifiers should be developed.

For practical use, it is essential that the method provide clear and useful indicators of why two models are inconsistent when the consistency property is unsatisfiable. The feed-back provided by the SMT tool in our demo was not directly useful when debugging model inconsistencies. SMT tools will identify unsatisfiable cores in such cases, but the output may in essence be the internal encoding, and the core may be an arbitrarily large subset of the entire specification. In our case, the tool would report that the entire set of assertions was unsatisfiable. We know that our SMT declarations of the abstract structure graphs are satisfiable in isolation. The unsatisfiabilities of interest are due to the nonexistence of a satisfying solution for the map array alone. It should be possible to use these properties to obtain useful error reporting.

Fully detailed solid designs are much larger and more complex than our model. Special-purpose add-in tools for wiring harness design are typically used. It would be an interesting future exercise to use a solid model with increased fidelity and publish/subscribe the wiring harness electrical analysis results view for use in consistency verification.

There are broader research questions that remain. How can we better formalize and verify the abstraction relations between the typed graphs and the detailed models? Can this approach specify and verify a usefully broad range of consistency properties that are effective in detecting defects that occur in practice? (We hypothesize this exercise could be adapted to any consistency property of the form "there exists a mapping between typed graph abstractions of model static structures that satisfies a set of properties.") Would applying these methods early in the process (for example to preliminary design models) significantly reduce defects that otherwise would remain latent until integration or acceptance testing? To address these questions, we are working to establish collaborations with developing organizations to perform studies using real-world models and defect data.

Acknowledgments

This work was supported by US Army AMCOM, Bruce Lewis, Army POC, bruce.a.lewis.civ@mail.mil.

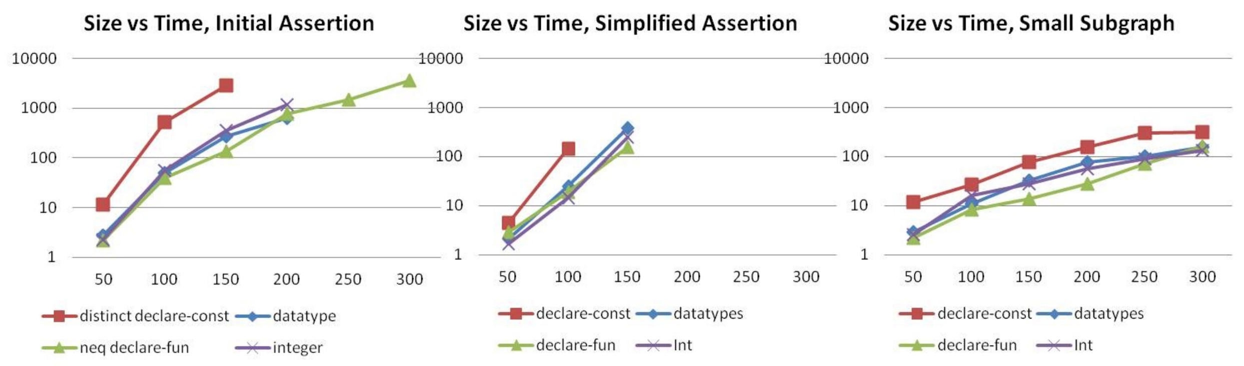


Figure 8. Initial synthetic benchmark results for various problem sizes and formulations were encouraging.

References

[1] Airbus A380. online, September 2013. http://en.wikipedia.org/wiki/A380.

[2] Autodesk Annual Report. online, April 2012. http://investors.autodesk.com/phoenix.zhtml?c=117861&p=irol-reportsAnnual.

[3] Boeing Reschedules Initial 787 Deliveries and First Flight. online, September 2013. http://www.boeing.com/news/releases/2007/q4/071010d_nr.html.

[4] Steven Bankes, Daniel Challou, David Cooper, Todd Haynes, Hillary Holloway, Paul Pukite, Jorge Tierno, and Christopher Wetland. META Adaptive, Reflective, Robust Workflow (ARRoW) Phase 1b Final Report. Technical Report TR-2742, BAE Systems, October 2011.

[5] Ajinkya Bhave, Bruce H. Krough, David Garlan, and Bradley Schmerl. View Consistency in Architectures for Cyber-Physical Systems. *International Conference on Cyber-Physical Systems*, 2011.

[6] Mars Climate Orbiter Mishap Investigation Board. Phase I Report, 1999. ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf.

[7] Mark Boddy, Martin Michalowski, August Schwerdfeger, Hazel Shackleton, and Steve Vestal. FUSED: A Tool Integration Framework for Collaborative System Engineering. *Analytic Virtual Integration of Cyber-Physical Systems Workshop*, 2011.

[8] David R. Cok. The SMT-LIBv2 Language and Tools: A Tutorial, March 2013. http://www.grammatech.com/resource/smt/SMTLIBTutorial.pdf.

[9] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM SYstems Journal*, 2006.

[10] iSight and the SIMULEA Simulation Engine. online, September 2013. http://www.3ds.com/products-services/simulia/portfolio/isight-simulia-execution-engine/latest-release/.

[11] David N. Card. Learning From Our Mistakes with Defect Causal Analysis. *IEEE Software*, January 1998.

[12] Peter H. Feiler, Jorgen Hansson, Dionisio de Niz, and Lutz Wrange. System Architecture Virtual Integration: An Industrial Case Study. Technical Report CMU/SEI-2009-TR-017, Software Engineering Institute, November 2009.

[13] GAO. Joint Strike Fighter Restructuring Places Program on Firmer Footing, but Progress Still Lags. Technical Report GAO-11-325, General Accounting Office, April 2011.

[14] Jimin Gao, Mats Heimdahl, and Eric Van Wyk. Flexible and Extensible Notations for Modeling Languages. *Proceedings of Conference on Fundamental Approaches to Software Engineering*, 2007.

[15] Paul Gustavson, Ali Nikolai, Roy Scrudder, Curtis Blaise, and Richard Daehler-Wilking. Discovery and Reuse of Modeling and Simulation Assets. online, September 2013. The M&S Journal, http://www.msco.mil/.

[16] Simon Frederick Königs, Grischa Beier, Asmus Figge, and Rainer Stark. Traceability in Systems Engineering – Review of industrial practices, state-of-the-art technologies and new research solutions. *Advanced Engineering Informatics*, 2012.

[17] Simon Frederick Königs, Grischa Beier, Asmus Figge, and Rainer Stark. Traceability in Systems Engineering – Review of industrial practices, state-of-the-art technologies and new research solutions. *Advanced Engineering Informatics*, 2012.

[18] Maggie Hamill and Katerina Goseva-Popstojanova. Common Trends in Software Fault and Failure Data. *Transactions on Software Engineering*, 1978.

[19] Robin Lutz. Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. *IEEE Requirements Engineering*, 1993.

[20] Architecture Analysis and Design Language. Technical Report AS5506, SAE, September 2012.

[21] OSATE 2. online, September 2013. https://wiki.sei.cmu.edu/aadl/index.php/Osate_2.

[22] Charles Simonya, Magnus Christerson, and Shane Clifford. Intentional Software. *OOPSLA*, 2006.

[23] Minnesota Extensible Language Tools, September 2013. http://melt.cs.umn.edu/index.html.

[24] Don Ward, Steve Helton, and Greg Polari. RoI Estimates from SAVI's Feasibility Demonstration, 2011. Systems Engineering Conference.

[25] Scott Woyak. Simulation Driven Design: Creating an Environment for Managing Simulation Tools, Processes, and Data. Technical report, Phoenix Integration, March 2010. http://www.phoenix-int.com/documents/pdf/white_papers/simulation-driven-design.pdf.

Towards a Safe Compositional Real-Time Scheduling Theory for Cyber-Physical Systems [*]

Linh Thi Xuan Phan
University of Pennsylvania

Abstract

Modern cyber-physical systems are becoming increasingly complex and distributed. These trends are making it more and more difficult to ensure the timing guarantees of these systems: traditional approaches were developed for much simpler systems and are difficult to scale. As system sizes are growing further, designing future cyber-physical systems is going to be even more challenging.

Compositional design and compositional analysis have emerged as an effective means to address this challenge. Several interface models and interface computation methods have been developed, which can be used to analyze complex systems in an efficient manner. The existing theories provide a foundation for ensuring the timing guarantees of cyber-physical systems; however, they also have several important limitations. This position paper discusses open challenges in this domain, and it highlights several research directions towards a safe and resource-efficient compositional theory for cyber-physical systems.

