Workshop - Systems Design Meets Equation-based Languages

Rantzer, Anders; Åkesson, Johan

2013

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Rantzer, A., & Åkesson, J. (2013). *Workshop - Systems Design Meets Equation-based Languages*. (Technical Reports TFRT-7638). Department of Automatic Control, Lund Institute of Technology, Lund University.

Total number of authors:
2

LCCC Workshop: Systems Design meets Equation-based Languages

19-21 September 2012

Old Bishop's Palace at Biskopsgatan 1 in Lund

Scientific Committee

Johan Åkesson, Lund University, Sweden (Chair)
Moritz Diehl, KU Leuven, Belgium
Hilding Elmqvist, Dassault Systèmes, Sweden
Claus Führer, Lund University, Sweden
Clas Jacobson, United Technologies Research Center, USA
Eric Van Wyk, University of Minnesota, USA
Anders Rantzer, Lund University, Sweden, LCCC coordinator

Organizing Committee

Claus Führer
Görel Hedin
Anders Rantzer
Eva Westin
Johan Åkesson

MAILING ADDRESS
Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND, SWEDEN

VISITING ADDRESS
Institutionen för Reglerteknik
Ole Römers väg 1
232 63 LUND

TELEPHONE
+46 46 222 87 87

FAX
+46 46 13 81 18

GENERIC E-MAIL ADDRESS
control@control.lth.se

WWW
www.lccc.lth.se

Printed: Media-Tryck, Lund, Sweden, August 2013

ISSN 0280-5316
ISRN LUTFD2/TFRT--7638--SE

Content

1. Introduction

LCCC workshops are organized in a 3-day format. About 20-25 speakers from academia and industry are invited for the workshop, selected for excellence and for an optimal coverage of the theme. The speakers are also encouraged to extend their stay beyond the workshop for further interaction with the local research environment. For each workshop, the research theme is chosen strategically to support the vision of a LCCC, usually with a cross-disciplinary perspective. An international scientific committee is responsible for the program.

1.1 WORKSHOP THEME

Equation-based object-oriented languages (EOOL), such as Modelica and VHDL-AMS, have become widely used in academia and industry during recent years. While these languages are mainly oriented towards dynamic simulation, they are well suited as a basis for solving a wider range of engineering design problems, making use of existing and new algorithms. Examples include sensitivity analysis, state and parameter estimation, optimal control and MPC, robust design, and model reduction.

1.2 SCOPE

The workshop focused on how EOOLs can be extended to support this wider range of problems in systems design. The following aspects are of primarily interest:

1. *Extension examples:* What kind of engineering design problems could benefit from support through EOOLs, or extensions to an EOOL language? What existing or new algorithms could be used for such extensions? An existing example for such an extension is Optimica which adds optimization capabilities to Modelica.
2. *Language extension design:* How can such extensions be formulated as language extensions? What different techniques, e.g., annotations, syntactic extensions, semantic extensions, or embedded DSLs are appropriate for different extensions? How can model execution standards, e.g., the Functional Mock-up Interface (FMI) be explored to link language extensions to algorithms?
3. *Language extension implementation:* How can these extensions be implemented in supporting tools like compilers? How can modularity with respect to core languages be maintained? How can interactive tools like IDEs be extended to support the language extensions? Examples of new metacompilation frameworks supporting language extensions include JastAdd, Silver, and Kiama.
4. *Applications:* What interesting industrial cases can be found that could benefit from such new developments?

Supporting such extensions to EOOL's would answer the strong industrial need for integrating existing EOOL models with systems design algorithms and on-line control systems.

The problems are cross disciplinary, and the aim of the workshop was therefore to bring together researchers and industrial practitioners in several fields, including engineering design (modeling, simulation, optimization, etc.), computer science (languages and tools), numerical analysis (algorithms for solving design problems), and applications.

The workshop supported the LCCC theme Modeling for design and verification. During the last few years, a local community has emerged, consisting of researchers at the departments of Mathematics, Computer Science and Automatic Control, and companies, notably Modelon, Lund, and ABB, Malmö. The local community is oriented towards the two open source projects

JModelica.org (an open-source implementation of Modelica) and JastAdd (a meta-compilation tool supporting language extension). The theme of the workshop stemmed from this environment cross-disciplinary interactions between researchers at Lund university, local companies, and students are frequent. Such interactions include joint master's thesis projects, joint scientific publications, joint PhD student advising, all inspired by industrial applications.

1.3 ORGANIZATION AND VENUE

The workshop was initiated by Claus Führer (Center for Mathematical Sciences), Görel Hedin (Department of Computer Science) and Johan Åkesson (Department of Automatic Control).

The scientific committee consisted of Johan Åkesson (chair), Moritz Diehl, Hilding Elmqvist, Claus Führer, Clas Jacobson and Eric van Wyk.

The local organization and interactions with workshop speakers and participants was handled by Eva Westin.

The workshop was held at the Pufendorf Institute at Lund University 19-21 September 2012.

2. Panel discussion

Participants: **Albert Benveniste**, **Hilding Elmqvist, Carl D. Laird**, **Edward E. Lee, Clas Jacobson**

Moderator: **Karl Johan Åström**

The panel discussion circled around three main themes; modeling and systems engineering in eduction, employing equation-based languages in systems design, and formalization of model representations.

2.1 MODELING AND SYSTEMS ENGINEERING IN EDUCATION

C. Jacobson put forward the observation that systems engineering is no longer taught by academic institutions. As a consequence, graduated engineers lack experience with systems design tools, which are widely used in industry. In cases where systems design courses are offered by universities, they are often taught by industrial practitioners that are brought in for the occasion.

E. Lee suggested to introduce a new topic into program curricula: Model Engineering. While this topic would build on established disciplines, it would emphasize that the concept of modeling as a key element in systems engineering. What is currently offered by universities in this area is generally weak. E. Lee made an analogy to software engineering, which has a long-time tradition within academia, and which contains a number of structured concepts that are taught systematically. Concepts suggested to be integrated into the topic model engineering include object-orientation, represented by languages such as Modelica, and refactoring of model code, which is a standard technique in software engineering.

A. Benveniste noted that mathematics is and must remain a fundamental element of systems engineering – mathematics is everywhere! It was also noted that French software industry emphasizes systems engineering for this particular reason.

2.2 EMPLOYING EQUATION-BASED LANGUAGES IN SYSTEMS DESIGN

In his opening note, H. Elmqvist talked about recent directions in the development of the Modelica language. The latest version of Modelica supports synchronous constructs. State-machines have been added in order to promote modeling of clock and sequential control systems. H. Elmqvist stressed the need to continue to expand the scope of Modelica to cover areas such as requirements management, integration with 3D modeling tools, Monte Carlo analysis, embedded optimization in physical models and systems design in general. H. Elmqvist also took the opportunity to invite everybody to interact and to contribute to the further development of Modelica.

C. Jacobson commented that equation-based languages are currently not used to their full potential. Given the languages and tools available today, we can move from experimentation based on simulation to computations in systems design. C. Jacobson mentioned Six Sigma and Monte Carlo techniques as targets for integration with computational frameworks based on physical models, and he highlighted rich opportunities for research in the area, for example in propagation of uncertainty.

C. Laird talked about the interplay between algorithm design and modeling, specifically in the context of dynamic optimization of large-scale non-linear systems. In effect, the way models are constructed is affected by the capabilities of such algorithms. In addition, the need for exploitation of structure in models was stressed.

2.3 FORMALIZATION OF MODELS

A. Benveniste used the fighter aircraft Rafale to exemplify the need for integrated and formal methods in requirements managment and verification. Approximately 250.000 requirements were considered in the design. The process was characterized by informal handling of the requirements, multiple engaged sub-contractors, and often, requirements verification without models. In other activities in the project, however, models were developed and used extensively, including system dimensioning, control design and Product Lifecycle Management (PLM). Typically, very different modeling tools were used for these purposes. Based on the example, A. Benveniste put forward questions to be adressed in research and in industrial practice. How to fuse the model-based tools in order for models to become widely available in different processes? How does the V-model for product development come into play in this context? What is needed in terms of Modelica extensions in order to accomodate the needs exemplified in the Rafale project?

In his remarks, E. Lee reasoned about what properties of models we should value. Three aspects were brought forward. Firstly, fidelity of models is a key property, that is to what degree the models mimic a given system. Secondly, understandability of a model, something we are often eager to sacrifice, should be valued. E. Lee called for a cultural change in this respect – we should be proud of small models! Thirdly, analyzability of a model is important in order to perform model-based analyses such as model checking and verification. E. Lee stressed in this context the need for formal model description formats.

3. Summary and outlook

3.1 IMPORTANT OBSERVATIONS

- Different approaches to modeling of hybrid systems were discussed during the workshop. This seems to be one of the core challenges in the area, i.e., to develop a rigorous mathematical formalism to describe the semantics of models encoded in languages such as Modelica, Ptolemy and VHDL-AMS, and in model exchange standards such as FMI.
- The interest in model exchange formats which are neutral with respect to physical domain, modeling language, and software tool is increasing. The Functional Mock-up Interface is rapidly being adopted in research and in industry, which was evident from several presentations. In addition, the CIF format which resulted from the MULTIFORMS project was presented.
- The interest in Modelica is broadening, and the scope of the language is expanding from primarily modeling of physical systems to control systems and systems design. Specifically, synchronous extensions to Modelica and optimization based on Modelica models were discussed. Also, the potential of Modelica in systems design was high-lighted during the panel discussion.
- The need for formal verification of requirements, and approaches to solving such problems was a strong theme during the workshop. This topic was high-lighted during the panel discussion in the context of aircraft control systems and in several presentations.
- Some speakers bore witness to difficulties in applying software for non-convex dynamic optimization to industrial problems. The level of maturity of existing algorithms for such problems seems to be significantly less than for simulation tools targeting the same class of systems.
- Extensible languages and compilers is becoming feasible through research efforts in the computer science community. Two different approaches to compiler extensibility was discussed in the workshop presentations.
- Python holds a strong position in the scientific computing field, which was underlined in a number of presentations.

3.2 OPEN PROBLEMS

- **Modeling formalizms for hybrid systems**. Several speakers touched upon modeling formalisms for hybrid systems. While there are different frameworks available for description of hybrid systems, consensus is yet to be reached upon the semantic behaviour and a unified mathematical theory.
- **Robustness of numerical optimization algorithms for large-scale non-linear dnamic systems**. The academic community has produced a large body of algorithms for optimization of large-scale non-linear dynamic systems. Still, industrial practitioners experiences significant challenges in applying such algorithms to problems relevant for their applications.
- **Physical modeling languages for convex optimization**. Current modeling languages such as VHDL-AMS and Modelica target construction of non-linear and hybrid physical system models, which are not immediately useful as a basis for the large body of available optimization algorithms for convex optimization. Still, many physical systems can be modeled in order to fulfill the requirements of convex optimization. Accordingly, challenges remain in combining concepts from EOOL and convex optimization.

3.3 ACTIONS

From the discussions during the workshop, it is clear that there are rich opportunities for cross fertilization between different fields represented by speakers and paricipants. Based on these discussions, the following actions are recommended.

- **More efforts are needed in terms of language support for optimization**. Several presentations touched upon this topic and several interesting directions were mentioned, including convex optimization formulations based on physical modeling languages, challenges in application of state-of-the-art optimization algorithms to large-scale physical models, and industrial applications.
- **Increased interaction is needed between communities working with modeling formalisms for hybrid systems**. It is clear that there are several research groups developing modeling formalisms for hybrid systems, as well as industrial initiatives such as FMI and Modelica. Interactions between these groups would be beneficial in order to develop a unified framework for modeling of hybrid systems. An initiative in this direction was taken by the Modelica community, represented by H. Elmqvist, who visited E. Lee’s group in the weeks following the workshop.
- **Establishment of a repository of dynamic benchmark models of industrial grade to support research in systems design**. Development of relevant industrial grade models requires a high level of expertise, that this not always available in research projects targeting systems design. Such projects benefit from freely available dynamic models.

Appendix A

PROGRAM

Wednesday, September 19, 2012

08.30-09.00	Registration
09.00-09.10	Opening session
09:10-10:10	*Non-standard semantics of hybrid systems modelers*
	Albert Benveniste, IRISA/INRIA
	Equations, Synchrony, Time, and Modes
	Edward A. Lee, EECS, UC Berkeley
10:10-10:40	Coffee
10:40-12:10	*Formal Modeling and Analysis of Software Systems with Lustre*
	Mike Whalen, University of Minnesota
	Systems Engineering: Status of Industrial Use, Opportunities and Needs
	Clas Jacobson, United Technologies Systems & Controls Engineering
	The OpenModelica Environment including Static and Dynamic Debugging of Modelica Models and Systems Engineering / Design verification
	Peter Fritzson, Linköping University, PELAB
12:10-13:30	Lunch
13:30-15:00	*The Dark Side of Object-Oriented Modelling: Numerical Problems, Existing Solutions, Future Perspectives*
	Francesco Casella, Politecnico di Milano
	Bridging between different modeling formalisms – results from the MULTIFORM project
	Sebastian Engell, TU Dortmund
	Equation-based Modeling and Control of Industrial Processes
	Johan Sjöberg, ABB AB, Corporate Research and Linköping university
15:00-15:30	Coffee
15:30-16:30	*FMI: Functional Mockup Interface for Model Exchange and Co-Simulation*
	Torsten Blochwitz, ITI GmbH Dresden
	Vertical Integration in Tool Chains for Modeling, Simulation and Optimization of Large-Scale Systems
	Johan Åkesson, Modelon AB and Lund University

Thursday, September 20, 2012

09:00-10:00	*System Design – From Requirements to Implementation*
	Alberto Ferrari, ALES S.r.l.
	Synchronous Control and State Machines in Modelica
	Hilding Elmqvist, Dassault Systèmes AB
10:00-10:30	Coffee

10:30-12:00	*Extensible Programming and Modeling Languages* **Eric Van Wyk**, University of Minnesota *Extensible compiler architecture – examples from JModelica.org* **Görel Hedin**, Lund University *Constraint satisfaction methods in embedded system design* **Krzysztof Kuchcinski**, Lund University
12:00-13:30	Lunch
13:30-15:00	Discussion
15:00-15:30	Coffee
15:30-16:30	*Dynamical models for industrial controls: use cases and challenges* **Fernando D'Amato**, GE Global Research Center *Origins of Equation-Based Modeling Languages* **Karl Johan Åström**, Lund University
18:20	Gathering at Bangatan 14 (next to Ica Kvantum Malmborgs)
19:00	Workshop dinner at Häckeberga castle

Friday, 21 September, 2012

09:15-10:00	Panel discussion
10:00-10:30	Coffee
10:30-12:00	*Pyomo: Optimization Modeling in Python* **Carl Laird**, Texas A&M University *Efficient symbolical and numerical algorithms for nonlinear model predictive control with OpenModelica* **Bernhard Bachmann**, Fachhochschule Bielefeld University of Applied Sciences *Algorithmic differentiation: Sensitivity analysis and the computation of adjoints* **Andrea Walther**, Universität Paderborn
12:00-13:00	Lunch
13:00-14:30	*CasADi: A Tool for Automatic Differentiation and Simulation-Based Nonlinear Programming* **Moritz Diehl**, OPTEC KU Leuven *Modeling Seen as Programming* **Klaus Havelund**, Jet Propulsion Laboratory, California Institute of Technology *Verification of Stiff Hybrid Systems by Modeling the Approximations of Computational Semantics* **Pieter J. Mosterman**, MathWorks
14:30-15:00	Coffee
15:00-16:00	*Assimulo – a Python package for solving differential equation with interface to equation based languages* **Claus Führer**, Lund University *Functional Development with Modelica* **Stefan-Alexander Schneider**, Schneider System Consulting
16:00-16:05	Closing

Appendix B

PARTICIPANTS

Christian Andersson	Lund University	chria@maths.lth.se
Joel Andersson	KU Leuven	joel.andersson@esat.kuleuven.be
Bernhard Bachmann	Bielefeld University	bernhard.bachmann@fh-bielefeld.de
Albert Benveniste	IRISA/INRIA	albert.benveniste@inria.fr
Karl Berntorp	Lund University	karl.berntorp@control.lth.se
Enrico Bini	Lund University	enrico.bini@control.lth.se
Torsten Blochwitz	ITI Gmbh	Blochwitz@itisim.com
Anders Blomdell	Lund University	anders.blomdell@control.lth.se
Francesco Casella	Politecnico di Milano	casella@elet.polimi.it
Fernando D'Amato	General Electric Global Research	damato@ge.com
Moritz Diehl	KU Leuven	Moritz.Diehl@esat.kuleuven.be
Adam Duracz	Halmstad University	adam.duracz@hh.se
Jonas Eborn	Modelon	jonas.eborn@modelon.com
Johans Eker	Ericsson	johan.eker@ericsson.com
Hilding Elmqvist	Dassault Systèmes AB	Hilding.ELMQVIST@3ds.com
Sebastian Engell	University of Dortmund	sebastian.engell@bci.tu-dortmund.de
Alberto Ferrari	ALES	alberto.ferrari@ales.eu.com
Niklas Fors	Lund University	niklas.fors@cs.lth.se
Peter Fritzson	Linköping University	peter.fritzson@liu.se
Claus Führer	Lund University	claus@maths.lth.se
Mahdi Ghazaei	Lund University	mahdi.ghazaei@control.lth.se
Joris Gillis	KU Leuven	joris.gillis@mech.kuleuven.be
Christian Grussler	Lund University	christian.grussler@control.lth.se
Manuel Gräber	TU Braunschweig	m.graeber@tu-bs.de
Meng Guo	KTH	mengg@kth.se
Magnus Gäfvert	Modelon	magnus.gafvert@modelon.com
Gabriel Hackebeil	Texas A&M University	gabehack@gmail.com
Mathias Haage	Lund University	Mathias.Haage@cs.lth.se
Per Hagander	Lund University	per.hagander@control.lth.se
Ulf Hagberg	ABB	ulf.hagberg@se.abb.com
Klaus Havelund	JPL-NASA	klaus@havelund.com
Görel Hedin	Lund University	Gorel.Hedin@cs.lth.se
Clas Jacobson	United Technologies Res. Center	JacobsCA@utrc.utc.com
Jörn Janneck	Lund University	jwj@cs.lth.se
Krzysztof Kuchcinski	Lund University	Krzysztof.Kuchcinski@cs.lth.se
Carl Laird	Texas A&M University	carl.laird@tamu.edu
Edward Lee	University of California	eal@eecs.berkeley.edu
Fredrik Magnusson	Lund University	fredrik.magnusson@control.lth.se
Sven Erik Mattsson	Dassault Systèmes AB	SvenErik.Mattsson@3ds.com
Pieter Mosterman	McGill University/Mathworks	pieter.mosterman@mathworks.com

Anders Nilsson	Lund University	anders.nilsson@control.lth.se
Björn Olofsson	Lund University	bjorn.olofsson@control.lth.se
Hans Olsson	Dassault Systèmes AB	hans.olsson@3ds.com
Alessandro Papadopoulos	Politecnico di Milano	papadopoulos@elet.polimi.it
Anders Rantzer	Lund University	anders.rantzer@control.lth.se
Stefan-Alexander Schneider	BMW	Stefan-Alexander.schneider@bmw.de
Eelco Scholte	United Technologies Res. Center	eelco.scholte@utas.utc.com
Johan Sjöberg	ABB Corporate Research	johan.sjoberg@se.abb.com
Emma Söderberg	Lund University	emma.soderberg@cs.lth.se
Walid Taha	Halmstad University	Walid.Taha@hh.se
Hubertus Tummescheit	Modelon AB	Hubertus.Tummescheit@modelon.com
Andreas Varchmin	TU Braunschweig	a.varchmin@tu-bs.de
Eric van Wyk	University of Minnesota	evw@cs.umn.edu
Andrea Walther	Universität Paderborn	andrea.walther@uni-paderborn.de
Mike Whalen	University of Minnesota	whalen@cs.umn.edu
Daniel Word	Texas A&M University	dword@tamu.edu
Johan Åkesson	Lund University/Modelon	johan.akesson@control.lth.se
Karl-Erik Årzén	Lund University	karlerik@control.lth.se
Karl Johan Åström	Lund University	karl_johan.astrom@control.lth.se

Appendix C

PRESENTATIONS

NON-STANDARD SEMANTICS OF HYBRID SYSTEMS MODELERS

Albert Benveniste, IRISA/INRIA

Hybrid system modelers have become a corner stone of complex embedded system development. Embedded systems include not only control components and software, but also physical devices. In this area, Simulink is a de facto standard design framework, and Modelica a new player. However, such tools raise several issues related to the lack of reproducibility of simulations (sensitivity to simulation parameters and to the choice of a simulation engine). In this paper we propose using techniques from non-standard analysis to define a semantic domain for hybrid systems. Non-standard analysis is an extension of classical analysis in which infinitesimal (the ε and η in the celebrated generic sentence $\forall\varepsilon\exists\eta\ldots$ of college maths) can be manipulated as first class citizens. This approach allows us to define both a denotational semantics, a constructive semantics, and a Kahn Process Network semantics for hybrid systems, thus establishing simulation engines on a sound but flexible mathematical foundation. These semantics offer a clear distinction between the concerns of the numerical analyst (solving differential equations) and those of the computer scientist (generating execution schemes). We also discuss a number of practical and fundamental issues in hybrid system modelers that give rise to non-reproducibility of results, non-determinism, and undesirable side effects. Of particular importance are cascaded mode changes (also called “zero-crossings” in the context of hybrid systems modelers).

Non-Standard Semantics of Hybrid Systems Modelers

Albert Benveniste Timothy Bourke
Benoît Caillaud Marc Pouzet

INRIA Rennes and ENS Ulm, France

September 14, 2012

Difficulties in Hybrid Systems Modelers

- Cascaded zero-crossings and start'n-kills of ODE/DAE
  - ZC can traverse, tangent, be thick. . . how to define them?
  - cascades: finite? bounded?
  - solver can stop in zero time if initialized on a zero-crossing
  - is this the duty of Continuous or Discrete?

Difficulties in Hybrid Systems Modelers

- Cascaded zero-crossings and start'n-kills of ODE/DAE
  - ZC can traverse, tangent, be thick. . . how to define them?
  - cascades: finite? bounded?
  - solver can stop in zero time if initialized on a zero-crossing
  - is this the duty of Continuous or Discrete?
- Use of a global solver
  - non-interacting subsystems interact!
  - time scales propagate everywhere
  - Hot/Cold restart of solvers

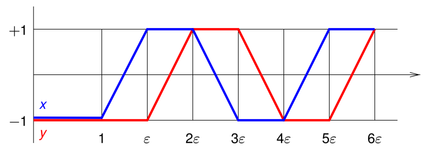
Difficulties in Hybrid Systems Modelers

- Cascaded zero-crossings and start'n-kills of ODE/DAE
  - ZC can traverse, tangent, be thick. . . how to define them?
  - cascades: finite? bounded?
  - solver can stop in zero time if initialized on a zero-crossing
  - is this the duty of Continuous or Discrete?
- Use of a global solver
  - non-interacting subsystems interact!
  - time scales propagate everywhere
  - Hot/Cold restart of solvers
- Slicing Discrete/Continuous is essential
  - strange hybrid D+C Simulink/Stateflow diagrams can be specified they get strange returns from the tool
  - the Modelica consortium has made this a central effort

Some examples 1: infinite cascade

$$\begin{cases} \dot{y} = & 0 \textbf{ init } -1 \quad \textbf{reset}\,[1,-1] \quad \textbf{every up}[x,-x] \\ \dot{x} = & 0 \textbf{ init } -1 \quad \textbf{reset}\,[-1,1,1] \ \textbf{every up}[y,-y,z] \\ \dot{z} = & 1 \textbf{ init } -1 \end{cases}$$

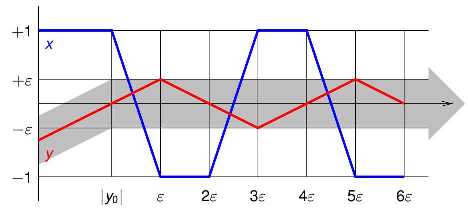
Note that z is just a physical clock. So, such an example can arise with "discrete" systems following the discrete/hybrid classification in force in the community of hybrid systems modelers.



here and subsequently, ε is infinitesimal

Some examples 2: sliding mode

$$\begin{cases} \dot{x} = 0 \textbf{ init } -\operatorname{sgn}(y_0)\ \textbf{reset}\,[-1,1]\ \textbf{every up}[y,-y] \\ \dot{y} = x \textbf{ init } y_0 \end{cases}$$

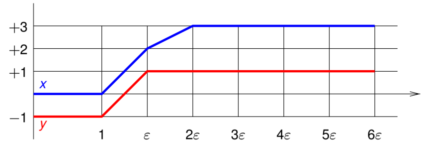


This is a simple form for an ABS system. Corresponding "averaged" system is:

$$\dot{\mathbf{y}} = \begin{cases} -\operatorname{sgn}(y_0), & \text{for the interval } [0, |y_0|\,) \\ 0 & \text{for } [\,|y_0|, \infty), \end{cases}$$

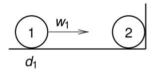
Some examples 3: finite cascade

$$\begin{cases} \dot{x} = 0 \textbf{ init } 0 \textbf{ reset } [\textbf{last}\,(x)+1, \textbf{last}\,(x)+2] \textbf{ every up}[y,z] \\ \dot{z} = 1 \textbf{ init } -1 \\ \dot{y} = 0 \textbf{ init } -1 \textbf{ reset } [1] \textbf{ every up}[z] \end{cases}$$



Here the question is: how should the reset on x and y be performed? Here we have adopted a mirco-step interpretation reflecting causality between the two resets. A different interpretation is often proposed by existing modelers.

Some examples 4: balls on wall



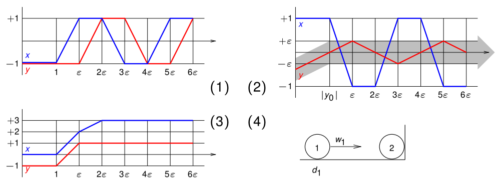
$$\begin{cases} \dot{x}_1 &= v_1 \textbf{ init } d_1 \\ \dot{x}_2 &= v_2 \textbf{ init } d_2 \\ \dot{v}_1 &= 0 \textbf{ init } w_1 \textbf{ reset last}\,(v_2) \textbf{ every up}[x_1 - x_2] \\ \dot{v}_2 &= 0 \textbf{ init } w_2 \textbf{ reset } [\textbf{last}\,(v_1), -\textbf{last}\,(v_2)] \textbf{ every up}[x_1 - x_2, x_2] \end{cases}$$

Here the difficulty is the cascade involving

1. ball 1 hitting ball 2, resulting in ball 2 moving to the right (reset)
2. which causes ball 2 to hit the wall immediately (ODE activated for zero time)
3. resulting in ball 2 moving backward (reset)
4. followed by the symmetric sheme.

Questions

- Can we propose a semantic domain for these (and all) examples?
- Can we use it
  - to identify example (1) as pathological, but not example (2)?
  - to decide on the semantics of example (3)?
  - to give a semantics to example (4)?
- More generally, can we develop a semantic domain to serve as a mathematical basis for the management of (possibly cascaded) zero-crossings?



The great idea: non-standard analysis

Suppose for a while that we can give a formal meaning to the following:

$$\dot{y} = x \quad \text{means, by definition:} \quad \frac{y_{t+\partial} - y_t}{\partial} = x_t$$

where ∂ is infinitesimal

Let's make a trial use of non-standard anaysis.
The ε of our examples will be identified with the above ∂.
By doing so, our drawings become the semantics of cascades and ODEs' semantics is written as transition relations involving ∂.

Non-Standard Time Base

Fix an infinitesimal base step ∂

$$\text{time base} : \mathbb{T} = \{t_n = n\partial \mid n \in {}^*\mathbb{Z}\}$$

$$\text{define } \forall t \in \mathbb{T} : {}^\bullet t = \max\{s \mid s \in \mathbb{T}, s < t\}$$
$$t^\bullet = \min\{s \mid s \in \mathbb{T}, s > t\}$$

$\mathbb{T}$ offers *"the butter and the money of the butter"* (popular french idiom):

(i) $\mathbb{T}$ is totally ordered

(ii) every subset of $\mathbb{T}$ that is bounded from above by a finite (non-standard) number has a unique maximal element

(iii) $\mathbb{T}$ is dense in $\mathbb{R}$

By (i) and (ii) $\mathbb{T}$ looks "discrete"
By (iii), $\mathbb{T}$ looks "continuous"

Non-Standard Time Base

$$\mathbb{T} = \{t_n = n\partial \mid n \in {}^*\mathbb{Z}\}$$
$$\forall t \in \mathbb{T} : {}^\bullet t = \max\{s \mid s \in \mathbb{T}, s < t\}$$
$$t^\bullet = \min\{s \mid s \in \mathbb{T}, s > t\}$$

ODE:

$$\underbrace{\dot{x} = f(x,u)}_{\text{(possibly not well defined)}} \quad \longleftrightarrow \quad \underbrace{x_t = x_{^\bullet t} + \partial \times f(x_{^\bullet t}, u_{^\bullet t})}_{\text{(always well defined)}}$$

Streams of events generated by the zero-crossings of x:

$$\zeta_x =_{\text{def}} \{t \in \mathbb{T} \mid x_{^\bullet t} < 0 \wedge x_t \geq 0\} \quad \text{(always well defined)}$$
$$\approx \{s \in \mathbb{R} \mid x_{s_-} < 0 \wedge x_s \geq 0\} \quad \text{(possibly not well defined)}$$

Cascades following t:

$$t, {}^\bullet t, {}^{\bullet\bullet} t, {}^{\bullet\bullet\bullet} t, \ldots \quad \longleftrightarrow \quad ????$$

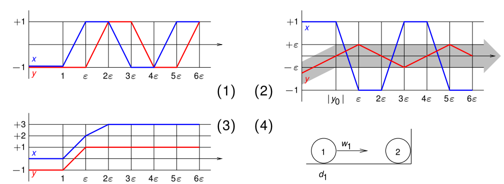
No standard counterpart using $\mathbb{R}$; $\mathbb{R} \times \mathbb{N}$ sufficient for finite cascades ("super-dense" time). Some cascades are worse (example 1) and cannot find their semantics in super-dense time

Back to the examples

Can we propose a semantic domain for these (and all) examples?
The drawings show the non-standard semantics with $\partial := \varepsilon$

Can we use it — yes we can

- to identify example (1) as pathological? — easy
- to identify example (2) as non-pathological? — less easy
- to decide on the semantics of example (3)? — easy
- to give a semantics to example (4)? — subtle



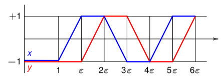
Back to the examples

Can we propose a semantic domain for these (and all) examples?
The drawings show the non-standard semantics with $\partial := \varepsilon$

Can we use it — yes we can

- to identify example (1) as pathological? — easy

The figure shows the non-standard semantics. The system oscillates for the whole $\mathbb{T}$ ("for ever"), for a non-standard number of times. Note that the sequence of instants $n\varepsilon$ tends to infinity because n can itself be an infinite non-standard integer. This trajectory possesses no standardisation.



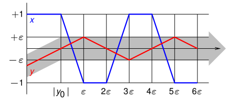
Back to the examples

Can we propose a semantic domain for these (and all) examples?
The drawings show the non-standard semantics with $\partial := \varepsilon$

Can we use it — yes we can

- to identify example (2) as non-pathological? — less easy

The figure shows the non-standard semantics. The system oscillates for the whole $\mathbb{T}$ ("for ever"), for a non-standard number of times. However, while the blue trajectory oscillates between -1 and $+1$, the red one oscillates between $-\varepsilon$ and $+\varepsilon$, and it can be proved that the standard part of this trajectory is indeed the thick grey polyline in which ε is intepreted as zero.



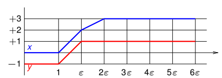
Back to the examples

Can we propose a semantic domain for these (and all) examples?
The drawings show the non-standard semantics with $\partial := \varepsilon$

Can we use it — yes we can

- to decide on the semantics of example (3)? — easy

The figure shows the non-standard semantics. The system has a first zero-crossing at $t = 1$, which causes a second one to occur on the blue trajectory at $t = 1 + \varepsilon$. This yields a classical super-dense time semantics.



Back to the examples

Can we propose a semantic domain for these (and all) examples?
The drawings show the non-standard semantics with $\partial := \varepsilon$

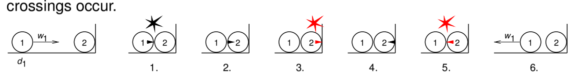
Can we use it — yes we can

- to give a semantics to example (4)? — subtle

Non-standard semantics of the colliding balls example:

1. $t = \partial$, $x_1 = \partial \cdot w_1 > 0 \Rightarrow$ z-c (zero-crossing) on $x_1 - x_2$.
2. $\Rightarrow$ at $t = 2\partial$ balls exchange velocities: $v_1 = 0$ and $v_2 = w_1$.
3. $t = 3\partial$, $x_1 = 2\partial \cdot w_1$ and $x_2 = \partial \cdot w_1 \Rightarrow$ ODE has immediate z-c on x_2
4. $t = 4\partial$, $x_1 = x_2 = 2\partial \cdot w_1$, $v_1 = 0$ and $v_2 = -w_1$.
5. $t = 5\partial$, $x_1 = 2\partial \cdot w_1$ and $x_2 = \partial \cdot w_1 \Rightarrow$ z-c $x_1 - x_2$
6. $\Rightarrow$ at $t = 6\partial$, $x_1 = 2\partial \cdot w_1$, $x_2 = 0$, $v_1 = -w_1$ and $v_2 = 0$.

Then, ball 1 moves toward $-\infty$ according to the ODEs and no further zero-crossings occur.



What is needed to establish the above on firm bases?

Two things are needed:

1. To establish on firm bases the juggling we plaid with ε and ∂ without care for both continuous and discrete dynamics
2. To relate it to "normal life semantics" where discrete dynamics, continuous dynamics and hybrid dynamics may or may not be well defined (existence/uniqueness/nonzenoness of solutions), not to speak about composition thereof

What is needed to establish the above on firm bases?

Two things are needed:

1. To establish on firm bases the juggling we plaid with ε and ∂ without care for both continuous and discrete dynamics
2. To relate it to "normal life semantics" where discrete dynamics, continuous dynamics and hybrid dynamics may or may not be well defined (existence/uniqueness/nonzenoness of solutions), not to speak about composition thereof

Answers to the above is provided by:

1. Non-Standard analysis *seriously* (don't be afraid...)
2. Standardisation of non-standard entities

What is needed to establish the above on firm bases?

- What Non-Standard semantics yields:
  1. NS semantics is always defined; it involves "quasi-discrete" dynamical systems indexed by $\mathbb{T} = \partial \times {}^*\mathbb{N}$ (NS semantics is thus ∂-dependent)

     $$\text{hybrid system program} \;\longrightarrow_\partial\; \text{NS semantics}$$

  2. Systems always compose

What is needed to establish the above on firm bases?

- What Non-Standard semantics yields:
  1. NS semantics is always defined; it involves "quasi-discrete" dynamical systems indexed by $\mathbb{T} = \partial \times {}^*\mathbb{N}$ (NS semantics is thus ∂-dependent)

     $$\text{hybrid system program} \;\longrightarrow_\partial\; \text{NS semantics}$$

  2. Systems always compose
- Standardisation principle: There exists a standardisation map

  $$\text{hybrid system program} \;\longrightarrow_\partial\; \text{NS semantics} \;\mapsto\; \text{S semantics}$$

  such that
  1. it is a partial map (sometimes NS systems have no S counterpart)
  2. when standardisation exists, then the above end-to-end map does not depend on ∂: NS semantics is intrinsic
  3. when system composition is well defined in the S domain, then we get commutative diagrams

Non-Standard Analysis

A bit of history

- Born in 1961 from Abraham Robinson, then developed by a small community of mathematicians.
- Proposed as a conservative enhancement of Zermelo-Fränkel set theory; some fancy axioms and principles; nice for the adicts
- Subject of controversies: what does it do for you that you cannot do using our brave analysis with $\forall\varepsilon\exists\eta\dots$?
- 1988: a nice presentation of the topic by T. Lindstrom, kind of "*non-standard analysis for the axiom-averse*"
- 2006: used in Simon Bliudze PhD where he proposes the counterpart of a "Turing machine" for hybrid systems (supervised by D. Krob)

Non-Standard Analysis

A bit of history

- Born in 1961 from Abraham Robinson, then developed by a small community of mathematicians.
- Proposed as a conservative enhancement of Zermelo-Fränkel set theory; some fancy axioms and principles; nice for the adicts
- Subject of controversies: what does it do for you that you cannot do using our brave analysis with $\forall\varepsilon\exists\eta\dots$?
- 1988: a nice presentation of the topic by T. Lindstrom, kind of "*non-standard analysis for the axiom-averse*"
- 2006: used in Simon Bliudze PhD where he proposes the counterpart of a "Turing machine" for hybrid systems (supervised by D. Krob)

Why is non-standard analysis interesting for the computer scientist?

- it offers a step-based view of continuous and hybrid systems
- it is non-effective; still, it is amenable to symbolic executions and can thus be used for symbolic analyses at compile (and even run) time

Non-Standard Analysis

The aim

- to augment $\mathbb{R} \cup \{\pm\infty\}$ with elements that are infinitely close to x for each $x \in \mathbb{R}$, call ${}^*\mathbb{R}$ the result;
- ${}^*\mathbb{R}$ should obey the same algebra as $\mathbb{R}$: total order, $+, \times, \dots$ any $f : \mathbb{R} \mapsto \mathbb{R}$ extends to ${}^*f : {}^*\mathbb{R} \mapsto {}^*\mathbb{R}$, etc

Idea:

- mimic the construction of $\mathbb{R}$ from $\mathbb{Q}$ as Cauchy sequences; candidates for infinitesimals include:

$$\text{close to } 0 \quad : \quad \left\{\frac{1}{\sqrt{n}}\right\} > \left\{\frac{1}{n}\right\} > \left\{\frac{1}{n^2}\right\} > 0$$

$$\text{close to } +\infty \quad : \quad \{\sqrt{n}\} < \{n\} < \{n^2\}$$

Non-Standard Analysis

The aim

- to augment $\mathbb{R} \cup \{\pm\infty\}$ with elements that are infinitely close to x for each $x \in \mathbb{R}$, call $^*\mathbb{R}$ the result;
- $^*\mathbb{R}$ should obey the same algebra as $\mathbb{R}$: total order, $+, \times, \ldots$ any $f : \mathbb{R} \mapsto \mathbb{R}$ extends to $^*f : {^*\mathbb{R}} \mapsto {^*\mathbb{R}}$, etc

Are we done? Not quite so:

- Sequences of reals $\{x_n\}$ generally do not converge
- Two sequences $\{x_n\}$ and $\{y_n\}$ converging to 0 may be s.t. $\{n \mid x_n > y_n\}$, $\{n \mid x_n < y_n\}$, and $\{n \mid x_n = y_n\}$ are all infinite sets

Non-Standard Analysis

The aim

- to augment $\mathbb{R} \cup \{\pm\infty\}$ with elements that are infinitely close to x for each $x \in \mathbb{R}$, call $^*\mathbb{R}$ the result;
- $^*\mathbb{R}$ should obey the same algebra as $\mathbb{R}$: total order, $+, \times, \ldots$ any $f : \mathbb{R} \mapsto \mathbb{R}$ extends to $^*f : {^*\mathbb{R}} \mapsto {^*\mathbb{R}}$, etc

Are we done? Not quite so:

- Sequences of reals $\{x_n\}$ generally do not converge
- Two sequences $\{x_n\}$ and $\{y_n\}$ converging to 0 may be s.t. $\{n \mid x_n > y_n\}$, $\{n \mid x_n < y_n\}$, and $\{n \mid x_n = y_n\}$ are all infinite sets

Lindström: partition subsets of $\mathbb{N}$ into neglectible/non-neglectible ones, so that:

- finite or empty subsets are all neglectible
- neglectible sets are stable under finite unions
- for any subset P, either P or its complement is non-neglectible

Having such a decision mechanism relies on Zorn Lemma ($\approx$ axiom of choice) and is formalized as explained next.