1. Introduction

Cyber-physical systems (CPS) are becoming increasingly complex and distributed at large scale. One way to scale timing analysis to such complex systems is to develop them in a compositional manner [39]: real-time workloads (such as tasks) are encapsulated in components, which expose their resource needs through resource-aware interfaces. These components can be composed under a scheduling algorithm or input/output interconnections to form larger components, and their interfaces can be computed efficiently via component abstraction and interface composition. Timing constraints of a component can then be guaranteed by ensuring that the platform satisfies the component's interface.

Several interface models and interface computation techniques have been developed (see e.g., [39, 9, 16, 27, 34, 38, 21, 14, 42, 30, 6, 10, 41, 20]), which enable efficient integration and isolation of independently-developed cyber-physical components. However, these existing theories face several important limitations: they ignore the platform overhead and the correlation between scheduling and communication, which could lead to timing violations in practice; they assume that nodes' clocks are closely synchronized, which is difficult and expensive to accomplish in a complex networked setting; they consider exclusively timeliness and ignore the semantics of timed interactions between components, or vice versa; and they focus only on the cyber aspects, while making implicit assumptions about the physical aspects that may not always be realizable.

Some of these assumptions are not unique to the current compositional theories; they come from the underlying schedulability and timing analysis. However, in large-scale distributed CPS, they are much more likely to be violated and can no longer be ignored. In this paper, we discuss these limitations in more detail, and we briefly sketch how the current theories could be extended to overcome them.[1]

2. Platform overhead matters

The current compositional theories assume a somewhat idealized platform in which all overhead is negligible. In practice, the platform overhead – such as release delay, preemption overhead, cache effects, context switches, and interrupt delay – can substantially interfere with the execution of tasks. Without considering such overhead, the computed interfaces can underestimate the components' resource needs; hence, the components can violate their timing constraints even if their interfaces are satisfied [35].

At first glance, it may seem that this issue can be solved by inflating the worst-case execution time (WCET) of each task by the overhead it experiences. However, this approach can be unsafe: including the overhead as part of the tasks' WCETs implies that the sources of overhead are assumed to be scheduled together with tasks, but this does not hold for certain types of overhead, such as interrupts and task release events. Further, it is difficult to compute a safe bound on the overhead experienced by a task, since such overhead accumulates with the number of tasks in the entire system[2], but the task-level details of one component is hidden from another component in a compositional setting.

Accounting for the platform overhead on multicore processors is even more challenging because of the complex interactions between various platform resources. In a compositional setting, the overhead a component incurs, e.g., due to cache interference, is harder to quantify: it depends not only on the direct interference between the component's own tasks but also on the indirect interference between these tasks and the interfaces of other components,

[*] This research was supported in part by the ARO grant W911NF-11-1-0403, ONR grant N00014-13-1-0802, and NSF grants CNS-1117185, ECCS-1135630 and CNS-1329984.

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop
December 3, 2013, Vancouver, Canada.
Copyright is held by the author/owner(s). The proceedings are published by Linkping University Electronic Press. Proceedings are available at:
`http://dx.doi.org/10.3384/ecp13090`
AVICPS website:
`http://www.analyticintegration.org/`

[1] Interface theories for ensuring functional correctness have been studied extensively (see e.g., [37, 32, 36]); in this paper, we focus primarily on interface theories from the scheduling perspective.

[2] This is because a task within a component may be delayed by the interrupt processing or task release events in other components.

and the latter in turn depends on the actual values of the interfaces and their implementations; consequentially, the interface computation becomes a cyclic process, which is often expensive and may not always converge.

In our recent work, we have proposed a new notion of overhead-aware interfaces, as well as methods for computing the interfaces that take into account platform overhead [35, 43]. While these initial solutions are promising, much work remains to be done to achieve a compositional theory that is both safe and resource-efficient in practice.

3. Data-dependent components

Existing compositional theories typically assume independent execution of tasks; in practice, however, CPS often operate on data flows with end-to-end timing constraints. Therefore, new interface models and interface analysis methods that can provide end-to-end timing guarantees for data-dependent and distributed components are necessary. Although there are a number of relevant formalisms that provide partial solutions to this problem, such as assume-guarantee interfaces [21, 14, 42], data flow graphs with LET semantics [30], and interfaces for network resource [38], none of them considers end-to-end timing constraints and compositional scheduling concurrently.

To meet the above needs, the component model and composition semantics need to be extended to capture not only resource sharing but also data communication semantics and input/output connections. Ideally, the component model should be self-sufficient for the analysis, i.e., it should contain all the information necessary for computing the interfaces, such as local timing constraints (e.g., local deadlines) and activation patterns (e.g., periods or arrival patterns) of tasks. However, such information about a component is difficult to obtain, since it depends on the local constraints assigned to other data-dependent components and, consequently, on those components' interfaces. In addition, deriving a composition semantics that encapsulates the intricate correlation between scheduling and communication is also non-trivial.

To illustrate the above cyclic relationship, consider an end-to-end data flow with end-to-end deadline D that is processed sequentially by two tasks, T_1 and T_2, which are located in components C_1 and C_2, respectively. Then, the deadline of T_2 is inverse proportional to the deadline of T_1. Further, the arrival pattern of T_2's input data – which is needed to compute C_2's interface – depends on the arrival pattern of T_1's output data, which in turn depends on T_1's deadline as well as the resource supply of C_1's interface.

One approach to tackle the above challenge is to adapt deadline decomposition methods, coupled with synchronization protocols, such as those outlined in [28]. This direction offers a self-sufficient component model, and it enables an efficient interface computation by directly applying existing results. However, it can also result in non-optimal interfaces, due to the cyclic relationship between the component model and the interfaces (discussed above). Therefore, enhancements of the deadline decomposition methods and synchronization protocols are required to improve the analysis accuracy.

Another interesting direction is to explore *parametric interfaces*, where an interface can be represented as a function of variables that denote unknown factors, such as local timing constraints, and the interface computation can be performed symbolically. The concrete values of the interfaces can then be realized at the top-level component based on the end-to-end constraints. Since the size of the composed interface grows with more composition steps, it would be useful to refine the interface, e.g., using a safe approximation of the interface, at each composition step to improve the analysis efficiency.

4. Clock synchronization

The current real-time scheduling theory and interface analysis methods assume a common notion of time. However, achieving strong synchrony in a distributed system is a known hard problem: due to clock drift and network propagation delays, the local clocks of the different nodes are always slightly different, and even frequent resynchronization (which would have a high overhead) would not be sufficient to achieve perfect synchrony.

Without perfect synchrony, the notion of a deadline becomes ambiguous. Since CPS interact directly with the physical environment, deadlines are usually given in terms of the physical time; however, when a control or data flow passes through multiple nodes, each node's local clock can deviate from the physical time, which can result in jobs being scheduled too early or too late; also, nodes can disagree whether a deadline has been missed or met.

At first glance, it may seem that timing variance due to clock drift is too small to matter in practice; however, even small discrepancies can cause scheduling anomalies and thus 'snowball' into large anomalies. For instance, suppose nodes A and B are expected to send messages to node C at a certain time to trigger tasks T_A and T_B on C, and suppose further that node A's message is meant to be sent first, so that T_A is released before T_B. If B's clock is slightly faster than A's, then B may send its message too soon, causing the messages to be reordered and T_B to be released and scheduled before T_A; as a result, T_A may miss its deadline, even though the schedulability analysis (which assumes perfect synchrony) may have predicted that the deadline would be met. Worse, since C's schedule is now different, the timing of C's own messages is also affected, which may lead to further changes on other nodes. Thus, while the original discrepancy on B is small, the resulting effect on the system as a whole can be much larger.

One approach towards solving this problem is to extend the system model with a bound on the clock drift and the length of the synchronization intervals, so that it becomes possible to reason about possible deviations from the reference time, and to make scheduling decisions accordingly. To enable this approach, component interfaces would have to be extended to expose information about drift and synchronization, and the information would have to be carried over when interfaces are composed and analyzed. To keep the complexity of the analysis manageable, it may be useful to 1) abstract some of the details in the interface, and/or 2) specify requirements for other subsystems that the component interacts with, such as an upper bound on the acceptable clock drift. One challenge is to determine good abstractions; another is to determine which requirements are useful and can be satisfied.