Non-Standard Analysis: the idea of Lindstrom

Pick $\mathcal{F}$ a free ultrafilter of $\mathbb{N}$:

- $\emptyset \notin \mathcal{F}$, $\mathcal{F}$ stable by intersection
- $P \in \mathcal{F}$ and $P \subseteq Q$ implies $Q \in \mathcal{F}$
- P finite implies $P \notin \mathcal{F}$
- either P or $\mathbb{N} - P$ belongs to $\mathcal{F}$

Existence of $\mathcal{F}$ follows from Zorn's lemma ($\Leftrightarrow$ axiom of choice)

Non-Standard Analysis: the idea of Lindstrom

Pick $\mathcal{F}$ a free ultrafilter of $\mathbb{N}$:

- $\emptyset \notin \mathcal{F}$, $\mathcal{F}$ stable by intersection
- $P \in \mathcal{F}$ and $P \subseteq Q$ implies $Q \in \mathcal{F}$
- P finite implies $P \notin \mathcal{F}$
- either P or $\mathbb{N} - P$ belongs to $\mathcal{F}$

Existence of $\mathcal{F}$ follows from Zorn's lemma ($\Leftrightarrow$ axiom of choice)

Define:

$$\mu(P) = \text{ if } P \in \mathcal{F} \text{ then 1 else 0}$$

- $P \cap Q = \emptyset \Rightarrow \mu(P \cup Q) = \mu(P) + \mu(Q)$; $\mu(\mathbb{N}) = 1$
- P finite implies $\mu(P) = 0$: P is neglectible

Non-Standard Analysis: the idea of Lindstrom

$(x_n), (x'_n) \in \mathbb{R}^{\mathbb{N}}$, define $(x_n) \approx (x'_n)$ iff set $\{n \mid x_n \neq x'_n\}$ is neglectible

$$^*\mathbb{R} = \mathbb{R}^{\mathbb{N}} / \approx \quad ; \quad \text{elements of } {}^*\mathbb{R} \text{ are written } [x_n]$$

- For any two (x_n), (y_n) exactly one among the sets $\{n \mid x_n > y_n\}$, $\{n \mid x_n < y_n\}$, $\{n \mid x_n = y_n\}$, is non-neglectible $\Rightarrow$ any two sequences can always be compared modulo $\approx$
- By pointwise extension, a 1[st]-order formula is true over $^*\mathbb{R}$ iff it is true over $\mathbb{R}$: this is known as the transfer principle
- Say that
$$x = st([x_n]) \text{ if } x_n \to x \text{ modulo neglectible sets}$$

Non-Standard Analysis: the idea of Lindstrom

Theorem: [standardisation] Any non-standard real $[x_n]$ possesses a unique standard part

Proof:

1. Pick
$$x = \sup\{u \in \mathbb{R} \mid [u] \leq [x_n]\}$$
where $[u]$ denotes the constant sequence equal to u.
2. Since $[x_n]$ is finite, x exists; remains to show that $[x_n] - x$ is infinitesimal.
3. If this is not true,
   - then there exists $y \in \mathbb{R}, y > 0$ such that $y < |x - [x_n]|$,
   - that is, either $x < [x_n] - [y]$ or $x > [x_n] + [y]$,
   - which both contradict the definition of x.
4. The uniqueness of x is clear, thus we can define $st([x_n]) = x$.

Infinite non-standard reals have no standard part in $\mathbb{R}$.

Integrals, ODE, and the Standardisation Principle

- internal functions and sets by pointwise extension:
$$\forall n, g_n : \mathbb{R} \mapsto \mathbb{R} \text{ yields } [g_n] : {}^*\mathbb{R} \mapsto {}^*\mathbb{R} \text{ by } [g_n]([x_n]) = [g_n(x_n)]$$
- Pick ∂ infinitesimal and $N \in {}^*\mathbb{N}$ such that $(N-1)\partial < 1 \leq N\partial$, and consider the set
$$T = \{0, \partial, 2\partial, \ldots, (N-1)\partial, 1\}$$
By definition, if $\partial = [d_n]$, then $N = [N_n]$ with $N_n = \frac{1}{d_n}$ and $T = [T_n]$ with
$$T_n = \{0, \partial, 2\partial, \ldots, (N_n - 1)\partial, 1\}$$
- For $f : [0,1] \mapsto \mathbb{R}$ a continuous function and $^*f = [f, f, \ldots]$ its non-standard version
$$\left[\sum_{t \in T_n} \tfrac{1}{N_n} f(t_n)\right] = \sum_{t \in T} \tfrac{1}{N} {}^*f(t)$$

Integrals, ODE, and the Standardisation Principle

- internal functions and sets by pointwise extension:
$$\forall n, g_n : \mathbb{R} \mapsto \mathbb{R} \text{ yields } [g_n] : {}^*\mathbb{R} \mapsto {}^*\mathbb{R} \text{ by } [g_n]([x_n]) = [g_n(x_n)]$$
- Pick ∂ infinitesimal and $N \in {}^*\mathbb{N}$ such that $(N-1)\partial < 1 \leq N\partial$, and consider the set
$$T = \{0, \partial, 2\partial, \ldots, (N-1)\partial, 1\}$$
By definition, if $\partial = [d_n]$, then $N = [N_n]$ with $N_n = \frac{1}{d_n}$ and $T = [T_n]$ with
$$T_n = \{0, \partial, 2\partial, \ldots, (N_n - 1)\partial, 1\}$$
- For $f : [0,1] \mapsto \mathbb{R}$ a continuous function and $^*f = [f, f, \ldots]$ its non-standard version
$$st\left(\left[\sum_{t \in T_n} \tfrac{1}{N_n} f(t_n)\right]\right) = st\left(\sum_{t \in T} \tfrac{1}{N} {}^*f(t)\right) = \int_0^1 f(t)dt$$

we claim this

Integrals, ODE, and the Standardisation Principle

Theorem: [standardisation] if $f : [0,1] \to \mathbb{R}$ is continuous, then

$$st\left(\left[\sum_{t \in T_n} \frac{1}{N_n} f(t_n)\right]\right) = st\left(\sum_{t \in T} \frac{1}{N} {}^{*}f(t)\right) = \int_0^1 f(t)dt$$

Proof: If $f : \mathbb{R} \to \mathbb{R}$ is a standard function, we always have

$$\sum_{t \in T} \frac{1}{N} {}^{*}f(t) = \left[\sum_{t \in T_n} \frac{1}{N_n} f(t_n)\right] \qquad (1)$$

Now, f continuous implies $\sum_{t \in T_n} \frac{1}{N_n} f(t_n) \to \int_0^1 f(t)dt$, so, by definition of non-standard reals,

$$\int_0^1 f(t)dt = st\left(\sum_{t \in T} \frac{1}{N} {}^{*}f(t)\right) \qquad (2)$$

Integrals, ODE, and the Standardisation Principle

Theorem: [standardisation] if $f : [0,1] \to \mathbb{R}$ is continuous, then

$$st\left(\left[\sum_{t \in T_n} \frac{1}{N_n} f(t_n)\right]\right) = st\left(\sum_{t \in T} \frac{1}{N} {}^{*}f(t)\right) = \int_0^1 f(t)dt$$

Proof: If $f : \mathbb{R} \to \mathbb{R}$ is a standard function, we always have

$$\sum_{t \in T} \frac{1}{N} {}^{*}f(t) = \left[\sum_{t \in T_n} \frac{1}{N_n} f(t_n)\right] \qquad (1)$$

Now, f continuous implies $\sum_{t \in T_n} \frac{1}{N_n} f(t_n) \to \int_0^1 f(t)dt$, so, by definition of non-standard reals,

$$\int_0^1 f(t)dt = st\left(\sum_{t \in T} \frac{1}{N} {}^{*}f(t)\right) \qquad (2)$$

- Thus, if f is smooth so that its Riemann integral is well defined, then any non-standard formulation of the integral of f has $\int_0^1 f(t)dt$ as its standard part
- The same philosophy applies to ODEs and Hybrid Systems

Integrals, ODE, and the Standardisation Principle

For every $0 < t \leq 1$:

$\int_0^t f(u)du = st\left(\sum_{u \in T, u \leq t} \frac{1}{N} {}^{*}f(t)\right)$ (Non-standard Riemann integral)

Integrals, ODE, and the Standardisation Principle

For every $0 < t \leq 1$:

$$\int_0^t f(u)du = st\left(\sum_{u\in T, u\leq t} \tfrac{1}{N} {}^*f(t)\right) \quad \text{(Non-standard Riemann integral)}$$

Set $\partial = \frac{1}{N}$ and consider the ODE $\dot{x} = f(x,t)$, x_0, in integral form

$$\begin{aligned} x(t) &= x_0 + \int_0^t f(x(u),u)du \quad \text{(with the needed smoothness)} \\ x(t) &= st\left(x_0 + \sum_{k\,:\,0\leq k\partial\leq t} \tfrac{1}{N} {}^*f({}^*x(k\partial), k\partial)\right) \end{aligned}$$

Integrals, ODE, and the Standardisation Principle

For every $0 < t \leq 1$:

$$\int_0^t f(u)du = st\left(\sum_{u\in T, u\leq t} \tfrac{1}{N} {}^*f(t)\right) \quad \text{(Non-standard Riemann integral)}$$

Set $\partial = \frac{1}{N}$ and consider the ODE $\dot{x} = f(x,t)$, x_0, in integral form

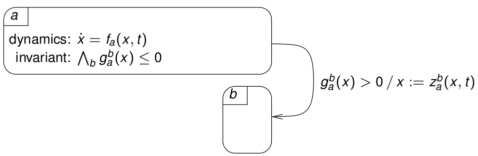
$$\begin{aligned} x(t) &= x_0 + \int_0^t f(x(u),u)du \quad \text{(with the needed smoothness)} \\ x(t) &= st\left(x_0 + \sum_{k\,:\,0\leq k\partial\leq t} \tfrac{1}{N} {}^*f({}^*x(k\partial), k\partial)\right) \\ &= st\left({}^*x(s_t)\right)\ ,\ \text{for } s_t = \max\{t_k \mid t_k = k\partial \leq t\} \end{aligned} \tag{3}$$

where ${}^*x$ is the non-standard semantics of the above ODE with time basis ∂:

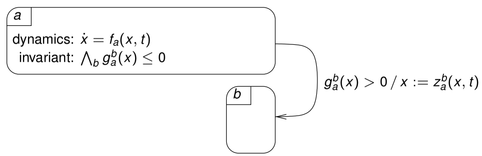
$$\begin{cases} {}^*x(t_k) = {}^*x(t_{k-1}) + \partial \times f({}^*x(t_{k-1}), t_{k-1}) \\ {}^*x(t_0) = x_0 \end{cases} \tag{4}$$

Theorem: [standardisation]
(4) is always defined as a non-standard dynamical system
(3) only holds if the ODE has a solution

Non-Standard Hybrid Systems, Standardisation Principle



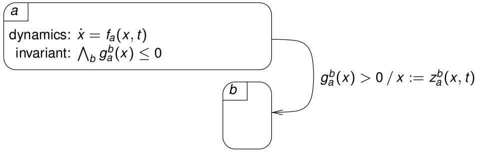
Non-Standard Hybrid Systems, Standardisation Principle



Standard semantics:

- spending standard > 0 duration within modes: ODE
- finite cascades of mode changes: super-dense time $(t,n) \in \mathbb{R} \times \mathbb{N}$

Non-Standard Hybrid Systems, Standardisation Principle



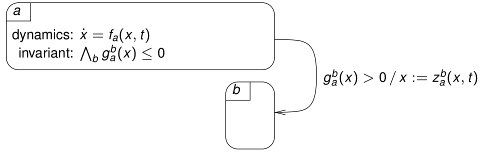
Standard semantics:

- spending standard > 0 duration within modes: ODE
- finite cascades of mode changes: super-dense time $(t, n) \in \mathbb{R} \times \mathbb{N}$

Non-standard (∂-dependent) semantics:

- spending $\geq$0 duration within modes: non-standard ODE
- cascades of mode changes: “discrete” dynamics indexed by $\mathbb{T}$

Non-Standard Hybrid Systems, Standardisation Principle



Standard semantics:

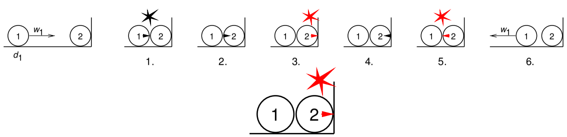
- spending standard > 0 duration within modes: ODE
- finite cascades of mode changes: super-dense time $(t, n) \in \mathbb{R} \times \mathbb{N}$

Non-standard (∂-dependent) semantics:

- spending $\geq$0 duration within modes: non-standard ODE
- cascades of mode changes: “discrete” dynamics indexed by $\mathbb{T}$

Theorem: [standardisation] if the S semantics is well-defined, then it is the standardisation of the NS (∂-dependent) semantics, for any choice of ∂

Non-Standard Hybrid Systems, Standardisation Principle (extended)

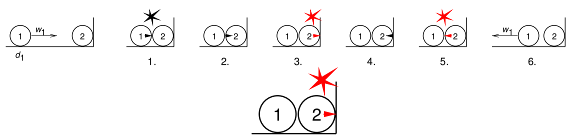


In this example, we successively have, within an infinitesimal period of time:

1. a first cascade of z-c (a hit causing changes in velocities)
2. the launching of an ODE with an immediate z-c
3. another cascade of z-c, followed by the symmetric scheme.

Provided that such a cascade of {z-c$+$ODE micro-steps} remains finite, a super-dense time semantics can be given. Execution is by executing the symbolic non-standard semantics: Extended Standardisation Principle.

Non-Standard Hybrid Systems, Standardisation Principle (extended)



Non-standard symbolic simulation of the colliding balls example:

1. $t = \partial$, $x_1 = \partial \cdot w_1 > 0 \Rightarrow$ z-c (zero-crossing) on $x_1 - x_2$.
2. $\Rightarrow$ at $t = 2\partial$ balls exchange velocities: $v_1 = 0$ and $v_2 = w_1$.
3. $t = 3\partial$, $x_1 = 2\partial \cdot w_1$ and $x_2 = \partial \cdot w_1 \Rightarrow$ ODE has immediate z-c on x_2
4. $t = 4\partial$, $x_1 = x_2 = 2\partial \cdot w_1$, $v_1 = 0$ and $v_2 = -w_1$.
5. $t = 5\partial$, $x_1 = 2\partial \cdot w_1$ and $x_2 = \partial \cdot w_1 \Rightarrow$ z-c $x_1 - x_2$
6. $\Rightarrow$ at $t = 6\partial$, $x_1 = 2\partial \cdot w_1$, $x_2 = 0$, $v_1 = -w_1$ and $v_2 = 0$.

The SimpleHybrid mini-language and its semantics

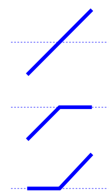
$$\mathbb{T} =_{\text{def}} \{n\partial\}_{n\in{}^*\mathbb{N}} \qquad {}^\bullet(n\partial) = (n-1)\partial$$
$$^\bullet x_t =_{\text{def}} x_{^\bullet t} \qquad (n\partial)^\bullet = (n+1)\partial$$

statement	transition relation
$y = f(x)$	$y = f(x)$
$y =$ **last** (x) **init** y_0	$y = {}^\bullet x$ **init** y_0
$\zeta =$ **up**(x)	$\zeta^\bullet = ([{}^\bullet x{<}0] \wedge [x{\geq}0]) \vee ([{}^\bullet x{\leq}0] \wedge [x{>}0])$
$\dot{y} = x$ **init** y_0 **reset** z	on $\tau \setminus \tau_z : y = {}^\bullet y + \partial \times {}^\bullet x$ on $\tau_z : y = z$
$y = x$ **every** ζ **init** y_0	before $\zeta : y = y_0$ on $\zeta : y = x$
$y =$ **pre** (x) **init** y_0	$\tau_y = \tau_x$ before $\min(\tau_y) : y = y_0$ on $\tau_y : y = {}^\bullet x$
$S_1 \parallel S_2$	conjunction

The SimpleHybrid mini-language and its semantics

$$\mathbb{T} =_{\text{def}} \{n\partial\}_{n\in{}^*\mathbb{N}} \qquad {}^\bullet(n\partial) = (n-1)\partial$$
$$^\bullet x_t =_{\text{def}} x_{^\bullet t} \qquad (n\partial)^\bullet = (n+1)\partial$$

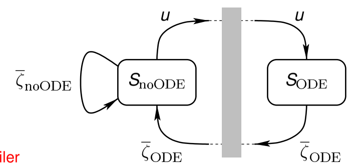
statement	transition relation
$y = f(x)$	$y = f(x)$
$y =$ **last** (x) **init** y_0	$y = {}^\bullet x$ **init** y_0
$\zeta =$ **up**(x)	$\zeta^\bullet = ([{}^\bullet x{<}0] \wedge [x{\geq}0]) \vee ([{}^\bullet x{\leq}0] \wedge [x{>}0])$
$\dot{y} = x$ **init** y_0 **reset** z	on $\tau \setminus \tau_z : y = {}^\bullet y + \partial \times {}^\bullet x$ on $\tau_z : y = z$
$y = x$ **every** ζ **init** y_0	before $\zeta : y = y_0$ on $\zeta : y = x$
$y =$ **pre** (x) **init** y_0	$\tau_y = \tau_x$ before $\min(\tau_y) : y = y_0$ on $\tau_y : y = {}^\bullet x$
$S_1 \parallel S_2$	conjunction



three types of zero-crossing
no need for left/right limit

all ZC + aborting ODE in S: $\overline{\zeta}_S$

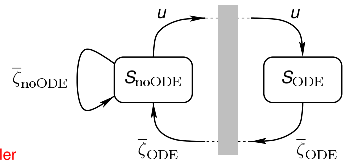
Slicing



discrete compiler

ODE solver

Slicing



discrete compiler — ODE solver

statement of S	Assigned to S_{noODE}	Assigned to S_{ODE}
$y = f([x])$	on $\overline{\zeta}_S : y = f([x])$	outside $\overline{\zeta}_S : y = f([x])$
$y =$ **last** (x)	on $\overline{\zeta}_S : y =$ **last** (x)	outside $\overline{\zeta}_S : y =$ **last** (x)
$\zeta =$ **up**(x)		$\zeta =$ **up**(x)
$\dot{y} = x$ **init** y_0 **reset** z	on $\overline{\zeta}_S \setminus \zeta_S : \dot{y} = x$ **init** y_0 **reset** z	outside $\overline{\zeta}_S : \dot{y} = x$ **init** y_0 **reset** z
$y = [x]$ **every** $[\zeta]$ **init** y_0	$y = [x]$ **every** $[\zeta]$ **init** y_0	
$y =$ **pre** (x) **init** y_0	$y =$ **pre** (x) **init** y_0	

Further use of Non-Standard Semantics

- Causality Analysis and Constructive Semantics
  - compilation and code generation
  - clock-aware compilation
  - new application: DAE and index analysis
- Kahn Network semantics (KPN arguments extend to $^*\mathbb{N}$)
  - distributed simulation & multiple solvers
    to avoid unwanted coupling due to adaptive step size

Conclusion

Non-standard semantics is not just for the fun of Albert Benveniste

- it gives a semantics to all syntactically well-formed programs
  - no hand waving, no need for obscure continuity/zeno assumption
  - compositional

  this is what the language designer needs
- provides semantic support for clock-aware causality analysis
  - clock-aware co-simulation (getting rid of global solvers)
  - future: extend to DAE
- provides semantic support for Discrete/Continuous slicing
  - NS symbolic simulation of aborting ODEs
  - future: singular perturbations and multiple time-scales

  Prevents the designer from the need for manual smoothing
  (non compositional because bandwidth-dependent)

You hybrid guys, go learning it!

EQUATIONS, SYNCHRONY, TIME, AND MODES

Edward A. Lee, EECS, UC Berkeley

The key principle behind equation-based languages is that components in a system interact with one another not by reacting to inputs to produce outputs, but rather by asserting relationships between the values of variables that they share. This principle is closely related to key principle behind synchronous-reactive (SR) languages, where the meaning of a composition of components is a fixed-point solution to a system of equations. In both cases, interactions between components is a dialog, with give and take, rather than a monolog. SR languages have been used to model discrete behaviors primarily, whereas equation-based languages, particularly Modelica, have been used to model continuous dynamics primarily. In this talk, I will show how to bridge the two.

Synchronous programs execute a sequence of (conceptually) simultaneous and instantaneous computations. Each step in the sequence is called a "tick" of a conceptual clock that governs the execution. Distinctly lacking, however, is any notion of metric or measurable time in this clock, so there is no foundation in these languages for modeling continuous dynamics. The ticks form a sequence, not a time line. In fact, a correct execution of a synchronous program (conformant with the semantics) can take as much time as it likes between ticks. The intervals need not even be constant or defined.