5. The gap between real-time scheduling theory and high-level formal models

Cyber-physical systems are traditionally modeled using two different paradigms: high-level models of computation and real-time task/resource models. These models capture different timing properties: The former focus on the high-level formal specifications of timed interactions, communications, and synchronization among a collection of independent processes or subsystems; examples include timed automata [4], I/O automata [29], and real-time process algebra [26]. The latter capture implementation-level task timing information (e.g., execution time, deadline, priority) and details on physical resources and resource sharing (e.g., processing speed, network bandwidth, memory, scheduling policy). Although both categories are intertwined, they are typically considered in isolation. On the one hand, verification techniques for timed concurrent models verify temporal properties based solely on the high-level model, without considering the platform aspects, such as communication delay and scheduling overhead (e.g., synchronization is assumed to be instantaneous). On the other hand, the task and resource models used in real-time scheduling theory are based on the execution platforms and the source-code of the software; unlike the high-level models, they do not take the semantics of communication into account. Thus, even if a higher-level property (e.g., a safety property) is proven in the higher-level model, it does not necessarily hold in the implemented system.

Several efforts have been made to bridge this gap by adding platform aspects to the high-level models. For instance, existing work on the implementability of timed automata incorporates the platform information in the timed automata model by explicitly modeling the execution platform [3] or by modifying the timed automata semantics to reflect the implementation platform, such as the sampling-based [23], almost ASAP [15, 11], time-triggered [22], and probabilistic and topological [8] semantics. Real-time scheduling has also been combined with timed automata in [2, 7] and with process algebra in [26, 31]. In addition, a number of automata- and actor-oriented scheduling interfaces have also been developed [6, 10, 41, 20].

However, the above approaches are not only expensive in terms of analysis complexity but also assume a very simple model of resources – for instance, the processor is assumed to have unit speed and be fully available. As we move towards multicore and distributed systems, these assumptions no longer hold: the processor is not always fully available due to various types of platform overhead (such as I/O, interrupts, communication), and these types of overhead can even vary between different nodes. We believe that it would be useful to develop new models and analysis techniques that combine both aspects.

One direction is to establish an intermediate 'glue layer' that connects the two classes of models, e.g., similar to the hierarchical heterogeneity approach for composing high-level models of computation in Ptolemy [17], or the functional mock-up interface for co-simulation and model exchange [1, 12]. The glue layer could precisely capture the assumptions (e.g., synchronization semantics) that the higher-level model makes about the platform, and it could be used to mechanically verify that a given platform satisfies these assumptions. The assumptions must be realistic (i.e., realizable on common platforms) but should abstract low-level details of the platform as much as possible. Since the assumptions may be relevant to other subsystems that the component communicates with, they should be taken into account by the interface analysis.

6. Analyzing state-based systems

While real-time scheduling theory provides a clean system abstraction and enables efficient analysis, it currently cannot be applied to scheduling state-based systems. For instance, a system might schedule tasks depending on the current state of their input buffers, or it might handle data differently based on its current buffer state (e.g., append new data items to an input buffer if there is room, and discard them otherwise). State-based scheduling is also inherent in adaptive systems, which need to respond to the external environment. For example, a video encoder component might send encoded video frames to a receiver component depending on how quickly the receiver can decode the video to avoid buffer overflows and buffer underflows. Although it is sometimes possible to derive a stateless approximation of the system, such approximation often leads to overly pessimistic analysis results.

A common approach to analyzing such systems is to use state-based models, such as timed automata [4] or event count automata [13], and to perform the analysis using formal verification. Here, if very small time steps are used, the resulting state space can be very large, and thus, the analysis can become expensive; hence, it is desirable to use large time steps wherever possible. However, large time steps can decrease accuracy, i.e., the analysis may fail for systems that are actually schedulable, or may succeed only with additional resources. It seems promising to apply abstraction refinement in these cases. Furthermore, some systems may contain a mixture of state-based and stateless components; to efficiently support this case, it would be useful to have a way to interconnect components of both types. It seems interesting to explore hybrid techniques, along the lines of [33] and [24].

7. Beyond resource-aware interfaces

The current compositional theories focus only on the cyber layer, such as timing and resource aspects, while making implicit assumptions about the physical system and the environment. As a result, failures may occur in the system, e.g., when these hidden assumptions are violated. To guarantee the safety and trustworthy of the system, it is critical to extend these theories beyond cyber concerns.

We believe that it would be essential to develop a notion of *safety-aware interfaces* to enable the compositional analysis of safety properties for CPS. For this, the component and interface models would need to be extended to capture not only the cyber aspects but also the physical aspects and the cyber-physical interactions of components, including e.g., the dynamics of the physical system, the control algorithm, safety-criticality levels of tasks, and safety goals of components under different environment conditions. One approach is to integrate control-theoretic multi-mode systems [25, 5] with a mixed-criticality extension of real-time multi-mode interfaces [34].

A challenge towards a safety-aware compositional theory is how to detect unsafe interactions between components, as well as to enforce their absences, during the interface composition. Undesirable interactions between components on the same cyber-physical platform can arise from multiple dimensions, such as via shared data and variables, via shared actuators and sensors, via computational and communication resource sharing, and via the physical environment. For instance, an unsafe interaction between an adaptive cruise control component and a collision avoidance component in an automotive system might arise via the physical environment when the former requests an increase in speed while the latter simultaneously requests a sharp turn, which could cause the vehicle to roll over.

New approaches for analyzing the coupling between control, safety and resource aspects are especially needed to detect undesirable interactions such as above. It seems useful here to adapt results on hazard analysis and safety assessment from the safety engineering domain (e.g., [18]) for the modeling and interface analysis. Further, as it may not be feasible to identify all interactions statically, it would be interesting to incorporate the idea of run-time monitoring and recovery [40] to enable automatic refinements of the interfaces and component interactions during run time.

Besides safety, there is a growing need for a *security-aware* compositional theory for CPS. This is highly challenging, as it may not be possible to 'decompose' the impact of an attack down to the component level, and it is also difficult to predict the attack behavior (e.g., of a malicious adversary). Existing work on compositional security [19] might provide useful insights to address this challenge.

8. Conclusion

As cyber-physical systems continue to grow in complexity, we believe that compositional approaches will remain to be effective for future CPS design and analysis. We have discussed in this paper several important existing limitations and research challenges in this research area. The list is not exhaustive, but through it we hope to inspire future research towards a safe, secure and resource-efficient compositional theory for CPS that is fully realizable in practice.

References

[1] Functional Mockup Interface. `http://www.fmi-standard.org`.

[2] Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *TCS*, 354(2):272–300, 2006.

[3] K. Altisen and S. Tripakis. Implementation of timed automata: An issue of semantics or modeling? In *FORMATS*, 2005.

[4] R. Alur and D. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.

[5] R. Alur, V. Forejt, S. Moarref, and A. Trivedi. Safe schedulability of bounded-rate multi-mode systems. In *HSCC*, 2013.

[6] R. Alur and G. Weiss. Rtcomposer: a framework for real-time components with scheduling interfaces. In *EMSOFT*, 2008.

[7] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: a tool for schedulability analysis and code generation of real-time systems. In *FORMATS*, 2004.

[8] C. Baier, N. Bertrand, P. Bouyer, T. Brihaye, and M. Größer. Probabilistic and topological semantics for timed automata. In *FSTTCS*, 2007.

[9] S. Baruah and N. Fisher. Component-based design in multiprocessor real-time systems. In *ICESS*, 2009.

[10] P. Bhaduri and I. Stierand. A proposal for real-time interfaces in speeds. In *DATE*, 2010.

[11] P. Bouyer, K. G. Larsen, N. Markey, O. Sankur, and C. Thrane. Timed automata can always be made implementable. In *CONCUR*, 2011.

[12] D. Broman, C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. Determinate composition of fmus for co-simulation. In *EMSOFT*, 2013.

[13] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *RTSS*, 2005.

[14] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *EMSOFT*, 2001.

[15] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost asap semantics: From timed models to timed implementations. In *HSCC*, 2004.

[16] A. Easwaran, I. Shin, and I. Lee. Optimal virtual cluster-based multiprocessor scheduling. *RTS*, 43(1):25–59, 2009.