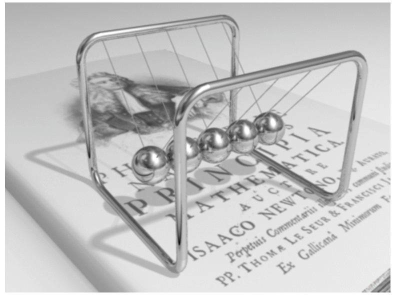
In this talk, I will review the principles of synchronous semantics and show how they can be extended to provide a rigorous foundation for timed systems that do have a metric notion of time. In particular, I will show how discrete-event (DE) and continuous-time models can be built on top of synchronous semantics. I will also introduce a hierarchical multiform time that allows time progress at different rates in different parts of the system, and I will show how the underlying synchronous semantics ensures determinacy and preserves causality. This multiform model of time provides a foundation for modal behaviors and hybrid systems.

Equations, Synchrony, Time, and Modes

Edward A. Lee
Robert S. Pepper Distinguished Professor
UC Berkeley

Collaborative with:
- ***Adam Cataldo***
- ***Patricia Derler***
- ***John Eidson***
- ***Xiaojun Liu***
- ***Eleftherios Matsikoudis***
- ***Haiyang Zheng***

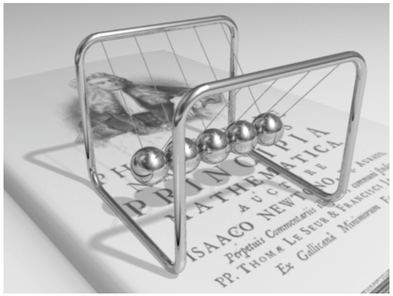
Invited Talk at Workshop:
System Design meets Equation-based Languages: Workshop Program
Lunds, Sweden,
Sept. 18-21



What is the momentum of the middle ball as a function of time?

$$\mathbf{p}(t) = m\mathbf{v}(t)$$

Lee, Berkeley 2



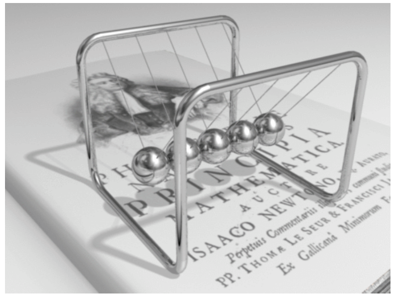
What is the momentum of the middle ball as a function of time?

$$\mathbf{p}(t) = m\mathbf{v}(t)$$

It might seem:

$$\mathbf{v}(t) = 0 \quad \Rightarrow \quad \mathbf{p}(t) = 0$$

Lee, Berkeley 3

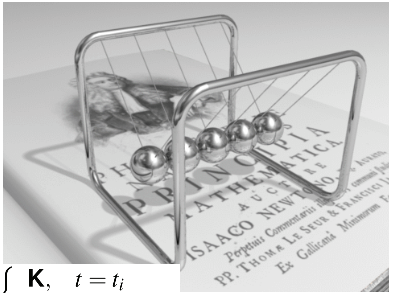


But no, it is:

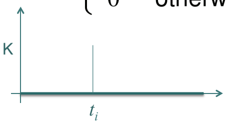
$$\mathbf{v}(t) = \begin{cases} \mathbf{K}, & t = t_i \\ 0 & \text{otherwise} \end{cases}$$

where t_i is the time of collision

Lee, Berkeley 4

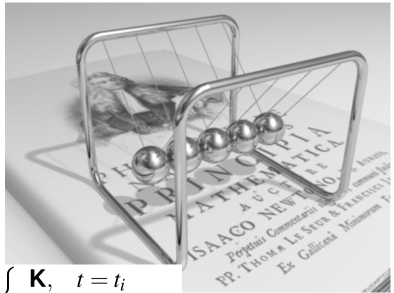


$$\mathbf{v}(t) = \begin{cases} \mathbf{K}, & t = t_i \\ 0 & \text{otherwise} \end{cases}$$

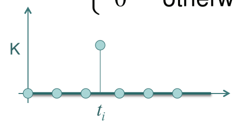


Since position is the integral of velocity, and the integral of **v** is zero, the ball does not move.

Lee, Berkeley 5



$$\mathbf{v}(t) = \begin{cases} \mathbf{K}, & t = t_i \\ 0 & \text{otherwise} \end{cases}$$



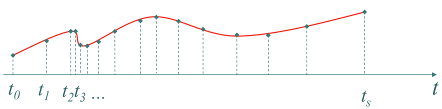
A *discrete* representation of this signal with *samples* is inadequate.

Lee, Berkeley 6

Samples yield discrete signals

A signal $s : T \to D$ *is sampled at tags*

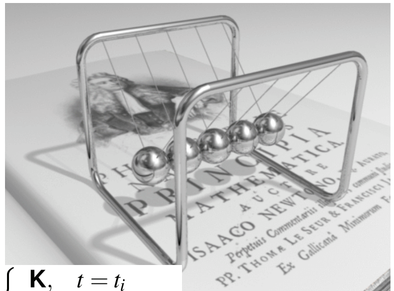
$$\pi(s) = \{t_0, t_1, \ldots\} \subset T$$



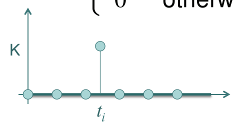
A signal s is *discrete* if there is an *order embedding* from its tag set $\pi(s)$ (the tags for which it is defined and not absent) to the natural numbers (under their usual order).

Note: Benveniste et al. use a different (and less useful?) notion of "discrete."

Lee, Berkeley 7

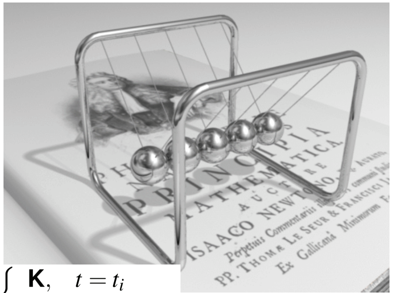


$$\mathbf{v}(t) = \begin{cases} \mathbf{K}, & t = t_i \\ 0 & \text{otherwise} \end{cases}$$

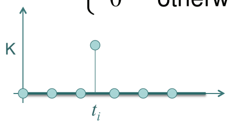


No discrete subset of real-valued times is adequate to unambiguously represent this signal.

Lee, Berkeley 8



$$\mathbf{v}(t) = \begin{cases} \mathbf{K}, & t = t_i \\ 0 & \text{otherwise} \end{cases}$$

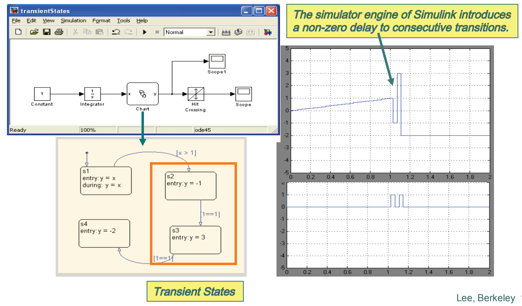


There is no *semantic distinction* between a discrete event and a rapidly varying continuous signal.

Lee, Berkeley 9

Simulink/Stateflow cannot accurately model such events.

In Simulink, a signal can only have one value at a given time. Hence Simulink introduces solver-dependent behavior.



Lee, Berkeley 10

Ptolemy II uses *Superdense Time*

[Maler, Manna, Pnuelli, 92]

for Continuous-Time Signals

$$\mathbf{v}\colon (\mathbb{R} \times \mathbb{N}) \to \mathbb{R}^3$$

Initial value: $\mathbf{v}(t_i, 0) = 0$

Intermediate value: $\mathbf{v}(t_i, 1) = \mathbf{K}$

Final value: $\mathbf{v}(t_i, n) = 0, \quad n \geq 2$

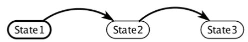
At each **tag**, the signal has *exactly one value*. At each time point, the signal has *a sequence of values*. Signals are *piecewise continuous*, in a well-defined technical sense, a property that makes ODE solvers work well.

11

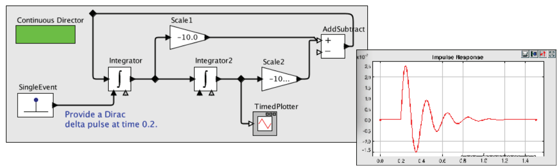
Lee, Berkeley 11

Consequences of using Superdense Time

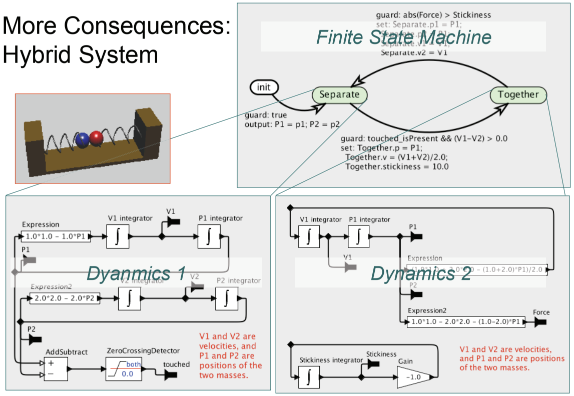
- Transient states are well represented:



- Infinitessimals (even Dirac delta functions):

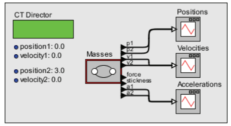


More Consequences: Hybrid System

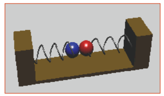


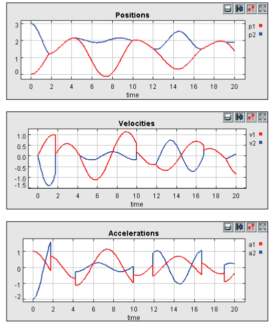
Lee, Berkeley 13

Transitions between modes are instantaneous



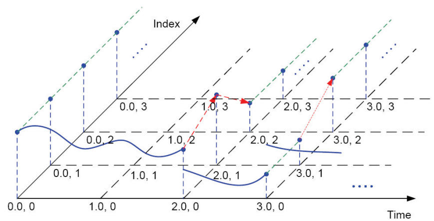
In the signals at the right, the velocities and accelerations proceed through a sequence of values at the times of the collisions and separations.





Lee, Berkeley 14

Superdense Time



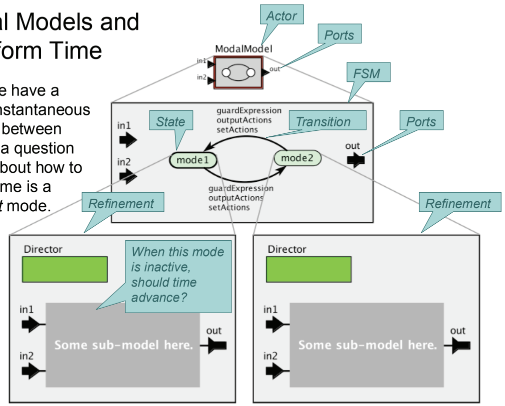
The red arrows indicate value changes between tags, which correspond to discontinuities. Signals are continuous from the left *and* continuous from the right at points of discontinuity.

15

Lee, Berkeley 15

Modal Models and Multiform Time

Once we have a clean, instantaneous handoff between modes, a question arises about how to model time is a *dormant* mode.



The Modal Model Muddle

It's about time

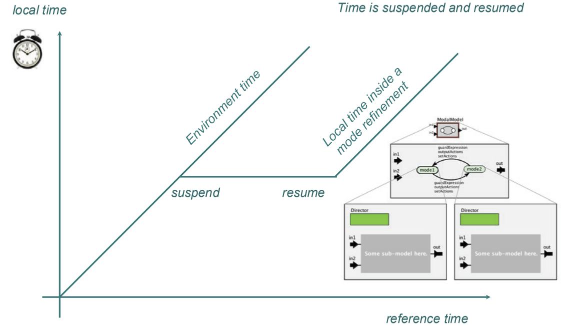
After trying several variants on the semantics of modal time, we settled on this:

A mode refinement has a *local* notion of time. When the mode refinement is inactive, local time does not advance. Local time has a monotonically increasing gap relative to environment time.

Lee, Berkeley 17

MultiForm Time in Ptolemy II

In Ptolemy II Modal Models, Time is suspended and resumed



Lee, Berkeley 18 1

Variants for the Semantics of Modal Time that we Tried or Considered, but that Failed

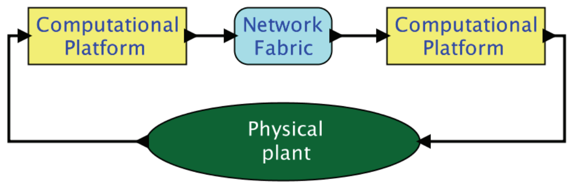
- Mode refinement executes while "inactive" but inputs are not provided and outputs are not observed.
- Time advances while mode is inactive, and mode refinement is responsible for "catching up."
- Mode refinement is "notified" when it has requested time increments that are not met because it is inactive.
- When a mode refinement is re-activated, it resumes from its first missed event.

All of these led to some very strange models…

Final solution: Local time does not advance while a mode is inactive. Monotonically growing gap between local time and environment time.

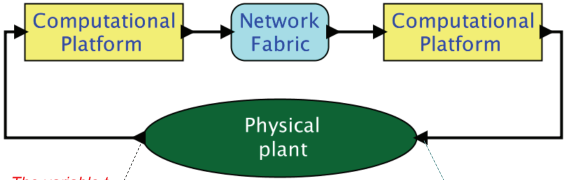
Lee, Berkeley 19

Once we have multiform time, we can build accurate models of cyber-physical systems



Lee, Berkeley 20

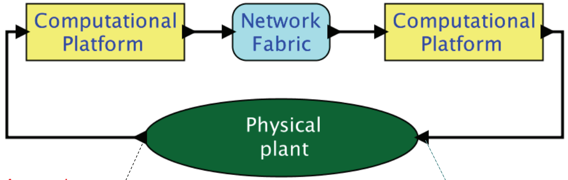
Engineers model physical dynamics using differential-algebraic equations.



$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_0^t \mathbf{T}(\tau)d\tau$$

Lee, Berkeley 21

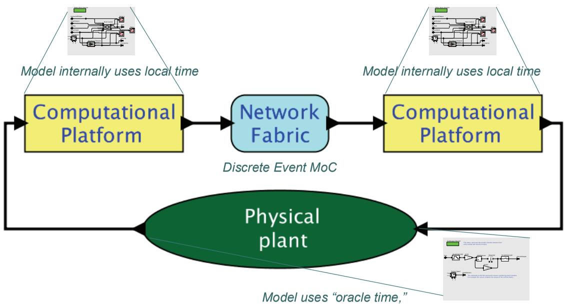
But computational platforms have no access to *t*. Instead, local measurements of time are used.



$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_0^t \mathbf{T}(\tau)d\tau$$

Lee, Berkeley 22

Local time within a hierarchy can advance at different rates.

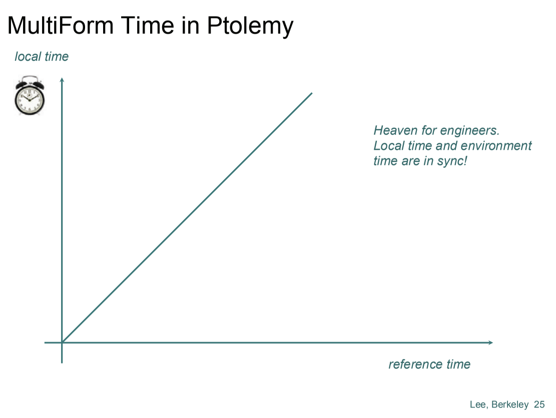


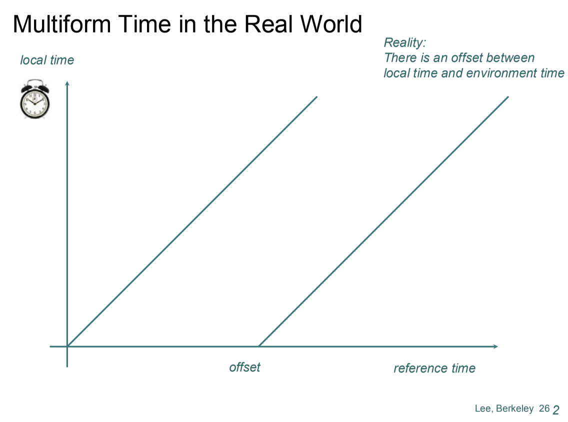
Lee, Berkeley 23

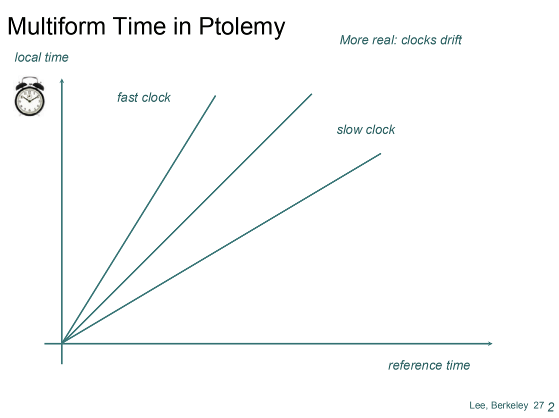
Clocks drift

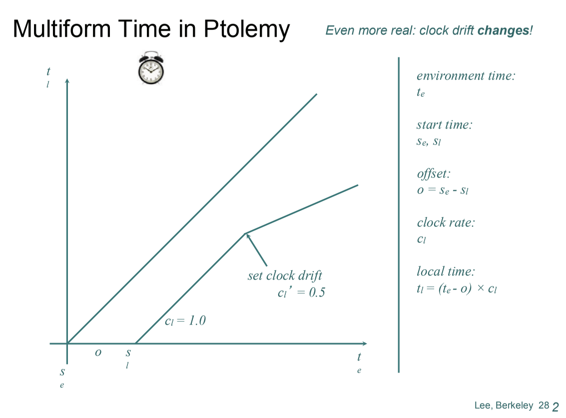
- Fabrication tolerance
- Aging
- Temperature
- Humidity
- Vibrations
- Quality of the quartz.
- Clock drifts measured in "parts per million" or ppm
  1 ppm corresponds to a deviation of 1μs every second

Lee, Berkeley 24


MultiForm Time in Ptolemy
local time
Heaven for engineers.
Local time and environment
time are in sync!
reference time
Lee, Berkeley 25

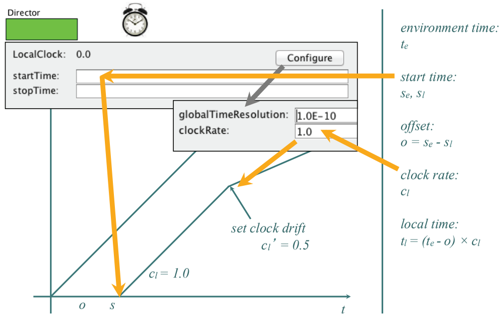

Multiform Time in the Real World
Reality:
There is an offset between
local time and environment time
local time
offset
reference time
Lee, Berkeley 26 2


Multiform Time in Ptolemy
More real: clocks drift
local time
fast clock
slow clock
reference time
Lee, Berkeley 27 2


Multiform Time in Ptolemy
Even more real: clock drift **changes**!
t_l
environment time:
t_e
start time:
s_e, s_l
offset:
$o = s_e - s_l$
clock rate:
c_l
local time:
$t_l = (t_e - o) \times c_l$
set clock drift
$c_l' = 0.5$
$c_l = 1.0$
o
s_l
s_e
t_e
Lee, Berkeley 28 2

Multiform Time in Ptolemy

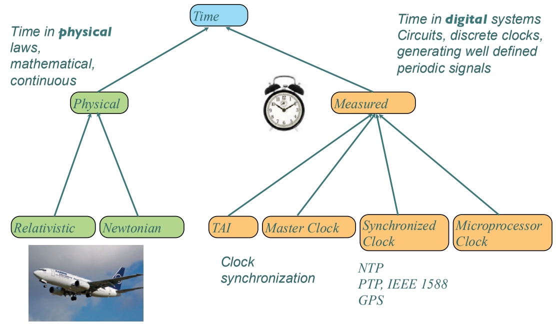
Ptolemy II provides a hierarchy of local clocks



This can be used, for example, to accurately model time synchronization protocols.

Lee, Berkeley 29 2

Multiform Time is Intrinsic!



Source: Patricia Derler and John Eidson

Lee, Berkeley 30

Other Questions about Time:

1. Precision
   - In floating-point formats, precision degrades as magnitude increases
2. Clear Semantics of Simultaneity
   - Requires precise addition and subtraction, e.g.
     (a + b) + c = a + (b + c).
     Floating-point numbers don't have this property.