[17] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity-the ptolemy approach. *Proc. IEEE*, 91(1):127–144, 2003.

[18] C. Ericson et al. *Hazard analysis techniques for system safety*. Wiley-Interscience, 2005.

[19] D. Garg, J. Franklin, D. Kaynar, and A. Datta. Compositional system security with interface-confined adversaries. *ENTCS*, 265:49–71, 2010.

[20] M. Geilen, S. Tripakis, and M. Wiggers. The earlier the better: a theory of timed actor interfaces. In *HSCC*, 2011.

[21] T. A. Henzinger and S. Matic. An interface algebra for real-time components. In *RTAS*, 2006.

[22] P. Krčál, L. Mokrushin, P. Thiagarajan, and W. Yi. Timed vs. time-triggered automata. In *CONCUR*, 2004.

[23] P. Krčál and R. Pelánek. On sampled semantics of timed systems. In *FSTTCS*, 2005.

[24] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *EMSOFT*, 2009.

[25] J. Le Ny and G. J. Pappas. Sequential composition of robust controller specifications. In *ICRA*, 2012.

[26] I. Lee, J.-Y. Choi, H. H. Kwak, A. Philippou, and O. Sokolsky. A family of resource-bound real-time process algebras. In *FORTE*, 2001.

[27] H. Leontyev and J. H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *RTS*, 43(1):60–92, Sep. 2009.

[28] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, 1st edition, 2000.

[29] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Info. and Comp.*, 185(1):105–157, 2003.

[30] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *RTSS*, 2005.

[31] M. Mousavi, M. Reniers, T. Basten, and M. Chaudron. Pars: a process algebra with resources and schedulers. In *FORMATS*, 2004.

[32] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis*. Springer, 1999.

[33] L. T. X. Phan, S. Chakraborty, P. S. Thiagarajan, and L. Thiele. Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In *RTSS*, 2007.

[34] L. T. X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *ECRTS*, 2010.

[35] L. T. X. Phan, M. Xu, J. Lee, I. Lee, and O. Sokolsky. Overhead-Aware Compositional Analysis of Real-Time Systems. In *RTAS*, 2013.

[36] B. C. Pierce. *Types and programming languages*. The MIT Press, 2002.

[37] J. A. Rowson and A. Sangiovanni-Vincentelli. Interface-based design. In *DAC*, 1997.

[38] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida. Multi-level hierarchical scheduling in ethernet switches. In *EMSOFT*, 2011.

[39] I. Shin and I. Lee. Compositional Real-Time Scheduling Framework. In *RTSS*, 2004.

[40] O. Sokolsky, U. Sammapun, I. Lee, and J. Kim. Run-time checking of dynamic properties. In *RV*, 2005.

[41] I. Stierand, P. Reinkemeier, T. Gezgin, and P. Bhaduri. Real-time scheduling interfaces and contracts for the design of distributed embedded systems. In *SIES*, 2013.

[42] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *RTAS*, 2006.

[43] M. Xu, L. T. X. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. D. Gill. Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms. In *RTSS*, 2013.

Early Phase Memory Leak Detection in Embedded Software Designs with Virtual Memory Management Model

Mabel Mary Joy[1] Wolfgang Müller[2] Franz J. Rammig[3]

[1,2]C-Lab, University of Paderborn, Germany, {mabeljoy,wolfgang}@c-lab.de

[3]Heinz Nixdorf Institute, University of Paderborn, Germany, franz@uni-paderborn.de

Abstract

Virtual platforms are gaining significant importance in early design tests of embedded software as it helps to redesign or optimize the system well advance in time and keeps flaws minimal in the production stage. As embedded system's size gets smaller, expensive resources like memory are limited. Hence memory needs to be managed efficiently and optimally. Memory leak is a serious issue that leads to wastage of expensive memory. We propose a novel approach to detect memory leaks in early design stages of soft real-time systems with no garbage collection. Our approach utilizes a virtual platform modeled in SystemC at an abstract level using Transaction Level Modeling. The software under test is run on top of this model. Potential memory leaks in the software are detected by applying a novel hybrid method combining both static and dynamic approaches. In early design stages where a real execution environment and complete executable software are unavailable, a simulation environment and a virtual platform are necessary. Virtual platforms provide flexibility to change the target architecture to be tested. Our proposed approach runs on the virtual platform we implement. This makes our approach faster and provides early results.

Keywords memory leak, soft real time system, virtual prototype, memory modelling

1. Introduction

With increasing design complexity, the early design tests of software become a crucial factor in electronic systems design. It helps to redesign or optimize the system at early design stages. This keeps flaws minimal in the production stage of embedded systems.

As embedded systems size gets smaller and smaller, memory becomes a crucial resource which needs to be managed efficiently and optimally. Inefficient memory management leads to severe memory problems which may result in system failures. Such flaws if detected early will lead to better designs with better performance.

Memory leak is the main memory related problem in soft real time systems and are considered as "hidden" problems that are hard to detect [23]. A memory leak is a memory location which is not freed after its use by the program, and hence unavailable for other components to re-use, as it is still reserved. A program causing leak allocate more and more memory over time leading the system to eventually run out of memory and fail. However, the code at the point of failure often has nothing to do with the leak and hence they are hard to detect. In soft real-time systems, where dynamic memory management is common, memory leaks have higher probability, especially for non-garbage collecting environments. Manual detection of leaks is tedious as they are hidden and hard to reproduce without any immediate symptoms. Hence an automated approach detecting leaks that is efficient and fast is needed.

Today, embedded systems software development is mainly conducted by virtual system prototypes, in combination with simulation-based approaches at different abstraction and refinement levels [20]. Memory leak, which is an important aspect to be tested at early design stages, could be effectively analyzed with virtual platforms. In this paper we introduce a novel approach to detect memory leaks with the help of virtual platforms. Existing approaches for memory leak detection are carried out at source code level or at actual run time [9], [12], whereas our approach is carried out at simulated abstract levels which makes early design tests feasible and are much faster than actual runs. We focus on memory leak detection in soft real-time systems with non-garbage collecting environments.

The remaining part of this paper is structured as follows. Section 2 describes related work in memory leak detection. Section 3 explains our approach for leak detection. Section 4 summarizes the paper and mentions the key challenges and future extensions of our work.

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop
December 3, 2013, Vancouver, Canada.

Copyright is held by the author/owner(s). The proceedings are published by Linköping University Electronic Press. Proceedings are available at:
http://dx.doi.org/10.3384/ecp13090

AVICPS website:
http://www.analyticintegration.org/

2. Related Work

In literature various methodologies are available to detect memory leaks, which could be classified into two main categories: static and dynamic.

2.1 Static Methods

Static methods assume the availability of source code or any other static form of the target software. These methods involve leak detection without actual execution. They have the advantage that they do not necessitate the availability of the execution environment and are much faster.

Approaches using Shape Graph [7], pointer analysis [25], escape analysis [30], shape analysis [11], contradiction analysis [24], liveness analysis [27], ownership model [14,15], procedural summaries [6], bi-abductive inferences [18], and value flow [4,28] are some of the prominent ones. LCLInt is a static leak detection tool which annotates the source code with formal specifications [9].

Although these approaches have above mentioned advantages, there are certain disadvantages too. These methods detect leaks to a certain extent, and are not capable of detecting all kinds of leaks, especially the ones that arise during actual run time such as leaks due to dynamic references [17]. Static methods are useful only to a certain extent and do not provide enough accuracy or guarantee to prevent crashes arising from memory anomalies during execution.

2.2 Dynamic Methods

Most of the memory anomalies appear in dynamic scenarios. Dynamic methods involve the actual execution of the tasks, and hence are much more accurate than the static methods in detecting leaks. Reference counting, reachability analysis and liveness are some of the prominent methods used [1,2].

Purify [12], Cork [17], Leakbot [19], Sleigh [1] are the most prominent state-of-the-art dynamic approaches. Other approaches include Hound [22] and SWAT [13]. [31] introduces object ownership profiling at runtime to detect leaks. Leakpoint [5] is yet another dynamic approach to pinpoint the leak and its location. Leakpruning [2], Plug [21], and Leaksurvivor [29] are other approaches to minimize the damage caused by leaks at runtime. [26] is a leak detection approach for android systems via PCB hooking. Apart from these there are several leak detection tools available commercially and non-commercially such as mtrace and valgrind [8].