Floating point numbers are a poor choice for modeling time!

Lee, Berkeley 32

Conclusions

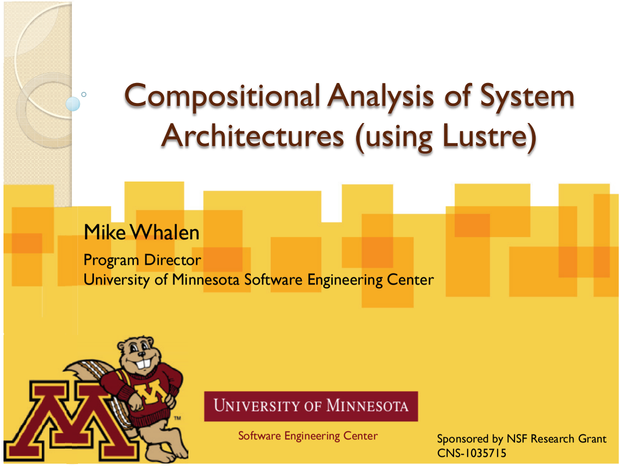
- Modeling time as a simple continuum is not adequate.
  - Superdense time offers clean semantics for instantaneous events.
- Homogeneous time advancing uniformly is not adequate.
  - Hierarchical multiform time enables accurate and practical models of heterogeneous distributed systems.
- Floating point numbers for time are not adequate.
  - A model with invariant precision and precise addition and subtraction is.

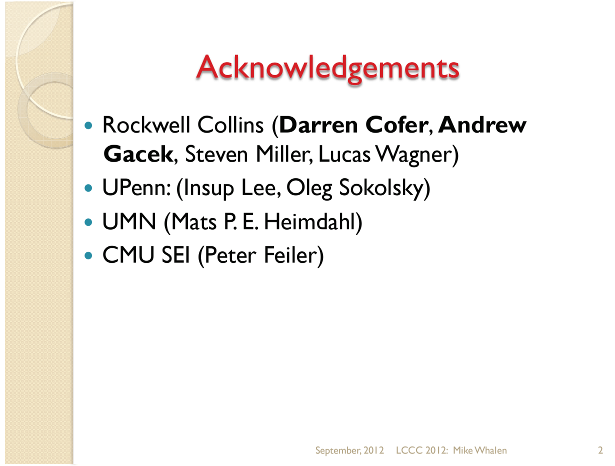
Lee, Berkeley 33

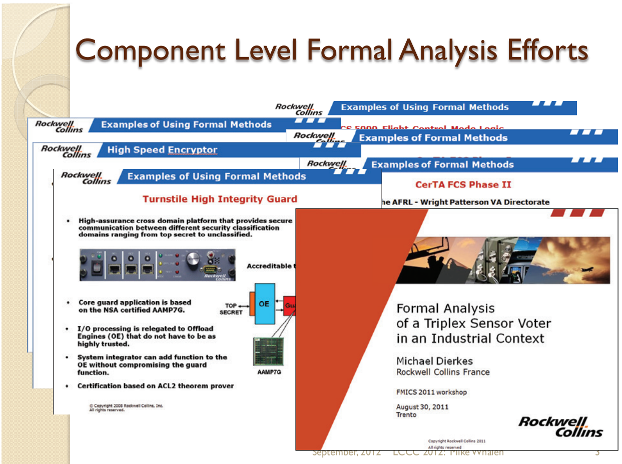
FORMAL MODELING AND ANALYSIS OF SOFTWARE SYSTEMS WITH LUSTRE

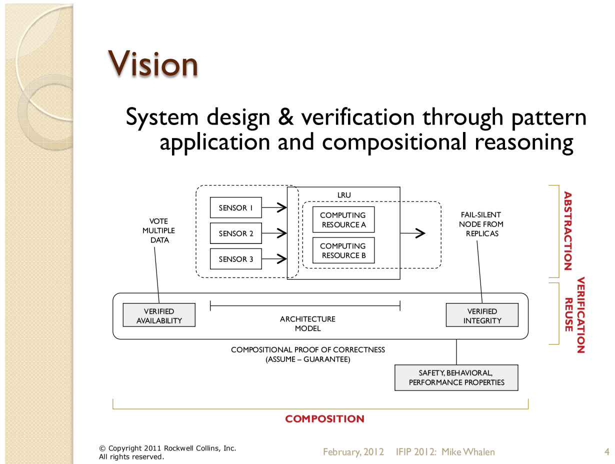
Mike Whalen, University of Minnesota

Rockwell Collins and the University of Minnesota have used the synchronous dataflow language Lustre as a basis for a variety of analyses of industrial critical systems both for component level models written in Simulink and system architectural models written in AADL. This talk describes the approach, several examples of analyzed models as well as several challenges to extend the scale and variety of systems that can be practically analyzed.

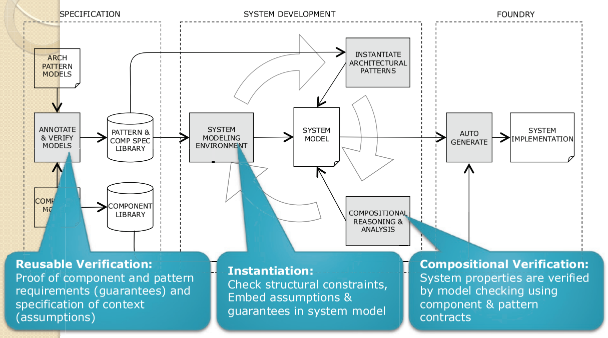

Compositional Analysis of System Architectures (using Lustre)
Mike Whalen
Program Director
University of Minnesota Software Engineering Center
University of Minnesota
Software Engineering Center
Sponsored by NSF Research Grant
CNS-1035715


Acknowledgements
• Rockwell Collins (Darren Cofer, Andrew Gacek, Steven Miller, Lucas Wagner)
• UPenn: (Insup Lee, Oleg Sokolsky)
• UMN (Mats P. E. Heimdahl)
• CMU SEI (Peter Feiler)
September, 2012 LCCC 2012: Mike Whalen
2


Component Level Formal Analysis Efforts
Examples of Using Formal Methods
High Speed Encryptor
Examples of Formal Methods
CerTA FCS Phase II
Turnstile High Integrity Guard
Formal Analysis
of a Triplex Sensor Voter
in an Industrial Context
Michael Dierkes
Rockwell Collins France
FMICS 2011 workshop
August 30, 2011
Trento


Vision
System design & verification through pattern application and compositional reasoning
SENSOR 1
SENSOR 2
SENSOR 3
LRU
COMPUTING RESOURCE A
COMPUTING RESOURCE B
VOTE MULTIPLE DATA
FAIL-SILENT NODE FROM REPLICAS
VERIFIED AVAILABILITY
ARCHITECTURE MODEL
VERIFIED INTEGRITY
COMPOSITIONAL PROOF OF CORRECTNESS (ASSUME – GUARANTEE)
SAFETY, BEHAVIORAL, PERFORMANCE PROPERTIES
ABSTRACTION
VERIFICATION REUSE
COMPOSITION
© Copyright 2011 Rockwell Collins, Inc. All rights reserved.
February, 2012 IFIP 2012: Mike Whalen
4

System verification

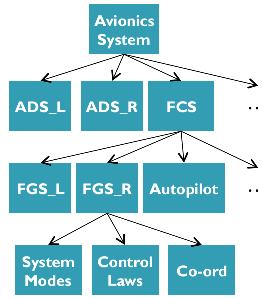


Hierarchical reasoning about systems

- Avionics system req

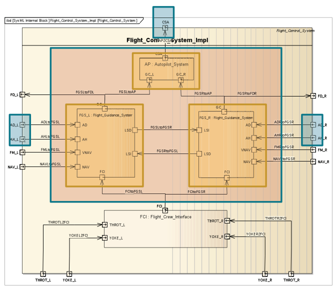
  Under single-fault assumption, GC output transient response is bounded in time and magnitude

- Relies upon
  - Accuracy of air data sensors
  - Control commands from FCS
    - Mode of FGS
    - FGS control law behavior
    - Failover behavior between FGS systems
    - ….
  - Response of Actuators
  - Timing/Lag/Latency of Communications



Compositional Reasoning for Active Standby

- Want to prove a **transient response** property
  - The autopilot will not cause a sharp change in pitch of aircraft.
  - Even when one FGS fails and the other assumes control
- Given assumptions about the **environment**
  - The sensed aircraft pitch from the air data system is within some absolute bound and doesn't change too quickly
  - The discrepancy in sensed pitch between left and right side sensors is bounded.
- and guarantees provided by **components**
  - When a FGS is active, it will generate an acceptable pitch rate
- As well as **facts** provided by pattern application
  - Leader selection: at least one FGS will always be active (modulo one "failover" step)




```
transient_response_1 : assert true ->
  abs(CSA.CSA_Pitch_Delta) < CSA_MAX_PITCH_DELTA ;
transient_response_2 : assert true ->
  abs(CSA.CSA_Pitch_Delta - prev(CSA.CSA_Pitch_Delta, 0.0))
     < CSA_MAX_PITCH_DELTA_STEP ;
```

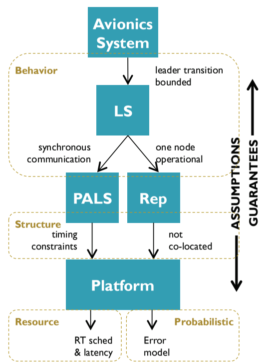

Contracts between patterns and components

- Avionics system requirement

  Under single-fault assumption, GC output transient response is bounded in time and magnitude

- Relies upon
  - Guarantees provided by patterns and components
  - Structural properties of model
  - Resource allocation feasibility
  - Probabilistic system-level failure characteristics

Principled mechanism for "passing the buck"



Contracts

- Derived from Lustre and Property Specification Language (PSL) formalism
  - IEEE standard
  - In wide use for hardware verification
- Assume / Guarantee style specification
  - Assumptions: "Under these conditions"
  - Promises (Guarantees): "… the system will do X"
- Local definitions can be created to simplify properties

```
Contract:

fun abs(x: real) : real = if (x > 0) then x else -x ;

const ADS_MAX_PITCH_DELTA: real = 3.0 ;
const FCS_MAX_PITCH_SIDE_DELTA: real = 2.0 ;
…

property AD_L_Pitch_Step_Delta_Valid =
  true ->
    abs(AD_L.pitch.val - prev(AD_L.pitch.val, 0.0)) <
        ADS_MAX_PITCH_DELTA ;

…

active_assumption: assume some_fgs_active ;

transient_assumption :
  assume AD_L_Pitch_Step_Delta_Valid and
         AD_R_Pitch_Step_Delta_Valid and Pitch_lr_ok ;


transient_response_1 :
  assert true -> abs(CSA.CSA_Pitch_Delta) <
                     CSA_MAX_PITCH_DELTA ;
transient_response_2 :
  assert true ->
      abs(CSA.CSA_Pitch_Delta -
          prev(CSA.CSA_Pitch_Delta, 0.0)) <
               CSA_MAX_PITCH_DELTA_STEP ;
```


Reasoning about contracts

- Notionally: *It is always the case that if the component assumption is true, then the component will ensure that the guarantee is true.*
  - G(A ⇒ P);
- An assumption violation in the past may prevent component from satisfying current guarantee, so we need to assert that the assumptions are true up to the current step:
  - G(H(A) ⇒ P) ;

Reasoning about Contracts

- Given the set of component contracts:
  $\Gamma = \{ G(H(A_c) \Rightarrow P_c) \mid c \in C \}$
- Architecture adds a set of obligations that tie the system assumption to the component assumptions

$$Q = \{H(A_s) \implies P_s\} \cup \{H(A_s) \implies A_c \mid c \in C\}$$

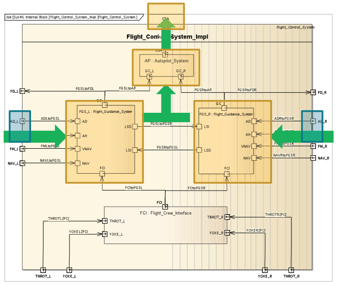
- This process can be repeated for any number of abstraction levels

Composition Formulation

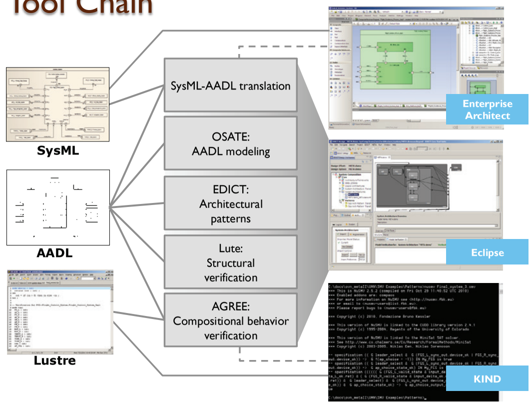
- Suppose we have
  - *Sets of formulas* Γ *and* Q
  - *A well-founded order* $\prec$ *on* Q
  - *Sets* $\Theta_q \subseteq \Delta_q \subseteq Q$, *such that* $r \in \Theta_q$ *implies* $r \prec q$
- Then if for all $q \in Q$
  - $\Gamma \Rightarrow G((Z(H(\Theta_q)) \wedge \Delta_q) \Rightarrow q)$
- Then:
  G(q) for all $q \in Q$
- [Adapted from McMillan]

A concrete example

- Order of data flow through system components is computed by reasoning engine
  - {System inputs} → {FGS_L, FGS_R}
  - {FGS_L, FGS_R} → {AP}
  - {AP} → {System outputs}
- Based on flow, we establish four proof obligations
  - System assumptions → FGS_L assumptions
  - System assumptions → FGS_R assumptions
  - System assumptions + FGS_L guarantees + FGS_R guarantees → AP assumptions
  - System assumptions + {FGS_L, FGS_R, AP} guarantees → System guarantees
- System can handle circular flows, but user has to choose where to break cycle



Tool Chain

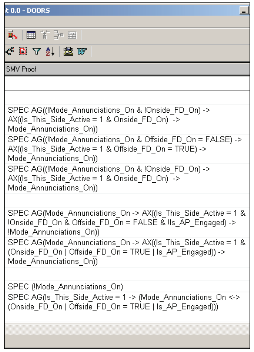


Proving

- Current back-end analysis performed using SMT-based k-induction model checking technique [Hagen and Tinelli: FMCAD 2008]
- Very scalable if properties can be inductively proven
- Unfortunately, Inductive proofs often fail because properties are too weak
- Lots of work on lemma/invariant discovery to strengthen properties
  - Bjesse and Claessen: *SAT-based verification without State Space Traversal*
  - Bradley: *SAT-based Model Checking without Unrolling*
  - Tinelli: *Instantiation-Based Invariant Discovery* [NFM 2011]
- These strengthening methods are not targeted towards our problem
- Only supports analysis of linear models

Scaling

- What do you do when systems and subcomponents have hundreds of requirements?
  - FGS mode logic: 280 requirements
  - DWM: >600 requirements
- Need to create automated slicing techniques for predicates rather than code.
  - Perhaps this will be in the form of counterexample-guided refinement



Assigning blame

- Counterexamples are often hard to understand for big models
- It is much worse (in my experience) for property-based models
- Given a counterexample, can you automatically assign blame to one or more subcomponents?
- Given a "blamed" component, can you automatically open the black box to strengthen the component guarantee?

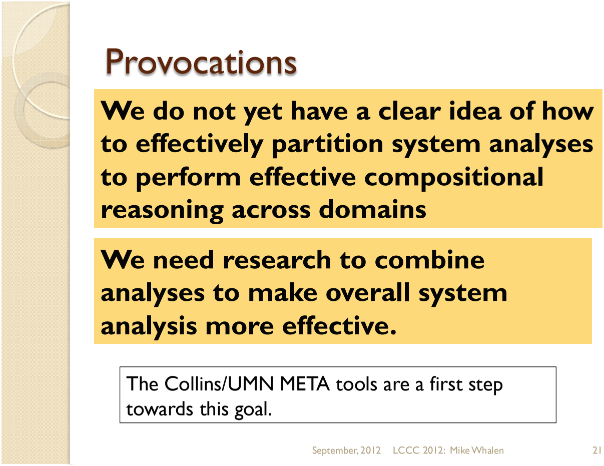
Signal	Step... 0	1	2	3	4	5
AD_L.pitch.val	-0.91	-1.83	-2.74	-3.65	-4.35	-4.39
AD_L.pitch.valid	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
AD_R.pitch.val	0.83	-0.09	-1.00	-1.91	-2.83	-3.74
AD_R.pitch.valid	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
AP.CSA.csa_pitch_delta	0.00	0.13	0.09	0.26	0.74	-4.26
AP.GC_L.cmds.pitch_delta	0.00	-4.91	-4.65	-4.57	-4.74	-4.35
AP.GC_L.mds.active	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE
AP.GC_R.cmds.pitch_delta	0.00	0.83	-4.43	-4.48	4.91	4.83
AP.GC_R.mds.active	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
Assumptions for AP	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Assumptions for FCI	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Assumptions for FGS_L	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Assumptions for FGS_R	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
FGS_L.GC.cmds.pitch_delta	-4.91	-4.65	-4.57	-4.74	-4.35	0.09
FGS_L.GC.mds.active	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
FGS_L.LSO.leader	2	2	3	2	1	3
FGS_L.LSO.valid	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FGS_R.GC.cmds.pitch_delta	0.83	-4.43	-4.48	4.91	4.83	3.91
FGS_R.GC.mds.active	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
FGS_R.LSO.leader	0	0	1	0	1	1
FGS_R.LSO.valid	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
leader_pitch_delta	0.00	0.83	0.83	0.83	0.83	-4.35
System level guarantees	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE

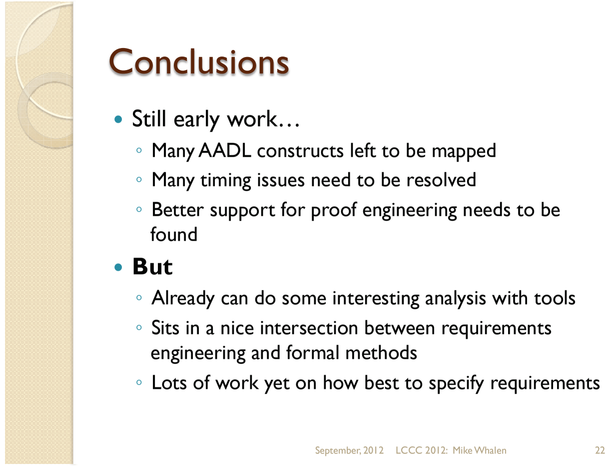
"Argument Engineering"

- Disparate kinds of evidence throughout the system
  - Probabilistic
  - Resource
  - Structural properties of model
  - Behavioral properties of model
- How do we tie these things together?
- Evidence graph, similar to proof graph in PVS
  - Shows evidential obligations that have not been discharged

Dealing with Time

- Current analysis is synchronous
  - It assumes all subcomponents run at the same rate
  - It assumes single-step delay between subcomponents
- This is not how the world works!
  - …unless you use Time-Triggered Architectures or PALS
- Adding more realistic support for time is crucial to accurate analyses
  - Time intervals tend to diverge in hierarchical verification
  - E.g. synchronization.


Provocations
We do not yet have a clear idea of how to effectively partition system analyses to perform effective compositional reasoning across domains
We need research to combine analyses to make overall system analysis more effective.
The Collins/UMN META tools are a first step towards this goal.
September, 2012 LCCC 2012: Mike Whalen 21


Conclusions
• Still early work…
◦ Many AADL constructs left to be mapped
◦ Many timing issues need to be resolved
◦ Better support for proof engineering needs to be found
• But
◦ Already can do some interesting analysis with tools
◦ Sits in a nice intersection between requirements engineering and formal methods
◦ Lots of work yet on how best to specify requirements
September, 2012 LCCC 2012: Mike Whalen 22


Thank you!
Ευχαριστώ
Merci
Grazie
Díky
Gracias
Vielen Dank
Obrigado!
Teşekkürler
شكراً
धन्यवाद
September, 2012 LCCC 2012: Mike Whalen 23

SYSTEMS ENGINEERING: STATUS OF INDUSTRIAL USE, OPPORTUNITIES AND NEEDS

Clas Jacobson, United Technologies Systems & Controls Engineering

This talk will survey aspects of systems engineering and the relevance of equation based modeling in industrial design flows. The particular areas of requirements, architectures, model based design and design flow processes will be addressed. The current landscape in certain industrial areas will be presented along with opportunities and also research needs.

United Technologies

Systems Engineering

Status of Industrial Use, Opportunities and Needs

Clas A. Jacobson
Chief Scientist
Systems & Controls Engineering

LCCC
Lund

September 19, 2012

TEAM

Alberto Sangiovanni Vincentelli, Alberto Ferrari, Mark Myers, John Cassidy, Richard Murray, Andrzej Banaszuk, Sean Meyn, Johan Akesson, Hubertus Tummescheit, Karl Astrom, Manfred Morari, Eelco Scholte, Rich Poisson, Satish Narayanan, Kevin Otto, John Burns, Igor Mezic, Marco Di Natale, Scott Bortoff…

2

AGENDA

System Design

Systems engineering:
(1) requirements,
(2) architecture,
(3) model based design,
(4) (design/development) process

Platform Based Design – design flows (orthogonalize concerns; hierarchy)

Opportunities & progress

System level modeling – positive on reusabality, speed…
Architecture exploration – not fully exploited - but enabled
Requirements – potential to move between formal languages (in progress for embeddded systems)
Model based development – positive on controls - MPC (and optimization), uncertainty (and use for robust design not there yet)
Process – progress on integration of tool chains; level of abstraction change (slightly) with domain (but separate into main product development cycles)

3

DRIVERS

System interactions (“emergent behavior”)

Requirements & acceptance testing (verification)

Safety (critical) (software intensive) systems

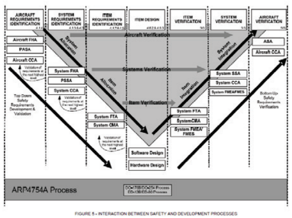
Reusable architectures (modularity)

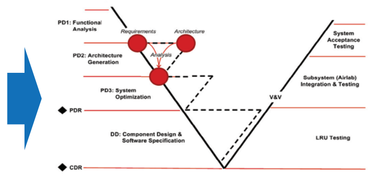
Robustness (risk, lifing)

4

SYSTEMS ENGINEERING (DESIGN)

Process





From process to analysis (model based development)

Bring forward in time the verification testing (SIL => HIL => acceptance)

Orthogonalize requirements (requested behavior) and architecture (delivering services)

5

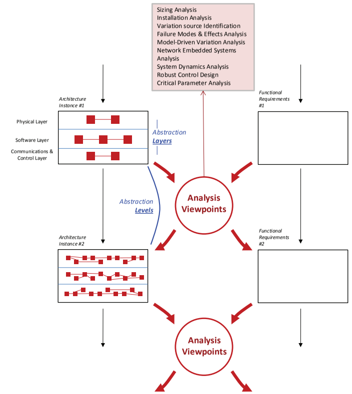
DESIGN PROCESS

Status & Opportunities

Range of analyses (views)

Hierarchy (refinement)

Separation of concerns (requirements, architecture, analysis)



6

SYSTEMS ENGINEERING (DESIGN)

Definition

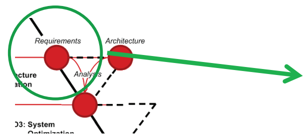
Systems engineering is a methodology for product system level design, optimization and verification that:

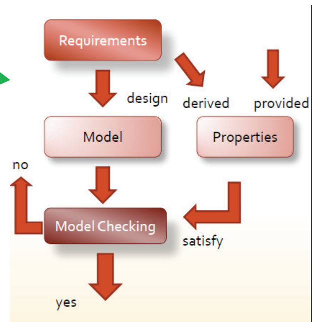
1. Provides guarantees of performance and reliability against customer **requirements** while achieving business cost and time-to-market objectives;
2. Produces modular, extensible **architectures** for products incorporating mechanical components, embedded systems and application software;
3. Exploits **model-based analytical tools and techniques** to determine design choices and ensure robust system performance despite variations caused by product manufacturing, integration with other products and customer operation; and
4. achieves these objectives through the coordinated execution of a prescriptive, repeatable and measurable **process.**

7

REQUIREMENTS

Status & Opportunities





Enabler – formal language (not just equation based language)

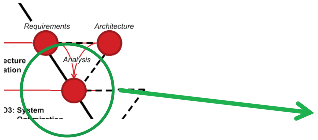
Status – strong for embedded systems; weak for continuous time (non-simulation based verification)

Opportunity – robust design/uncertainty

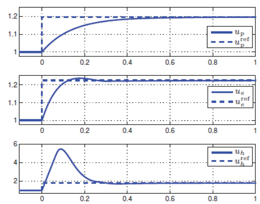
8

MODEL BASED DEVELOPMENT

Status & Opportunities




```
optimization VDP_Opt(objective=cost(finalTime),
    startTime=0,
    finalTime(free=true, initialGuess=1))
  VDP vdp(u(free=true,initialGuess=0.0));
  Real cost (start=0);
equation
  der(cost) = 1;
constraint
  vdp.x1(finalTime) = 0;
  vdp.x2(finalTime) = 0;
  vdp.u >= -1; vdp.u <= 1;
end VDP_Opt;
```



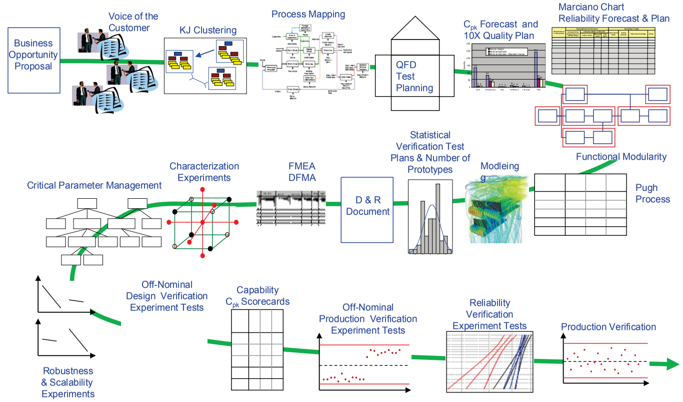
Enabler – equations; interconnection structure

Status of use of equation based language – strong for optimization (MPC; Akesson-Optimica) ; not exploited for robust design; weak for architecture exploration

Opportunity – robust design/uncertainty

9

ROBUST DESIGN



ROBUST DESIGN & UNCERTAINTY

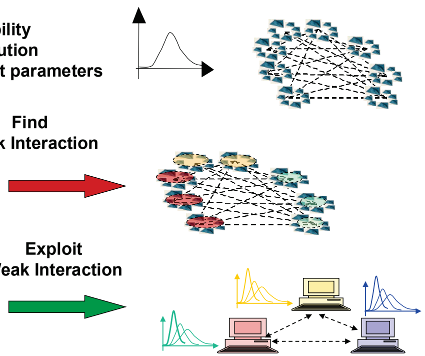
Status & Opportunities: Exploit Structure

Utilize interconnection structure to tear system into strong & weak connections; propagate uncertainty (Meyn-Mathew (and others) DARPA RUM 2008)

Probability Distribution of input parameters

Find Weak Interaction

Exploit Weak Interaction



11

SUMMARY

System Design

Systems engineering :
(1) requirements,
(2) architecture,
(3) model based design,
(4) process

Platform Based Design – design flows (orthogonalize concerns; hierarchy)

Opportunities & progress

System level modeling – positive on reusabality, speed…
Architecture exploration – not fully exploited - but enabled
Requirements – potential to move between formal languages (in progress for embeddded systems)
Model based development – positive on controls - MPC (and optimization), uncertainty (and use for robust design not there yet)
Process – progress on integration of tool chains; level of abstraction change (slightly) with domain (but separate into main product development cycles)

Summary

Big needs on uncertainty/robust design (much wider view of product development);
Opportunity for realizing potential of tool integration (FMI) and with PLM (data management)

12

KEY POINTS

System Design

Systems engineering :
(1) requirements,
(2) architecture,
(3) model based design,
(4) process

Platform Based Design – design flows (orthogonalize concerns; hierarchy)

Opportunities & progress

System level modeling – positive on reusabality, speed…
Architecture exploration – not fully exploited - but enabled
Requirements – potential to move between formal languages (in progress for embeddded systems)
Model based development – MPC (and optimization), uncertainty (not there yet)
Process – integration of tool chains; level of abstraction change (slightly) with domain (but separate into main product development cycles)

Summary

Big needs on uncertainty/robust design;
Opportunity for realizing potential of integration (FMI) with tool chain and PLM

13

THE OPENMODELICA ENVIRONMENT INCLUDING STATIC AND DYNAMIC DEBUGGING OF MODELICA MODELS AND SYSTEMS ENGINEERING/DESIGN VERIFICATION

Peter Fritzson, Linköping University, Department of Computer and Information Science, PELAB – Programming Environment Laboratory

This talk gives an overview of the OpenModelica environment, especially highlighting two aspects:

1) model debugging and 2) systems engineering including design verification against requirements.

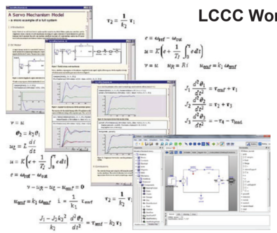
The high abstraction level of equation-based object-oriented languages (EOO) such as Modelica has the drawback that programming and modeling errors are often hard to find. In this paper we present static and dynamic debugging methods for Modelica models and a debugger prototype that addresses several of those problems. The goal is an integrated debugging framework that combines classical debugging techniques with special techniques for equation-based languages partly based on graph visualization and interaction.

The static transformational debugging functionality addresses the problem that model compilers are optimized so heavily that it is hard to tell the origin of an equation during runtime. This work proposes and implements a prototype of a method that is efficient with less than one percent overhead, yet manages to keep track of all the transformations/operations that the compiler performs on the model. Modelica models often contain functions and algorithm sections with algorithmic code. The fraction of algorithmic code is increasing since Modelica, in addition to equation-based modeling, is also used for embedded system control code as well as symbolic model transformations in applications using the MetaModelica language extension.

Our earlier work in debuggers for the algorithmic subset of Modelica used instrumentation-based techniques which are portable but turned out to have too much overhead for large applications. The new debugger is the first Modelica debugger that can operate without run-time information from instrumented code. Instead it communicates with a low-level C-language symbolic debugger to directly extract information from a running executable, set and remove break-points, etc. This is made possible by the new bootstrapped OpenModelica compiler which keeps track of a detailed mapping from the high level Modelica code down to the generated C code compiled to machine code. The debugger is operational, supports both standard Modelica data structures and tree/list data structures, and operates efficiently on large applications such as the OpenModelica compiler with more than 100 000 lines of code. Moreover, an integrated debugging approach is proposed that combines static and dynamic debugging. To our knowledge, this is the first Modelica debugger that supports transformational debugging and algorithmic code debugging.

The second aspect, systems engineering including design verification against requirements, is supported by the OpenModelica ModelicaML profile. This profile implements the vVDR (Virtual Verification of Designs against Requirements) method that enables a model-based design verification against requirements. In the vVDR method there are different kinds of models that are created independently and that will become dependent and need to be related to each other in some concrete verification usage. The aim is to reduce modeling errors and modeling effort by automatically generating composite verification models from their constituting sub-models based on data dependencies that are defined using so-called mediators, which allow expressing data dependencies between models without affecting, i.e., changing, the models themselves.

The OpenModelica Environment including Static and Dynamic Debugging of Modelica Models and Systems Engineering / Design Verification



LCCC Workshop in Lund

September 19-21, 2012

Peter Fritzson

Professor at Linköping University, Sweden
Vice Chairman of Modelica Association
Director of Open Source Modelica Consortium
peter.fritzson@liu.se

Main Contributors, these topics:

Wladimir Schamai, Martin Sjölund, Adrian Pop, Adeel Asghar & rest of OpenModelica team

Linköpings universitet

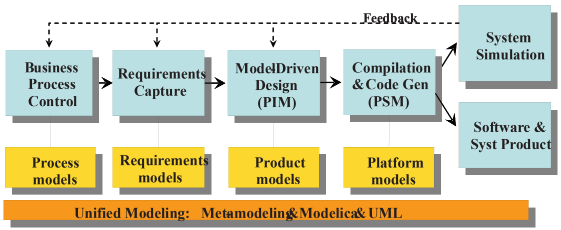
MODELICA

Overview

- Background
- Debugging models
- Dynamic verification of requirements

2

Vision of Integrated Model-Based Development



Vision of unified modeling framework for model-driven product development from platform independent models (PIM) to platform specific models (PSM)

3

Formal Specification of Modelica Static Semantics

- First Structured Operational Semantics (SOS) Modelica subset formal specification
  - First version1998, main parts of Modelica static semantics
  - Primarily Big step semantics / Natural Semantics
  - Generating first version of the OpenModelica compiler
- Generating efficient compiler using RML tool
- 2005 converting rule-based syntax into MetaModelica syntax
- 2011 full integration with standard Modelica
  - Bootstrapping of the OpenModelica compiler

4

Main Language Extensions

- MetaModelica 2005
  - Recursive data structures, lists
  - Pattern matching
  - Failure/exception handling, backtracking
- ParModelica 2011
  - Dataparallel language constructs, multi-core, e.g. mapping to OpenCL
  - Memory hierarchy for data allocation
- Optimization extension 2012
  - Follow same syntax as Optimica in Jmodelica.org
- ModelicaML extension from 2007
  - Integrate UML/SysML graphical language and requirement handling
  - Separate tool, not yet integrated in Modelica and the OpenModelica compiler

5 MODELICA

OpenModelica – An Open Source Environment
Open Source Modelica Consortium, 43 org members Aug 2012

Founded Dec 4, 2007

Open-source community services
- **Website and Support Forum**
- **Version-controlled source base**
- **Bug database**
- **Development courses**
- **www.openmodelica.org**

Interactive Modelica compiler (OMC)
- **Compiles the Modelica Language**
- **Modelica and Python scripting**

Environment for creating models
- **OMShell – scripting commands**
- **OMNotebook – interactive notebook**
- **MDT –Eclipse plug-in**
- **OMEdit graphic Editor**
- **OMOptim optimization tool**
- **ModelicaML UML Profile**

6 MODELICA

Debugging Equation-Based Languages and Background

7 MODELICA

Problems

- Large Gap in Abstraction Level
  from Equations to Executable Code
- Example error message (hard to undestand)

  Error solving nonlinear system 132
  time = 0.002
  residual[0] = 0.288956
  x[0] = 1.105149
  residual[1] = 17.000400
  x[1] = 1.248448
  ...

8 MODELICA

Static vs Dynamic Debugging

- Static Debugging
  - Analyze the model/program at compile-time
  - Explain inconsistencies and errors, trace error dependencies
  - Example: Underconstrained/overconstrained systems of equations
  - Example: errors in symbolic transformations of models
- Dynamic Debugging
  - Find sources of errors at run-time, for a particular execution
  - **Declarative dynamic debugging** – compare the execution with a specfication and semi- automatically find the location of the error
  - **Traditional dynamic debugging** – interactively step through the program, set breakpoints, display and modify data structures, trace, stack inspection
- Goal: Integrated Static and Dynamic Debugging

9

Previous PhD Theses on Dynamic/Static Debugging in Our Group

- *Dynamic*. Nahid Shahmeri(1991). Generalized Algorithmic Debugging
- *Dynamic*. Mariam Kamkar(1993). Interprocedural Dynamic Slicing with Applications to Debugging and Testing
- *Dynamic*. Henrik Nilsson(1998). Declarative Debugging for Lazy Functional Languages
- *Static/Dynamic*. Peter Bunus (June 2004). Debugging Techniques for Equation-Based Languages.
- *Dynamic*. Adrian Pop (June 5, 2008). Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages

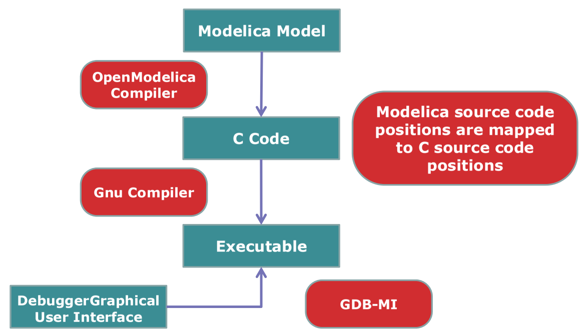
10

Dynamic Debugging

Large Modelica Algorithmic Code Models

11

Tool Architecture and Communication



12

Example Mapping Modelica Postions to C Code

Convert Modelica code to C source code by adding Modelica line number references.

HelloWorld.mo

```
function HelloWorld
  input Real x;
  output Real y;
algorithm
  y := sin(x);
end HelloWorld;
```

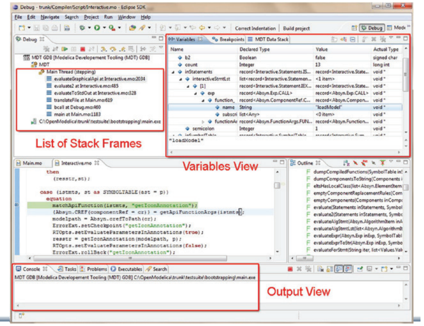
HelloWorld.conv.c

```
#line 29 "HelloWorld.c"
  /* functionBodyRegularFunction: var inits */
#line 30 "HelloWorld.c"
  /* functionBodyRegularFunction: body */
#line 5 "/c/workspace/HelloWorld/HelloWorld.mo"
  tmp2 = sin(_x);
#line 5 "/c/workspace/HelloWorld/HelloWorld.mo"
  _y = tmp2;
#line 35 "HelloWorld.c"
```

13 MODELICA

Debugger Integrated in Eclipse OpenModelica MDT Environment

- Eclipse plugin **MDT (Modelica Development Tooling)** is the integrated development environment
- Debugger is a debug plug-in within MDT



14 MODELICA

Static Debugging

Transformational Debugging of Equation-Based Models

15 MODELICA

Debugging Equation Systems

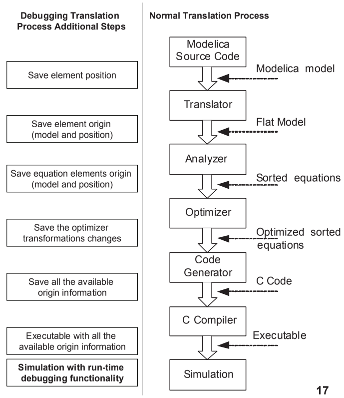
Modelica Compiler Backend

- Complex mathematical transformations
- Hidden to users
- Users want to access this information
- Not intuitive, because
  - No explicit control flow
  - Numerical solvers
  - Linear/Non-linear blocks
  - Optimization
  - Events

16 MODELICA

Translation Phases with Model Debugging

- Include debugging support within the translation process



17

Input to Debugger: Modelica Model

```
class RC // 24 equations and variables
…
equation
 …
 ground1.p.v = 0.0;
 0.0 = resistor1.p.i + resistor1.n.i;
 resistor1.i = resistor1.p.i;
 resistor1.T_heatPort = resistor1.T;
 capacitor1.i = capacitor1.C * der(capacitor1.v);
 capacitor1.v = capacitor1.p.v - capacitor1.n.v;
 0.0 = capacitor1.p.i + capacitor1.n.i;
 capacitor1.i = capacitor1.p.i;
 …
end RC;
```

18

Output from Compiler Frontend:
Sorted ODE or DAE (Differential Algebraic Equations)

```
class RC // 24 equations and variables
…
equation
 …
 ground1.p.v = 0.0;
 0.0 = resistor1.p.i + resistor1.n.i;
 resistor1.i = resistor1.p.i;
 resistor1.T_heatPort = resistor1.T;
 capacitor1.i = capacitor1.C *
   der(capacitor1.v);
 capacitor1.v = capacitor1.p.v –
   capacitor1.n.v;
 0.0 = capacitor1.p.i + capacitor1.n.i;
 capacitor1.i = capacitor1.p.i;
 ...
end RC;
```

```
class RC // 5 equations and variables
…
// 14 alias variables 5 constants
equation
 sinevoltage1.signalSource.y =
sinevoltage1.signalSource.offset + (if time <
sinevoltage1.signalSource.startTime then 0.0
else sinevoltage1.signalSource.amplitude *
sin(6.28318530717959 *
(sinevoltage1.signalSource.freqHz * (time -
sinevoltage1.signalSource.startTime)) +
sinevoltage1.signalSource.phase));
 resistor1.v = capacitor1.v -
sinevoltage1.signalSource.y;
 capacitor1.i = -resistor1.v / resistor1.R_actual;
 resistor1.LossPower = -resistor1.v *
capacitor1.i;
 der(capacitor1.v) = capacitor1.i / capacitor1.C;
end RC;
```

19

Symbolic Transformations

- From source code to flat equations
  - Most of the structure remains
  - Few symbolic manipulations (mostly simplification/evaluation)
- Equation System Optimization
  - Changes structure
  - Strong connected components
  - Variable replacements
  - … and more

20

Tracing Symbolic Transformations

- Simple Idea
  - Store transformations as equation metadata
- Works best for operations on single equations
  - Alias Elimination (a = b)
  - Equation solving ($f_1(a,b) = f_2(a,b)$, solve for a)
- Multiple equations require special handling
  - Gaussian Elimination (linear systems, several equations)
  - ...