A major limitation of these approaches and tools are that they require the program tested to be actually executed, which mandates the execution environment and its dependencies. Leak detection at execution also implies more overhead and this affects the overall performance.

The approach we propose is a hybrid approach which combines the advantages of both static and dynamic methodologies.

3. Virtual Memory Modeling

3.1 Simulation Framework

Early design tests are important in scenarios where software is developed in modules independently and integrated later. Hence a final test is possible only after all the modules are ready. Moreover, the real target environment may not be available at hand or it may be too expensive to afford just for testing. Hence there should be some sort of early design environments available to test these modules independent of the availability of the whole project. The simulation environment gains importance here. Simulations have the advantage that they are fast, re-usable and could be customized for each individual case and at the same time not expensive as the real hardware or the platform. It provides almost the same accuracy in terms of performance and other tests and is flexible and portable.

In order to efficiently explore the design space, designers need models of the embedded software running in its execution environment, providing rapid and early feedback about effects of design decisions, such as the chosen scheduling strategy. Models at higher levels of abstraction with fast simulation speeds with enough accuracy are needed [20]. Moreover each design targets a different architecture, and hence simulation environment provides flexibility to adapt different architecture without great changes to the execution environment.

The ARTOS (Abstract Real-Time Operating System) is a simulation framework developed by C-LAB , in order to simulate and analyze the schedulabilty of real time tasks [32]. This framework currently estimates the performance in terms of time consumed for the tasks and thereby determines the failures or design alternatives, which could improve the time. In this framework there is already a platform with various APIs to abstract the software and to run it for determining the run time of the tasks [32]. Currently ARTOS does not consider the memory transactions of the simulated software. Therefore we propose to integrate our memory model into ARTOS, and thereby provide a common platform to carry out timing analysis and detect memory leaks simultaneously.

3.2 Memory Management Model

We follow an abstract implementation of memory management, where transactions involving memory requests and allocations are of major focus. The memory model is basically a transaction level model, where the emphasis is given to the transactions occurring between memory requesting object and the memory object allocated. We implement the model using TLM library in SystemC [3,10]. Transaction-level modeling (TLM) is a high-level approach for modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture. Communication mechanisms are modeled as channels and are presented to modules using SystemC interface classes. Transaction requests take place by calling interface functions of these channel models which encapsulate low-level details of the information exchange [3]. This makes it easier for the system-level designer to conduct experiments on the required abstraction level, without the need to consider other dependencies required for the actual execution and behavioral correctness. Our model in TLM will initially include the basic request-allocate process and later a paging/segmentation model. The available memory in our model is considered as fixed-sized blocks for the respective target architecture.

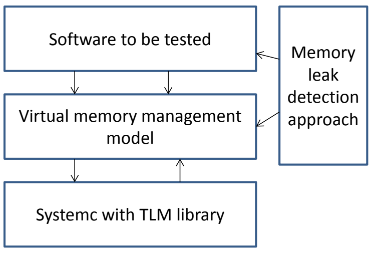

Software to be tested
Memory leak detection approach
Virtual memory management model
Systemc with TLM library

Figure 1: Virtual memory model for leak detection

A memory arbiter would be responsible for servicing the requests arriving from the tested software. The requests generated by the design under test is encapsulated and abstracted and passed to the memory arbiter. The arbiter checks for the available memory in the free memory area. The best-fit algorithm is used to allocate the most suitable sized memory block [23]. The acknowledgement and the allocated block with size and address are passed back to the requested entity. If there is no available free memory, the request is not served, and a request denial is sent back instead of acknowledgement by the arbiter. All these transactions are finally SystemC calls. An overview of our approach is shown in Figure 1.

3.3 Memory Leak Detection

Our approach of leak detection follows a hybrid model. The software under test undergoes static analysis before it is executed on the virtual model. The static analysis involves an automated control flow graph generation technique adopted from our previous work [16] as shown in Figure 2. The source code of the software to be tested is first disassembled. A graph based approach with automated labeling of basic blocks of the program is developed and the graph is then compacted for efficient detection of leaks.

Memory *request-allocate* paths are plotted on the graph with the help of disassembled code analysis. Each of such paths is checked for the corresponding *free* method for any *allocate* method. A one-to-one mapping is generated from the graph for *allocate* and *free* methods. If such a mapping is missing for any allocate function, a leak is notified and the corresponding location and object is marked.

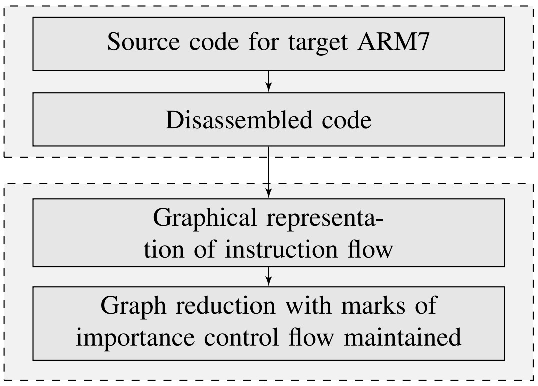

Source code for target ARM7
Disassembled code
Graphical representation of instruction flow
Graph reduction with marks of importance control flow maintained

Figure 2: Control flow graph generation technique

Next step is the dynamic analysis, where the software to test is run at an abstract level on the virtual memory model. The memory request calls (e.g. malloc), are encapsulated, and such function calls are redirected to the respective calls to the arbiter in the memory model. The arbiter according to the availability of the memory responds with an acknowledgement or request denial message back to the requested object. A memory control block is generated for each block of memory in the model. The memory control block is responsible to keep track of used and free memory blocks. After each allocation process in the model, the status of simulated memory in the virtual model is tested with the help of all memory control blocks. In *liveness* analysis method [2] the object requesting memory is checked if it still exists, otherwise the memory allocated is reclaimed. Similarly the methods of *reference counting* [1] is applied, where in the pointer references are tracked down to check if any unused memory is producing a leak. The *reachability* [2] approach is applied to check if any of the still live memory requesting and allocated objects are reachable via any pointer references at the end of the execution; and if unreachable it is considered a leak. At the end of each execution, the memory allocation pattern in the model is verified and the probability of various leaks reported through the above three methods are analyzed with various number of executions. The leak and the probability of leak occurrence are generated for further optimizations in the software.

Currently we consider C and C++ programs as our test cases along with gcc compiler. The target architecture we consider is ARM9, with virtual ARM integrator CP development board.

4. Conclusion and Future work

Memory leaks are serious problems in resource constrained embedded environments and need to be fixed at early stages of design, especially in non-garbage collecting environments. A fast and efficient leak detection system at early stages of development is required. We proposed a novel hybrid memory leak detection method in a simulation environment that can support different target architectures. The memory model is implemented at an abstract level using Transaction Level Modeling. The software under test is run at an abstract level on top of this model. Our proposed approach is aimed to be faster, since the analysis is done on top of a virtual platform with simulation. The main challenges include the optimization of simulation time, and we aim to make the simulation as fast as possible, along with precision. Our approach is intended for non-garbage collecting environments as the probability of leaks are higher in such environments.

The future work on this methodology includes the extension of the memory model to detect other memory problems such as fragmentation and corruptions. Moreover we need to have support for other languages and compilers, to make our model more generic. Another extension in this direction would be to incorporate this model to the ARTOS framework, so that users can have both memory and process related checks under one common framework.

References

[1] Michael D. Bond and Kathryn S. McKinley. Bell: bit-encoding online memory leak detection. *SIGPLAN Not.*, 41(11):61–72, October 2006.

[2] Michael D. Bond and Kathryn S. McKinley. Leak pruning. *SIGARCH Computer Architecture News*, 37(1):277–288, March 2009.

[3] L. Cai and D. Gajski. Transaction level modeling: an overview. In *Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on*, pages 19–24, 2003.

[4] Sigmund Cherem, Lonnie Princehouse, and Radu Rugina. Practical memory leak detection using guarded value-flow analysis. *SIGPLAN Not.*, 42(6):480–491, June 2007.

[5] James Clause and Alessandro Orso. Leakpoint: pinpointing the causes of memory leaks. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 515–524, New York, NY, USA, 2010. ACM.