21 MODELICA

Tracing Overhead?

- OpenModelica compiler implementation is so fast that tracing is enabled by default
  - 1 extra comparison and/or cons operation per optimization
  - Not noticeable during normal compilation
  - Less than 1% time overhead for tracing
- No real overhead unless you output the trace

22 MODELICA

Substitution Example, Storing the Trace

a = b
c = a + b
d = a - b

c = a + b (subst a=b) =>
c = b + b (simplify) =>
c = 2 * b

d = a - b (subst a=b) =>
d = b - b (simplify) =>
d = 0.0

- The alias relation a=b stored in variable a
- The equations are e.g. stored as (lhs,rhs,list<ops>)

23 MODELICA

Debugging Using the Transformation Trace

- Text output
  - Initial implementation
  - Verify performance and correctness of the trace
- Structured output based on database storage
  - Graphical debugging
  - Cross-referencing equations (dependents/parents)
  - Ability to see why a variable is solved in a particular way
  - Requires a schema
  - Future work/work in progress

24 MODELICA

Trace Example (1)

0 = y + der(x * time * z); **z = 1.0;**

(1) substitution:
y + der(x * (time * z))
=>
y + der(x * (time * 1.0))

(2) simplify:
y + der(x * (time * 1.0))
=>
y + der(x * time)

(3) expand derivative (symbolic diff):
y + der(x * time)
=>
y + (x + der(x) * time)

(4) solve:
0.0 = y + (x + der(x) * time)
=>
der(x) = ((-y) - x) / time

25

Trace Example (2)

differentiation:
d/dtime L ^ 2.0
=>
0.0

differentiation:
d/dtime x ^ 2.0 + y ^ 2.0
=>
2.0 * (der(x) * x + der(y) * y)

Substitution:
2.0 * (der(x) * x + der(y) * y)
=>
2.0 * ($DER.x * x + $DER.y * y)
=>
2.0 * (u * x + $DER.y * y)
=>
2.0 * (u * x + v * y)
=>
2.0 * (u * xloc[1] + v * xloc[0])

26

Readability of Transformation Trace

- Most equations have very **few** transformations on them
- Most of the interesting equations have a few
  - Still rather readable
- Some extra care to handle Modelica variable aliasing

MSL 3.1 MultiBody DoublePendulum

# Ops	Frequency	Comment
0	457	Parameters
1	89	Dummy eq & know var
2	720	Alias vars
3	479	Alias vars
4	124	Alias after simplify
5	25	Alias after simplify
6	99	Alias after simplify
7	55	Scalar eq
8	37	...
9	110	...
10	72	...
11	12	...
12	25	...
13	35	...
14	3	Known constant after many replacements
21	27	World object (3x3 matrix with many occurances of aliased vars)

27

Future Work on Transformational Debugging

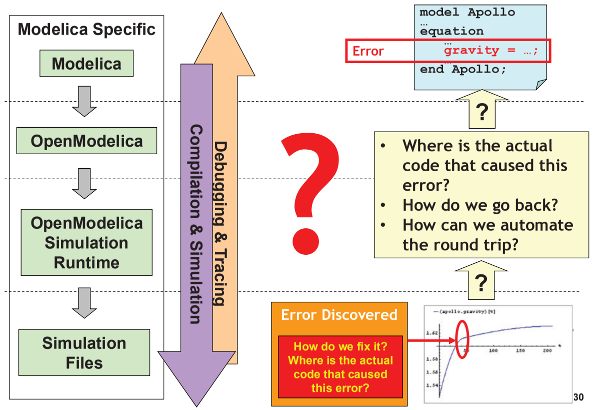
- Structural debug information queries based on a database
- Graphical debugger
- Simulation runtime uses database
- More operations recorded
  - Dead code elimination
  - Control flow and events
  - Forgotten optimization modules

28

Integrated Debugging

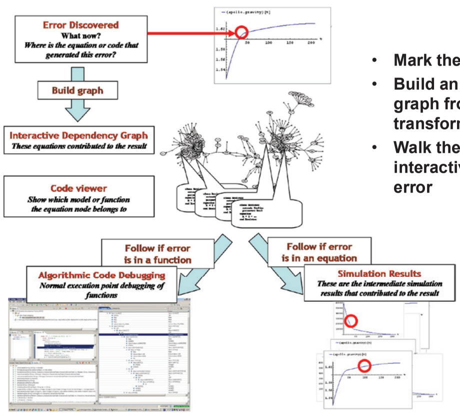
29

Need to Combine Approaches to Help the User



30

Integrated Debugging Approach



- Mark the error
- Build an interactive graph from the transformation trace
- Walk the graph interactively to find the error

31

Debugging Based on User Interaction

- The interactive dependency graph contains two types of edges:
  - *Calculation dependency edges*
  - *Origin edges from traced symbolic transformations*
- The user interacts with the dependency graph in several ways:
  - *Displaying simulation results* through selection of the variables
  - *Classifying a variable* as having wrong values
  - *Classifying an equation as correct*
  - *Building a new dependency graph* based on the new set of variables with wrong values (classified variables) or by modifying the equations or parameter values nodes.
  - *Displaying model code* by following origin edges
  - *Invoking the algorithmic code debugging subsystem*

32

Debugging Summary

- Debugging **equation-based** models present new **challenges**
- **Equation** systems are **transformed** symbolically to a form hard for the user to recognize
- Static **transformational** debugging **explains** the transformations and maintains a mapping between the low level and the high level model
- **Dynamic debugging** helps to **walk** through a model/program and **inspect** data for an execution
- **Goal: integrated static/dynamic debugging approach**

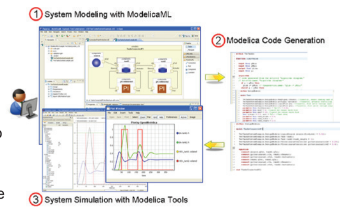
33

Requirements traceability and dynamic model verification

34

Introduction: ModelicaML Background

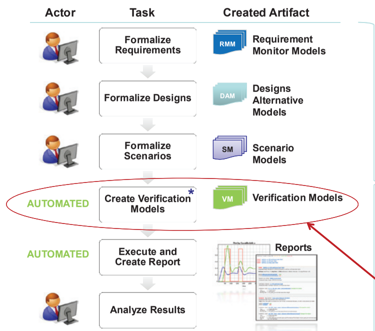
- **ModelicaML Eclipse plug-in Modelica/UML profile** integrates a subset of the UML and the Modelica language in order to leverage standardized graphical notations of UML for system modeling and the simulation power of Modelica
- ModelicaML enables engineers to describe
  - System **requirements**
  - System **design** (structure and behavior)
  - Usage-, test **scenarios**



- **vVDR (Virtual Verification of Designs against Requirements) is a method that enables a model-based design verification against requirements**
- **vVDR is supported in ModelicaML**

35

Introduction: vVDR Method



Goal: Enable on-demand verification of designs against requirements using automated model composition at any time during development.

Focus of this presentation

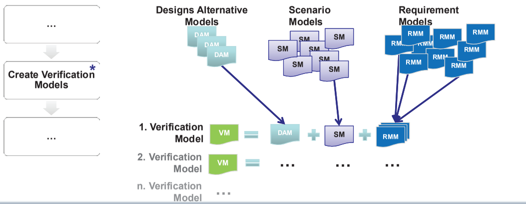
36

Challenge

• We want to verify **different design alternatives** against **sets of requirements** using **different scenarios**. Issues:

1) How to **find valid combinations** of **design alternatives**, **scenarios** and **requirements** in order to enable an automated composition of verification models?

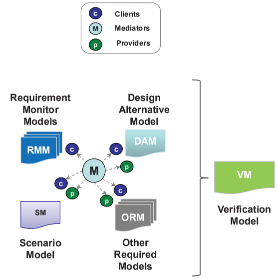
2) Having found a valid combination: How to **bind all components correctly**?



37

Solution Proposal: Value Bindings

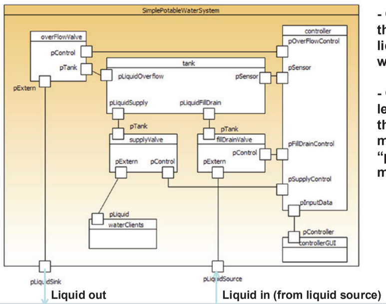
- **Value Binding** enables the automation of verification model composition
- Value Bindings include the definition of:
  - **Client** (component that requires data from other components)
  - **Provider** (component that provides data for other components)
  - **Mediator** (mediates between clients and providers)
- Depending on which mediators and providers are in place we can:
  - Determine which clients can be satisfied
  - **Find valid combinations** and generate verification models
  - Generate **binding** code for client components in verification models



38 Page 38

Example: Design Alternative Model

- Simplified Aircraft Potable Water System



- Overhead tank system that can be filled using a liquid source from bottom with the aircraft on ground.

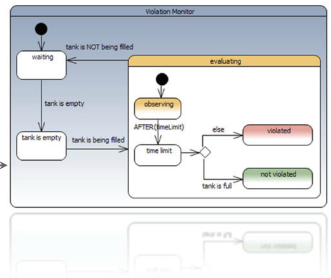
- Controller monitors the level of liquid and controls the valves according to its mode (e.g. "fill", "drain", "pre-selected value fill"-mode).

39

Example: Requirement Monitor Model
"The time to fill an empty tank shall be 300 sec. max."

Clients to get input values from design model providers

001 - Tank filling time
- input Boolean tankIsEmpty = false
- input Boolean tankIsBeingFilled = false
- input Boolean tankIsFull = false
- parameter Real timeLimit = 300
- output Integer status
- Violation Monitor



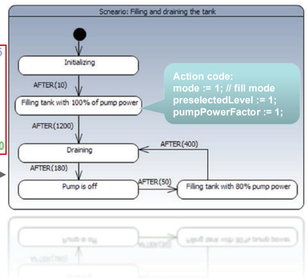
"status" is set by the violation monitor and indicates the following:
0 = not evaluated
1 = evaluated and not violated
2 = violated

40

Example: Scenario Model "*Filling and draining the ta*

Providers for design model clients

- s1-Fill and Drain Tank
  - parameter output Real ambPressure(..) = 101325
  - output Integer mode(..)
  - output Real pumpPowerFactor
  - output Real preselectedLevel
  - output Integer overflowValveStuckAt = 0
  - output Integer fillDrainValveStuckAt = -1
  - output Integer supplyVavleStuckAtPosition = 100
  - input Real tankHeight = 1
  - Sceanrio: Filling and draining the tank

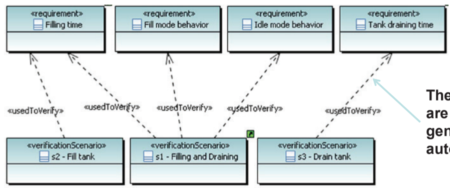


Example scenario: Tank cleaning by filling and draining the tank several times when the aircraft is on ground.

41

Example: Mapping Scenarios to Requirements

- Automatic generation/selection of which scenarios are appropriate to verify which requirements
  - **One scenario** can be used to **verify multiple requirements** (to increase requirements coverage and confidence in verification results)
  - Each requirement should be referenced by at least one scenario



These relations are now generated automatically

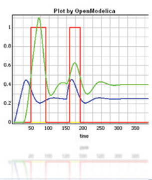
42

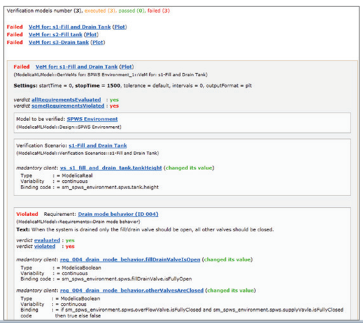
Simulation and Report Generation in ModelicaML

- Verification models are simulated.

The generated Verification Report is a prepared summary of:

- **Configuration, bindings**
- **Violations of requirements**
- **etc.**




Verification models number (3), executed (3), passed (0), failed (3)

Failed VeM for: s1-Fill and Drain Tank (Plot)
Failed VeM for: s2-Fill tank (Plot)
Failed VeM for: s3-Drain tank (Plot)

Failed VeM for: s1-Fill and Drain Tank (Plot)

Settings: startTime = 0, stopTime = 1500, tolerance = default, intervals = 0, outputFormat = plt

verdict allRequirementsEvaluated : yes
verdict someRequirementsViolated : yes

Model to be verified: SPWS Environment

Verification Scenario: s1-Fill and Drain Tank

Violated Requirement: Drain mode behavior (ID 004)

Text: When the system is drained only the fill/drain valve should be open, all other valves should be closed.

verdict evaluated : yes
verdict violated : yes

43

Conclusion

- The ModelicaML Value Bindings approach enables automated model composition, which is used in ModelicaML for **automatic generate verification models**
- Bindings do not modify client or provider models (important when libraries are used)
- Using binding definitions we can find **valid combinations** and **automatically** generate **verification** models
- The generated **verification models** become artifacts that are **created automatically on-demand** and do not need to be maintained

44

Overall Summary

- Goal of integrated model-based development

This talk covers two aspects

- Integrated static/dynamic debugging of models
  - Dynamic debugging of large algorithmic models fully functional
  - Static Equation debugging prototype need to be integrated and scaled up for large models
- Requirements traceability and verification
  - Automated dynamic verification and generation of verification models
  - Need to be integrated in Modelica standard

THE DARK SIDE OF OBJECT-ORIENTED MODELLING: NUMERICAL PROBLEMS, EXISTING SOLUTIONS, FUTURE PERSPECTIVES

Francesco Casella, Politecnico di Milano, Dipartimento di Elettronica e Informazione

Object-Oriented Modelling languages allow to tackle the system-level modelling of engineering system in a modular and convenient way, ideally focusing on clarity and generality of the code rather than on numerical robustness and computational efficiency. Symbolic and numerical methods are now available to generate efficient simulation code from this models; however, there are still some problems (most notably initialization) that are far from being solved in a completely satisfactory way from the end-user perspective. O-O models are now also starting to be used for optimization, which has very different numerical requirements than simulation; numerical and symbolic methods similar in purpose to those already existing for simulation are badly needed in order to automatically turn high-level O-O models into low-level equations that can be optimized reliably and efficiently, thus avoiding the manual tuning of the code for this purpose. The talk will present the current state of the art and point out the needs that haven't been satisfied yet, making the case for new cross-disciplinary research in the field.

The Dark Side of Object-Oriented Modelling: Numerical Problems, Existing Solutions, Future Perspectives

Francesco Casella
(francesco.casella@polimi.it)

Dipartimento di Elettronica e Informazione
Politecnico di Milano

OpenOffice.org

Introduction

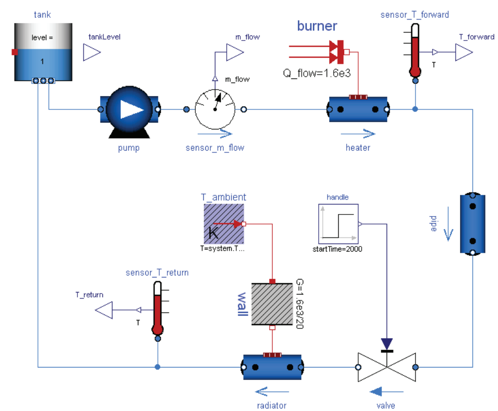
- Equation-Based Object-Oriented Modelling (OOM) approach well established in industry and academia
- Modelica Language emerging
- A-causal approach (write equations, not how they are solved) makes fully modular modelling of multi-domain physical system possible
- Hierarchical modelling, handling of complex systems
- Replaceable models, handling of reconfigurable systems
- Fancy GUIs, model management tools, ...

2

but...

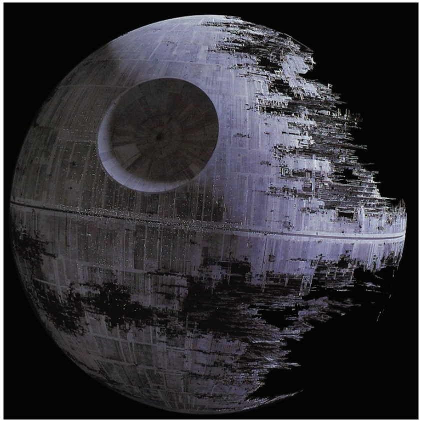
3

A (not so nice) anecdote



4

The dark side of O-O modelling



5

Numerical Errors in OOM: A Taboo Topic?

- *All users* of OOM encounter this kind of problems
- High-level expertise in simulation techinques required for troubleshooting
- Domain experts use the model!
- These errors are a blocker (no result obtained at all unti. solved)
- They can scare off people from OOM technology

6

Numerical Errors in OOM: A Taboo Topic?

- *All users* of OOM encounter this kind of problems
- High-level expertise in simulation techinques required for troubleshooting
- Domain experts use the model!
- These errors are a blocker (no result obtained at all unti. solved)
- They can scare off people from OOM technology

- People cope somehow, eventually
- Fact never mentioned in technical or scientific literature
(something not worth mentioning, or even to be somwhat ashamed of?)

- Little progress in solving these problems in a sound and systematic way
- Error handling and debugging techniques much less developed
than simulation techniques

7

Numerical Problems in OOM Simulation

8

Numerical Problems in OOM Simulation

OO Model (Modelica)

Dynamic equations
System behaviour
$F(x, \dot{x}, v, p, t) = 0$

Initial equations
Initial states
& unknown parameters
$F_i(x, \dot{x}, v, p) = 0$

9

Simulation Problem

- At each time step,
  solve $F(x, \dot{x}, v, p, t) = 0$ given x, p, t
- Behaviour equations
  - well-tested model library
  - physically meaningful connection equations
- Known states problem broken into many small subproblems
- Advanced numerical/symbolic techniques used for efficient solution
- Modelling errors wrong behaviour wrong trajectories

10

Simulation Problem

- At each time step,
  solve $F(x, \dot{x}, v, p, t) = 0$ given x, p, t
- Behaviour equations
  - well-tested model library
  - physically meaningful connection equations
- Known states problem broken into many small subproblems
- Advanced numerical/symbolic techniques used for efficient solution
- Modelling errors wrong behaviour wrong trajectories

Errors can be investigated
by domain-expert modeller

11

Initialization problem

- At time $t = 0$, solve $F(x, \dot{x}, v, p, 0) = 0$
  $F_i(x, \dot{x}, v, p) = 0$
- Typically large systems of highly non-linear equations
- In case of solver errors: no trajectories available for inspection

12

Initialization problem

- At time $t = 0$, solve $F(x, \dot{x}, v, p, 0) = 0$
  $F_i(x, \dot{x}, v, p) = 0$
- Typically large systems of highly non-linear equations
- In case of solver errors: no trajectories available for inspection

Error scenarios

(1) Well-posed problem, solver fails
(2) Underconstrained/overconstrained problem
(3) System-level singular problem, solver fails
(4) Structurally well-posed problem, wrong parameters (no solution)

13

(1) Well-Posed Problem, Simulation Fails

- Nonlinear solver needs initial guess for all iteration variables
- Providing close enough values often unfeasible or very tedious
- Can't tell whether the guess values are wrong, or the problem has no solution at all

14

(1) Well-Posed Problem, Simulation Fails

- Nonlinear solver needs initial guess for all iteration variables
- Providing close enough values often unfeasible or very tedious
- Can't tell whether the guess values are wrong, or the problem has no solution at all

➡ *Homotopy-based initialization*

➡ *High-level debugging*

15

Homotopy-Based Initialization

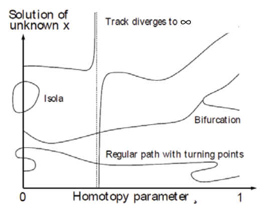
- Define a simplified problem which is easier to solve

$$F_s(x) = 0$$

- Continuously transform into the actual problem

$$(1 - \lambda)\, F_s(x) + \lambda\, F_a(x) = 0$$

- New Modelica operator `homotopy(actual, simplified)` introduced in 2011
- Beware of singularities!