[6] Dino Distefano and Ivana FilipoviĂĞ. Memory leaks detection in java by bi-abductive inference. In DavidS. Rosenblum and Gabriele Taentzer, editors, *Fundamental Approaches to Software Engineering*, volume 6013 of *Lecture Notes in Computer Science*, pages 278–292. Springer Berlin Heidelberg, 2010.

[7] Nurit Dor, Michael Rodeh, and Mooly Sagiv. Checking cleanness in linked lists. In Jens Palsberg, editor, *Static Analysis*, volume 1824 of *Lecture Notes in Computer Science*, pages 115–134. Springer Berlin Heidelberg, 2000.

[8] Cal Erickson. Memory leak detection in embedded systems. *Linux Journal*, August 2002.

[9] David Evans, John Guttag, James Horning, and Yang Meng Tan. Lclint: a tool for using specifications to check code. *SIGSOFT Softw. Eng. Notes*, 19(5):87–96, December 1994.

[10] Thorsten Grotker. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[11] Brian Hackett and Radu Rugina. Region-based shape analysis with tracked locations. *SIGPLAN Not.*, 40(1):310–323, January 2005.

[12] Reed Hastings and Bob Joyce. Purify: Fast detection of memory leaks and access errors. In *In Proc. of the Winter 1992 USENIX Conference*, pages 125–138, 1991.

[13] Matthias Hauswirth and Trishul M. Chilimbi. Low-overhead memory leak detection using adaptive statistical profiling. *SIGPLAN Not.*, 39(11):156–164, October 2004.

[14] David L. Heine and Monica S. Lam. A practical flow-sensitive and context-sensitive c and c++ memory leak detector. *SIGPLAN Not.*, 38(5):168–181, May 2003.

[15] David L. Heine and Monica S. Lam. Static detection of leaks in polymorphic containers. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 252–261, New York, NY, USA, 2006. ACM.

[16] M.M. Joy, M. Becker, W. Mueller, and E. Mathews. Automated source code annotation for timing analysis of embedded software. In *Advanced Computing and Communications (ADCOM), 2012 18th Annual International Conference on*, pages 12–18, 2012.

[17] Maria Jump and Kathryn S. McKinley. Cork: dynamic memory leak detection for garbage-collected languages. *SIGPLAN Not.*, 42(1):31–38, January 2007.

[18] Yungbum Jung and Kwangkeun Yi. Practical memory leak detector based on parameterized procedural summaries. In *Proceedings of the 7th international symposium on Memory management*, ISMM '08, pages 131–140, New York, NY, USA, 2008. ACM.

[19] Nick Mitchell and Gary Sevitsky. Leakbot: An automated and lightweight tool for diagnosing memory leaks in large java applications. In Luca Cardelli, editor, *ECOOP 2003 âĂŞ Object-Oriented Programming*, volume 2743 of *Lecture Notes in Computer Science*, pages 351–377. Springer Berlin Heidelberg, 2003.

[20] W. Mueller, M. Becker, A. Elfeky, and A. DiPasquale. Virtual prototyping of cyber-physical systems. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 219–226, 2012.

[21] Gene Novark, Emery D. Berger, and Benjamin G. Zorn. Plug: Automatically tolerating memory leaks in c and c++ applications. Technical report, Technical Report UM-CS-2008-009, University of Massachusetts, 2008.

[22] Gene Novark, Emery D. Berger, and Benjamin G. Zorn. Efficiently and precisely locating memory leaks and bloat. *SIGPLAN Not.*, 44(6):397–407, June 2009.

[23] Gary J. Nutt. *Operating systems - a modern perspective (3. ed.)*. Addison-Wesley-Longman, 2004.

[24] Maksim Orlovich and Radu Rugina. Memory leak analysis by contradiction. In *Proceedings of the 13th international conference on Static Analysis*, SAS'06, pages 405–424, Berlin, Heidelberg, 2006. Springer-Verlag.

[25] Berhard Scholz, Johann Blieberger, and Thomas Fahringer. Symbolic pointer analysis for detecting memory leaks. *SIGPLAN Not.*, 34(11):104–113, November 1999.

[26] Jooyoung Seo, Byoungju Choi, and Suengwan Yang. A profiling method by pcb hooking and its application for memory fault detection in embedded system operational test. *Inf. Softw. Technol.*, 53(1):106–119, January 2011.

[27] Ran Shaham, Elliot K. Kolodner, and Shmuel Sagiv. Automatic removal of array memory leaks in java. In *Proceedings of the 9th International Conference on Compiler Construction*, CC '00, pages 50–66, London, UK, UK, 2000. Springer-Verlag.

[28] Yulei Sui, Ding Ye, and Jingling Xue. Static memory leak detection using full-sparse value-flow analysis. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 254–264, New York, NY, USA, 2012. ACM.

[29] Yan Tang, Yan Tang, Qi Gao, Qi Gao, Feng Qin, and Feng Qin. Leaksurvivor: towards safely tolerating memory leaks for garbage-collected languages. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 307–320, Berkeley, CA, USA, 2008. USENIX Association.

[30] Yichen Xie and Alex Aiken. Context- and path-sensitive memory leak detection. In *In Proceedings of ESEC/FSE 2005*, pages 115–125. ACM Press, 2005.

[31] Guoqing Xu, Michael D. Bond, Feng Qin, and Atanas Rountev. Leakchaser: helping programmers narrow down causes of memory leaks. *SIGPLAN Not.*, 46(6):270–282, June 2011.

[32] Henning Zabel, Wolfgang Müller, and A. Gerstlauer. Accurate rtos modelling and analysis with systemc, January 2009.

Refinement of AADL models using early-stage analysis methods

Guillaume Brau[1] Jérôme Hugues[2] Nicolas Navet[1]

[1]University of Luxembourg, Laboratory of Advanced Software Systems,
6 rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
{guillaume.brau, nicolas.navet}@uni.lu

[2]Université de Toulouse – ISAE, 10 avenue E. Belin, 31055 Toulouse, France
jerome.hugues@isae.fr

Abstract

Model-Driven Engineering (MDE) is a relevant approach to support the engineering of distributed embedded systems with performance and dependability constraints. MDE involves models definitions and transformations to cover most of the system life-cycle: design, implementation and Verification & Validation activities towards system qualification. Still, few works evaluate the early integration of performance evaluation based on architectural models. In this paper, we investigate the early-stage use of analysis in AADL modeling. Precisely, we exemplify on an avionics case study how to dimension the data flows for an application distributed over an AFDX network. Based on the insight from this study, we suggest a simple framework and associated techniques to efficiently support analysis activities in the early-stage design phases.

Keywords AADL, ARINC653, AFDX, analysis combination, WCTT evaluation, Network Calculus.

For more details and experimental results, an extended version of this paper is available as a technical report [1].

1. Introduction

Context of the paper. Distributed Real-time Embedded (DRE) systems are present in safety-critical domains such as transportation, telecommunications, health services, military or space. These systems have to meet both the *functional* and *non-functional* requirements. Hence, DRE systems encompass specific technologies to realize the required service with the expected performance metrics (*e.g.* time, security or safety) through dedicated networks, processors or real-time operating systems. In addition, the engineering process has to address efficiently system modeling and evaluation of all metrics.

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop
December 3, 2013, Vancouver, Canada.

Definition of the problem. Many experiments indicate that the distance between the activities steps in a classical V-cycle is detrimental and usually slows down the development process [2]. In practice, a significant part of errors is injected at early-stage of the engineering process, while being detected during later integration phases. As a consequence, regressions and rework activities have an important weight on the overall project costs.

We believe that this problem can be lifted considering models with sufficient power of expression to guide the conception phases (e.g. using interface or behavioral models on which one can both reason and iterate) and supporting early-stage analysis. This “integrate, then build” approach, also known as *virtual integration* is promising to support the design of complex systems.

Contributions and objectives. In this paper, we use the Architecture Analysis & Design Language (AADL) [3] as the pivot to capture the system architecture and derive its performances. AADL is an Architecture Description Language (ADL) suitable to describe systems, capturing the *functional* and *non-functional* concerns together with the *operational platform*. As the AADL provides modeling elements with a precise *syntax* and well-defined *semantics*, it is possible to : 1) perform analysis and 2) derive implementations thanks to code generation [4].

The focus of this paper is twofold. First, taking a specific example coming from the avionics, we show how to accurately seize important parameters in the AADL model. As the system that is modeled includes technologies not supported within the core language, we build on and extend the work in [5] in order to include in the AADL model the networking elements and capture the overall DRE system.

Secondly, based on lessons learned during the modeling experience, we propose to jointly use AADL models and analysis methods so as to gradually refine and verify the model. We investigate and examplify this strategy on the avionics case study. Taking as example the network communications and the expressed timing constraints, we show that using complementary analysis methods allows to deduce missing parameters while maintaining the consistency of the model (*i.e.* chosen parameters guarantee that non-functional constraints are met).

The paper is organized as follows: we first introduce the avionics case study and give elements to model it with AADL (Section 2). In Section 3, we underline relationships between modeling concerns, and, using analysis, we explain how to solve those dependencies for some important communication parameters. Finally, in Section 4, we draw conclusions from this case study and discuss future work.

2. Modeling avionics systems with AADL

In this section we describe how to jointly capture with AADL the functional description of an avionics system, its temporal constraints and the target execution platform.

2.1 The Flight Management System

The avionics system to model, coming from Lauer et al. [6], is part of an aircraft's Flight Management System (FMS). It interacts with the crew and provides static and dynamic information about the flight plan (*i.e.* the predefined path between departure and arrival points) : current location, remaining distance and estimated arrival time.

Functional description. The system is made up of five main functions as depicted in Figure 1. The *Keyboard and cursor control Unit (KU)* reads data inputted by the pilot (or copilot) through the keyboards while the *Multi Functional Display (MFD)* refreshes the displays consecutively. From the KU function, crew requests are forwarded to the *Flight Manager (FM)* function which computes the response about the flight plan and returns it to the MFD. For this, it requests static data to the *Navigation Data Base (NDB)* function and also relies on dynamic data from the *Air Data Inertial Reference Unit (ADIRU)*, computed based on sensors measurements.

Temporal constraints. In the avionics context, the system has to conserve a predictable behavior. Several temporal constraints may be expressed upon different "locations". Typically, temporal constraints concern : 1) response times which are the delays needed to carry out the functions, 2) traversal times refering to communication delays between functions and 3) latencies along functional chains that encompass a succession of response times and traversal times.

Execution and communication platform. The *functions* are executed in an *Integrated Modular Avionics (IMA)* environment. In particular, this execution and communication platform supports two standards, used in the case study, that defines the use of the shared hardware and software resources in a deterministic way.

The ***ARINC653*** [7] standard defines management of the functions hosted by a same hardware/software platform (referred as an execution *module* in the following). In this environment, each function is located in a different *partition* with a strict access to processing and memory resources.

The ***ARINC664*** [8] standard defines a deterministic communication network called *Avionics Full Duplex-Switched Ethernet* (AFDX). AFDX implements the core concept of *Virtual Link* (VL) which is an unidirectional logical connection from one sender to one or several receiver(s). Each VL has a limited bandwidth according to its *Bandwidth Allocation Gap* (BAG in the following) – the minimum time elapsed between two frames sending – and a *maximal allowed packet size* (s_{max} in the remainder).

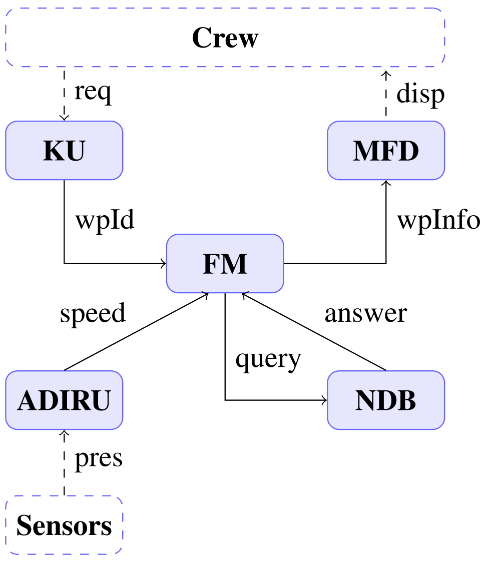


Figure 1. The Flight Management System functional architecture depicts the functions and the data exchanged as well as the interactions with the actors.

2.2 Modeling the FMS in AADL

The Architecture Analysis & Design Language (AADL) [9] is an international standard by the *Society of Automotive Engineers (SAE)*, defining the basics of an architecture description language dedicated to the design of DRE systems. AADL is component-centric and allows to specify both software and hardware parts of a system. See [3] for a complete presentation of AADL.

The full model of the FMS uses AADLv2 core specifications and the ARINC653 Annex. The model is made up of AADL *components* that describe the ARINC653 execution modules hosting the avionics functions and the AFDX network that supports the data exchanges. The full AADLv2 textual model – that spans over 770 SLOCs – is part of the AADLib project and is available at `http://www.openaadl.org`.

The model follows the initial specifications and AADL guidelines for ARINC653 systems : a module is a distinct `system` (containing a global `memory` and a `processor`) that hosts partitions (each is a `process`) bound to separate `memory` segments and `virtual processors` (representing spatial and temporal partitioning). `thread` components contained in partitions realize the avionics functions. Thanks to annex guidelines, we can model precisely the ARINC653 components and associated parameters.

AADL does not provide specific guidelines for modeling AFDX networks. The AADL concept of `virtual bus` defines a connection supported in a `bus`. We use this concept to define AFDX virtual links. Switches are represented by `device` components bound to the virtual links. A dedicated property set has been defined to model parameters attached to virtual links, end systems and switches.

3. Analysis as part of the design process

We discussed how AADL allows to capture both the FMS functional and non-functional aspects as well as the IMA platform description. From this model, we may now consider further analysis of the full architecture. The current design could be used to validate a given architecture. Yet, the most challenging part is actually guiding the designer in finding a suitable definition of the architecture parameters in order to respect the constraints expressed in the model.

3.1 Discussion : there are dependencies between modeling concerns

Figure 2 summarizes the three traits caught in the architecture model : the functions to realize, the hardware and software platform hosting the functions and the constraints to comply with. It implies that the architecture model components and the attached properties have to integrate and solve the dependencies between these views.

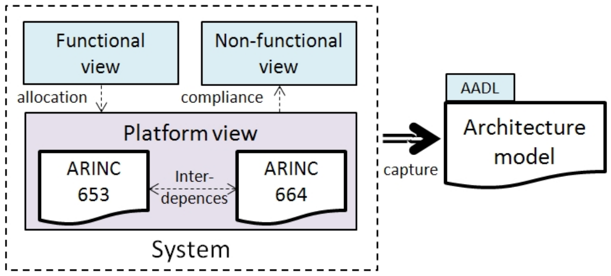


Figure 2. The architecture model captures jointly the system functional, non-functional and platform concerns and has to integrate the dependencies between these aspects.

For instance, let us consider the design of virtual links in AFDX networks. The virtual links characterization (***platform*** view) depends on :

- the data flows needed to realize the functions (***functional*** view) and their features, *e.g* the number (n) of messages sent by a function and their size (m),
- the constraints expressed onto the data flows (***non-functional*** view), *e.g* a communication between two specific tasks can be subject to timing constraints (noted L_C in the sequel).

Moreover, finding a suitable design for the virtual links in AFDX networks can be a difficult problem because of the interferences between VLs definitions. For further information, the issue is addressed in a more comprehensive setting in [10].

Proposed approach. We believe it is possible to deal with the dependencies between the modeling concerns by executing relevant analysis methods onto the model under construction. In the sequel, we show how to define progressively the VLs parameters taking into account information in the model such as the constraints expressed upon the communications. From an evaluation perspective, it implies : 1) isolating model input parameters that can be combined to 2) propose a feasible solution which is 3) later assessed. Some parameters are mandatory, while others can be assumed. This is discussed in the following paragraphs.

3.2 Example : integration of the Bandwidth Allocation Gap (BAG) parameter into an incomplete AADL model

Let us consider a partially completed AADL model. This initial model contains basic information : the functions hosted in the modules and their properties as well as the data exchanges between them. We also know the constraints of the system expressed onto the communications.

At this stage, dimensioning the BAG, which has a direct impact on the respect of timing constraints and the network load, may be a difficult task because the design space can be huge. Indeed, as this parameter ought to respect the formula BAG= 2^k [ms] with k integer in range 0 to 7 (see [8]), if we take as assumption that one VL is dedicated to one data flow, then there are $sol_{bag} = 8^f$ conceivable solutions, with f the number of flows.

To overcome this problem and complete the model, we execute the process pictured on Figure 3. We propose to:

1. use a *pivotal* Worst-Case Traversal Time evaluation (WCTT in the following) in order to identify the set of suitable BAGs,
2. use a *complementary* analysis method, relying on Network Calculus (NC in the following), to improve the results of the main WCTT evaluation.

WCTT evaluation. The Worst-Case Traversal Time analysis aims to assess the delay experienced by each data flow in the AFDX network. Without too much details, the WCTT evaluation gives the formula of an upper-bounded delay (D_T^{wctt}) suffered by a frame as a function of several parameters. Hence, it is possible to compare the delay against the expressed constraint ($D_T^{wctt} \leq L_c$) and to calculate the suitable set of BAGs [1]. The WCTT evaluation gives a set of suitable BAGs for each VL ($BAG_{vl_i}^{wctt}$). Of course, the accuracy of the BAGs sets depends on the precision of the model and of the assumptions. In addition, at the first stage, there is no NC feedback, i.e. $D_{sw\_vl_i} = 0$.

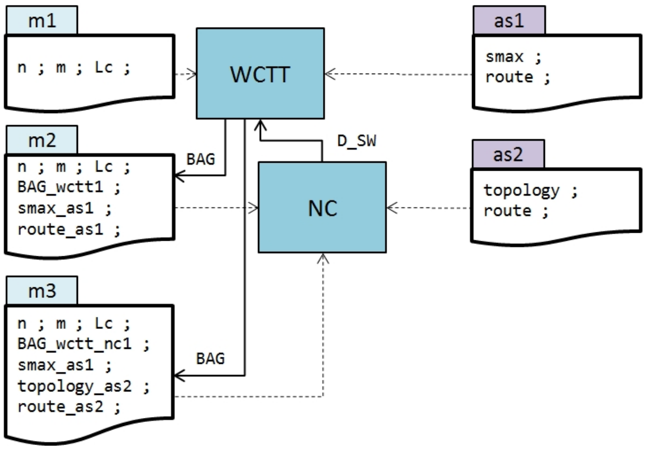


Figure 3. The BAG refinement process includes AADL models (blue-headed shapes), analysis methods (portrayed by green rectangles) and assumptions models (purple-headed shapes).

NC analysis. Network Calculus (NC) is an algebra for computing accurately the end-to-end delay of a data flow in the AFDX network. In the scope of this paper, this analysis is performed using RTaW-Pegase, which is a commercial product implementing a state-of-the-art network calculus AFDX timing analysis [11]. In our case, the NC analysis computes complementary information ($D^{nc}_{sw\_vl_i}$) and allows to refine results of the main WCTT evaluation.

Model refinement. We can see through the modeling and analysis flow (figure 3) that, as long as the required analysis inputs are present, the model is enhanced. The model refinements are done in line with : 1) the model evolution ($m1$, $m2$, $m3$), 2) the analysis methods outputs (BAG^{wctt} and D^{nc}_{sw}) and 3) the feasible assumptions ($as1, as2$).

The first execution of the pivotal analysis method takes as input the data of the incomplete model ($m1$) and the deduced assumptions ($as1$). The first coarse-grained WCTT evaluation reduces the space of BAGs solutions for the VLs ($BAG^{wctt_1}_{vl_i}$) attached to the data flows in order to verify the latency constraints ($L^{m_1}_{C\_vl_i}$).

Thanks to the first WCTT computation, the initial model (m_1) is enriched with a crucial missing parameter : the BAG (m_2). We are then able to perform the complementary NC analysis that aims to evaluate the upper delay suffered in the switches for each VL ($D^{nc_1}_{sw\_vl_i}$). To execute the complementary analysis, the assumption model ($as2$) contains additional information about the network topology and the VLs routes. The NC computed latency is passed as a refinement parameter to the WCTT method so as to precise the BAGs sets. Taking into account the calculated $D^{nc_1}_{sw\_vl_i}$ reduces the set of eligible BAGs ($BAG^{wctt\_nc_1}_{vl_i}$) : solutions that do not meet the initial $L^{m_1}_{C\_vl_i}$ constraints are discarded.

At the third iteration, the model ($m3$) contains the refined BAGs ($BAG^{wctt\_nc_1}_{vl_i}$). It is then necessary to calculate the delay suffered in the switches for each VL ($D^{nc_2}_{sw\_vl_i}$) and refine the BAGs sets ($BAG^{wctt\_nc_2}_{vl_i}$). In our example, a new combined execution of WCTT and NC shows that a fixed-point is reached : the model m_3 cannot be refined anymore against the Bandwidth Allocation Gap if the input parameters and assumptions stay stable.

4. Conclusions and perspectives

Our first contribution (Section 2) dealt with the architectural description of an avionics system with AADL, combining the *functional*, *non-functional* and *platform* concerns. We extended existing patterns dedicated to ARINC653 systems to also model AFDX networks.

We then addressed (Section 3) the definition and evaluation of the AADL model components and their properties. We showed that dealing with the architecture description amounts to solve the dependencies between the AADL model concerns. Starting from an incomplete AADL model, we implemented a process combining two analysis methods to evaluate and refine the Bandwidth Allocation Gap parameter. This process combines early and in-depth analysis to help the designer to narrow the design space.

We believe it is necessary to formalize the use of analysis methods along with modeling languages in order to tackle the early system architecture definition and evaluation. In a previous work, we defined REAL [12] – Requirement Enforcement and Analysis Language. It allows one to define a set of predicates, and check whether a system satisfies them. We plan to extend REAL to define a library of predicates for *tools*. Such predicates would 1) define conditions under which a given analysis becomes feasible and 2) detect and exploit relationships between analysis methods. This would guide the designer to trigger relevant analysis as early as possible in the design flow.

The definition of those predicates, in the context of the FMS, is currently under implementation.

References

[1] G. Brau, J. Hugues, and N. Navet. Refinement of AADL models using early-stage analysis methods. Technical Report TR-LASSY-13-06, LASSY, University of Luxembourg, 2013. Available at `http://orbilu.uni.lu/`.

[2] D. Redman, D. Ward, J. Chilenski, and G. Pollari. Virtual Integration for Improved System Design. In *Proceedings of The First Analytic Virtual Integration of Cyber-Physical Systems Workshop*, San Diego, California, USA, Nov. 2010.

[3] P. H. Feiler and D. P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 1st edition, 2012.

[4] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues. Ocarina : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In *Proceedings of the 14th Ada-Europe International Conference*, Brest, France, June 8-12 2009.

[5] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, and F. Kordon. Validate, simulate, and implement ARINC653 systems using the AADL. In *SIGAda annual international conference on Ada and related technologies*, SIGAda '09, pages 31–44, New York, NY, USA, 2009.

[6] M. Lauer. *Une méthode globale pour la vérification d'exigences temps réel - Application à l'Avionique Modulaire Intégrée*. Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, juin 2012.

[7] Aeronautical Radio Incorporated. *ARINC Report 653P0 Avionics Application Software Standard Interface, Part 0, Overview of ARINC 653*.

[8] Aeronautical Radio Incorporated. *ARINC Report 664P7-1 Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network.*

[9] SAE/AS2-C. Architecture Analysis & Design Language V2 (AS5506A), January 2009.

[10] A. Al Sheikh, O. Brun, M. Chéramy, and P.-E. Hladik. Optimal design of virtual links in AFDX networks. *Real-Time Systems*, 49(3):308–336, 2013.

[11] M. Boyer, J. Migge, and M. Fumey. PEGASE - A Robust and Efficient Tool for Worst-Case Network Traversal Time Evaluation on AFDX. In *SAE AeroTech Congress & Exhibition*, Toulouse, France, October 18-21 2011.

[12] O. Gilles and J. Hugues. Expressing and enforcing user-defined constraints of AADL models. In *Proceedings of the 5th UML& AADL Workshop*, Oxford, UK, 2010.

www.analyticintegration.org

Series: Linköping Electronic Conference Proceedings, No. 90
ISSN (print): 1650-3686
ISSN (online): 1650-3740
ISBN: 978-91-7519-451-6
DOI: http://dx.doi.org/10.3384/ecp13090