16

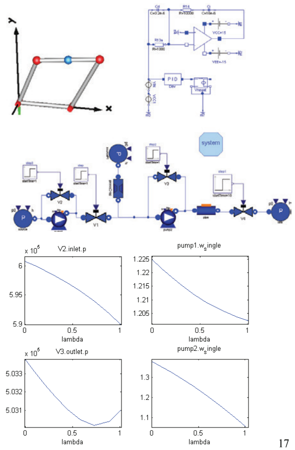
Homotopy-Based initialization – First Results

Modelica Conference 2011
(Casella, Sielemann et al.)

- Multibody systems with multiple configurations
- Analog electronic circuits
- Hydraulic networks
- Calibration of air-conditiong systems
- Large power plants

Modelica Conference 2012
(Sielemann)

- Probability one homotopy



17

Homotopy-Based initialization – First Results

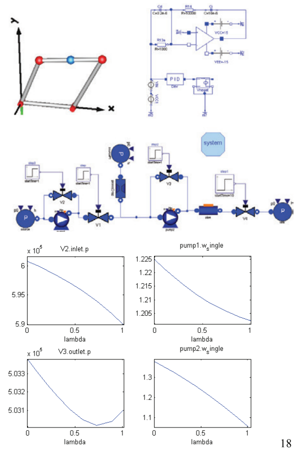
Modelica Conference 2011
(Casella, Sielemann et al.)

- Multibody systems with multiple configurations
- Analog electronic circuits
- Hydraulic networks
- Calibration of air-conditiong systems
- Large power plants

Modelica Conference 2012
(Sielemann)

- Probability one homotopy

Promising approach
more operational experience
needed before getting mature



18

High-Level Debugging: the Open Modelica Debugger

- Idea #1: keep track of all model transformations in the generated code
- Idea #2: explore this information graphically, by traversing graphs

19

High-Level Debugging: the Open Modelica Debugger

- Idea #1: keep track of all model transformations in the generated code
- Idea #2: explore this information graphically, by traversing graphs

- The solver fails when trying to solve an equation in the BLT
  or
  A result is found, clearly wrong from a physical perspective

20

High-Level Debugging: the Open Modelica Debugger

- Idea #1: keep track of all model transformations in the generated code
- Idea #2: explore this information graphically, by traversing graphs

- The solver fails when trying to solve an equation in the BLT
  or
  A result is found, clearly wrong from a physical perspective

Code Dependency Graph

- where does the equation come from in the original model?
- which transformations have been applied?

Variable Dependency Graph

- from which other variables it depends?
- what values they have?

- see paper by Pop et al., Modelica Conference 2012

21

(2) Underconstrained/Overconstrained problem

- Initial conditions are usually set at the single component level
- Initialization problem is a system-level problem
  (exp. with control systems and steady-state conditions!)
- Additional initial equations added determine unknown parameters (trimming problem)
- Initial equations made redundant in case of index reduction

22

(2) Underconstrained/Overconstrained problem

- Initial conditions are usually set at the single component level
- Initialization problem is a system-level problem
  (exp. with control systems and steady-state conditions!)
- Additional initial equations added determine unknown parameters (trimming problem)
- Initial equations made redundant in case of index reduction

- Often init problem turns out to be underconstrained or overconstrained
- Underconstrained tool adds extra initial equations based on heuristics
  - might not reflect the end user's intention
- Overconstrained the tool asks to remove initial equations
  - not clear which one is "the right one"
  - not clear how to do it on structured model via GUI

No satisfactory solution available yet

23

(3) System-level singular problem, solver fails

- The initialization problem can be square but singular for system-level structural reasons
- Examples:
  - Closed thermohydraulic systems with steady-state initial equations
  - Closed hydraulic systems with incompressible fluid
  - Electrical circuits without ground connection
- The Jacobian of the initialization problem equations is singular, no unique solution, solver fails

24

(3) System-level singular problem, solver fails

- The initialization problem can be square but singular for system-level structural reasons
- Examples:
  - Closed thermohydraulic systems with steady-state initial equations
  - Closed hydraulic systems with incompressible fluid
  - Electrical circuits without ground connection
- The Jacobian of the initialization problem equations is singular, no unique solution, solver fails

- Analysis of the nullspace of the Jacobian can reveal the subset of equations causing the singularity
- Meaningful annotations on key equations which are always part of the singular system can be used for high-level diagnostics to the end user
- Proposed by Casella, Modelica Conference 2012, not yet implemented

25

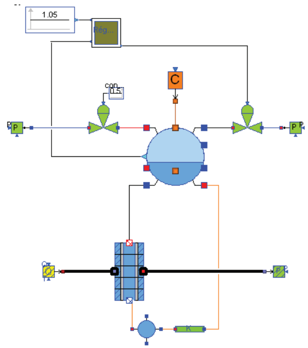
(4) Structurally well-posed problem, wrong parameters

- The strange case of the steam generator
- Steam valve Cv too small
- No steady state solution exists!

- Difficult to determine "the" wrong parameter
  - steam valve too small?
  - too much flue gas flow?
  - flue gas too hot?

- Traditional troubleshooting strategy: divide & conquer

- New ideas?



26

Numerical Problems in OOM Optimization

27

Dynamic optimization of OO Models

- OO models can be used for dynamic optimization (optimal control) of physical systems
- The OO modelling language is used to write the model DAEs which are the dynamic constraint of the optimization problem
- Extension of Modelica: Optimica (J. Åkesson PhD work)

```
optimization DIMinTime (
    objective=finalTime,
    startTime=0,
    finalTime(free=true,initialGuess=1))
  Real x(start=0,fixed=true);
  Real v(start=0,fixed=true);
  input Real u;
equation
  der(x)=v;
  der(v)=u;
constraint
  x(finalTime)=1;
  v(finalTime)=0;
  v<=0.5;
  u>=-1;
  u<=1;
end DIMinTime;
```

28

Dynamic optimization of OO Models

- OO models can be used for dynamic optimization (optimal control) of physical systems
- The OO modelling language is used to write the model DAEs which are the dynamic constraint of the optimization problem
- Extension of Modelica: Optimica (J. Åkesson PhD work)

```
optimization DIMinTime (
     objective=finalTime,
     startTime=0,
     finalTime(free=true,initialGuess=1))
  Real x(start=0,fixed=true);
  Real v(start=0,fixed=true);
  input Real u;
equation
  der(x)=v;
  der(v)=u;
constraint
  x(finalTime)=1;
  v(finalTime)=0;
  v<=0.5;
  u>=-1;
  u<=1;
end DIMinTime;
```

Model DAEs are constraints

Very large non-convex optimization problems

Convergence of solver is a major issue (more than in simulation problems!)

29

Symbolic Manipulation of Equations

for Simulation

- DAEs solved for state derivatives, then time integration
- alias elimination
- BLT ordering
- index reduction / dummy der
- symbolic solution of implicit equations
- tearing of implicit systems
- common subexpression elimination

- scaling of variables (via `nominal` attribute)
- state selection
- inverse annotations
- simplified expressions and homotopy for initialization

for Optimization

- DAEs used as constraints
- alias elimination

- scaling of variables (via `nominal` attribute)
- bounds on variables (via `min`/`max` attributes)

... ?

30

Consequences on Modelling

for Simulation

- The modeller can focus on clarity, readability, generality
- The modeller can (almost) ignore how equations will be solved
- Efficient and robust simulation code automatically generated regardless how the equations are written

for Optimization

- The modeller must focus on solvability and convergence issues when writing the equations
- Mathematically equivalent models

completely different optimization problems

completely different convergence properties

~~*Declarative OOM*~~

31

Example 1

```
optimization Example1A(
  objective = a1/b1*(x1 - x10)^2 +
              a2/b2*(x2 - x20)^2 +
              a3/b3*(x3 - x30)^2,
...
end Example1A;
```

```
optimization Example1B(
  objective = f1 + f2 + f3,
...
equation
  f1 = a1/b1*(x1 - x10)^2;
  f2 = a2/b2*(x2 - x20)^2;
  f3 = a3/b3*(x3 - x30)^2;
...
end Example1B;
```

32

Example 2

```
optimization Example2A(
  parameter Real PR = 10;
...
equation
  p_in/p_out = PR "Turbine pressure ratio";
...
end Example1A;


optimization Example2A(
  parameter Real PR = 10;
...
equation
  p_in = PR*p_out "Turbine pressure ratio";
...
end Example1A;
```

33

Research Goals

Investigate structural properties of model DAEs that help NLP solver convergence

Devise structural analysis and manipulation techniques that can transform DAEs into "more favourable" equations

Talk to NLP solver developers so they understand the structure of "our" problems in depth and help us solve them better

Bring the field of OOM Optimization from adolescence (or childhood?) into adulthood

34

Conclusions

- Object-Oriented Modelling concepts are getting more and more widely used in system engineering and control
- OOM has a huge potential to help closing the gap between theory and practice in optimal control
- Use of OOM for simulation is now fairly mature, but still has some dark areas related to initialization problems that need to be addressed for wider acceptance and use
- Use of OOM for optimization still quite young, comparably little effort spent in automatic (or assisted) streamling of model equations
- Some steps in the right direction have already been made in recent years
- More research is needed to
  - improve the numerical robustness and mitigate convergence problems in tools
  - develop effective and easy-to-use debugging tools for domain experts
  - possibly by embedding expert a-priori knowledge in the modelling code and exploiting it cleverly

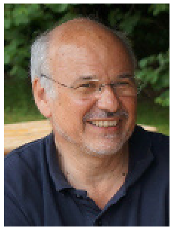
35

Thank you for you kind attention!

36

BRIDGING BETWEEN DIFFERENT MODELING FORMALISMS – RESULTS FROM THE MULTIFORM PROJECT

Sebastian Engell, Process Dynamics and Operations Group, Department of Biochemical and Chemical Engineering, TU Dortmund



The European project MULTIFORM that ended in May, 2012, had the goal to provide interoperability between different modeling, simulation, optimization and analysis tools for hybrid dynamic systems (systems with continuous dynamics modeled by DAE and switching logic). For this purpose, the Compositional Interchange Format (CIF) that was developed by a group at TU Eindhoven was employed. The CIF is a formally defined, automata-based modeling language for hybrid systems. Transformations between the CIF and Modelica, gPROMS, Uppaal (a tool for the analysis of timed-automata models), SpaceEx (a tool for the reachability analysis of hybrid automata) and other tools for logic controller design and synthesis and control software verification have been developed. Tool chains for the development and simulation of logic controllers interacting with continuous dynamic systems were demonstrated. The model transformations and results handling are supported by the MULTIFORM Design Framework in which supports (partly) model-based design processes. We illustrate the idea and the results of MULTIFORM by the design of a miniature pipeless plant for education and demonstration purposes and discuss the experience gained.

multiform

Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

Bridging between different modeling formalisms - results from the MULTIFORM project

Martin Hüfner, Christian Sonntag, Sebastian Engell

Process Dynamics and Operations Group
Department of Biochemical and Chemical Engineering
TU Dortmund
Germany

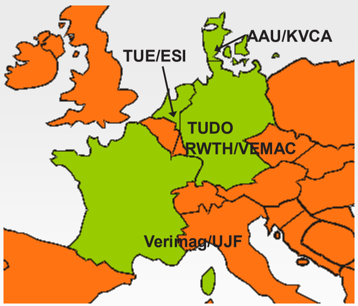
System Design meets Equation-based Languages, Sept. 19, 2012, Lund

Outline

- The MULTIFORM project
- Design flow example
- Tool developments
- Model exchange and model transformations
- Lessons learned

multiform 2

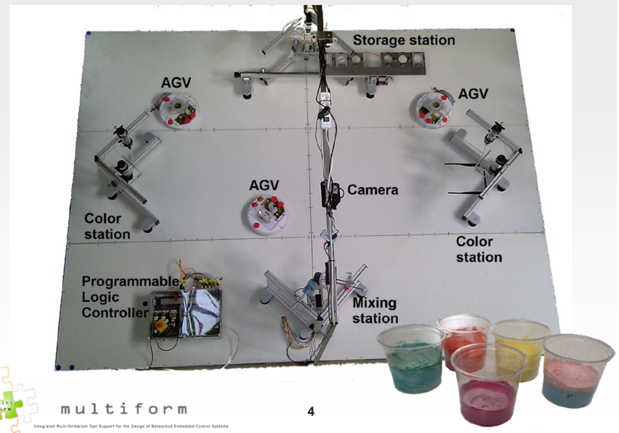
MULTIFORM: EU ICT STREP 9/2008 – 5/2012



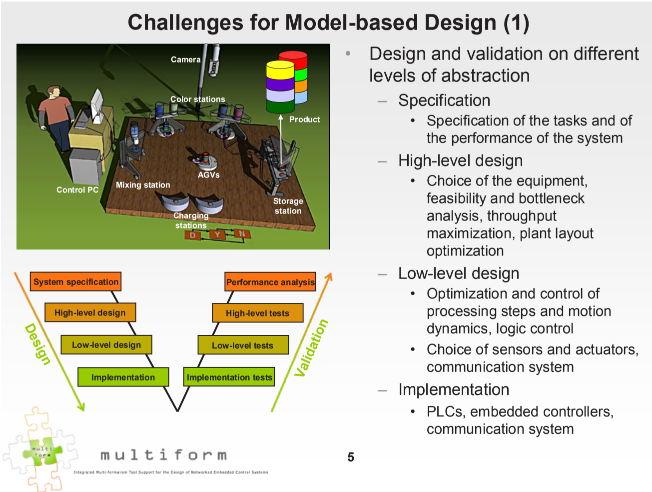
- **VEMAC**
  - Aachen, Germany
    Michael Reke
- **KVCA**
  - "Danish Cooling Cluster"
    Jens Andersen
  - Closely working with DANFOSS
- **TUDO (Coordinator)**
  - TU Dortmund, Germany
    Sebastian Engell
- **TUE**
  - TU Eindhoven, Netherlands
    Koos Rooda, Bert van Beek, Jos Baeten
- **Verimag/ UJF**
  - Universite Joseph Fourier, Grenoble, France
    Goran Frehse, Oded Maler
- **RWTH**
  - RWTH Aachen, Germany
    Stefan Kowalewski
- **AAU**
  - Aalborg Universitet, Denmark
    Kim Larsen, Brian Nielsen
- **ESI**
  - Stichting Embedded Systems Institute
    Ed Brinksma, Boudewijn Haverkort

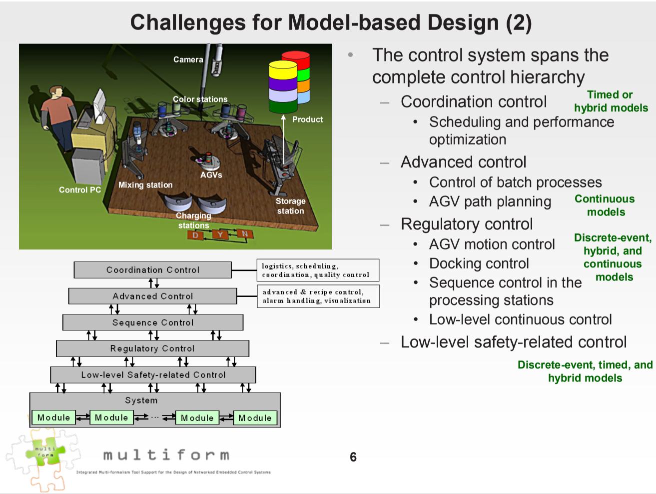
multiform 3

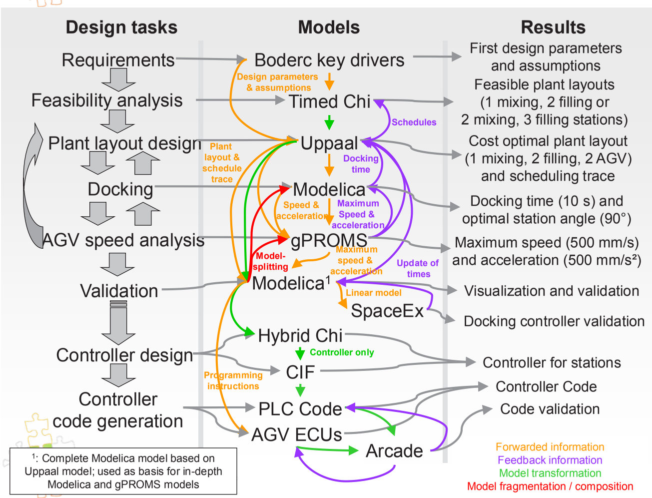
Example: Design of a Pipeless Plant

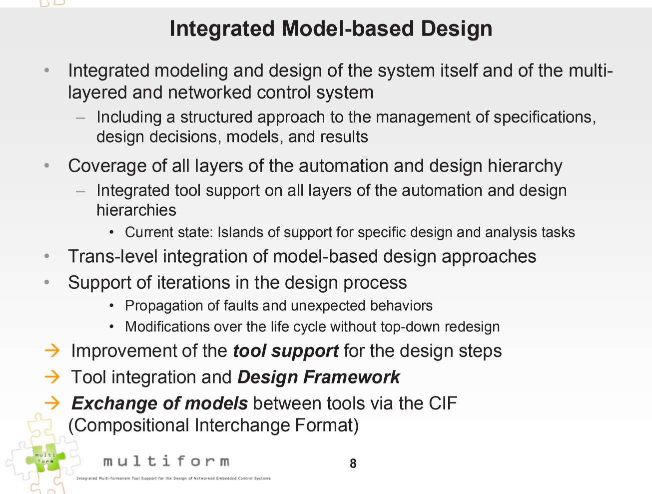


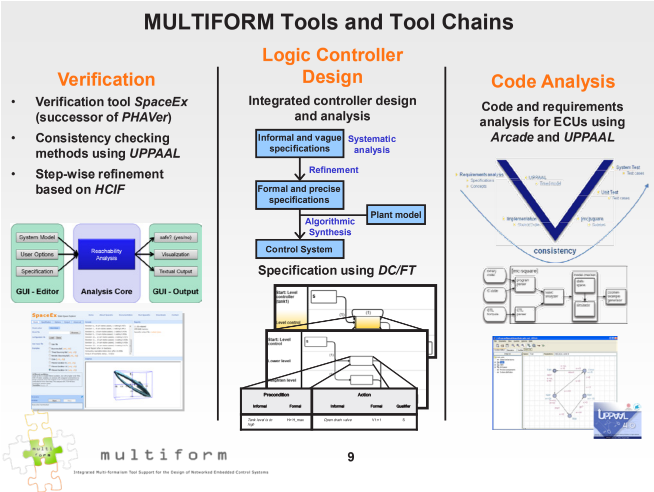
multiform 4

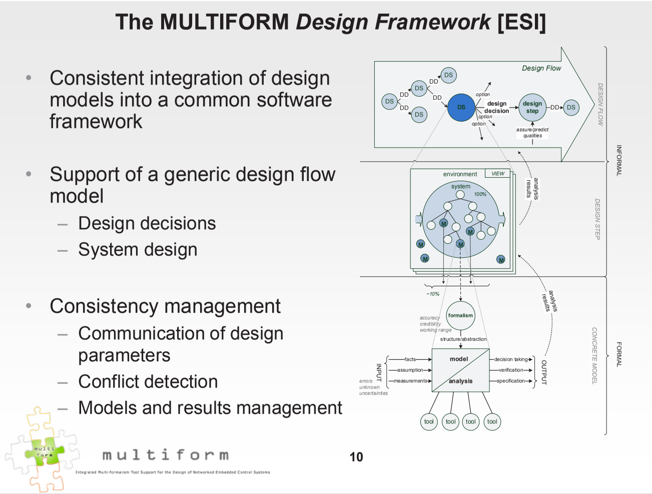

Challenges for Model-based Design (1)
Camera
Color stations
Product
Control PC
Mixing station
AGVs
Storage station
Charging stations
System specification
High-level design
Low-level design
Implementation
Performance analysis
High-level tests
Low-level tests
Implementation tests
Design
Validation
Design and validation on different levels of abstraction
Specification
Specification of the tasks and of the performance of the system
High-level design
Choice of the equipment, feasibility and bottleneck analysis, throughput maximization, plant layout optimization
Low-level design
Optimization and control of processing steps and motion dynamics, logic control
Choice of sensors and actuators, communication system
Implementation
PLCs, embedded controllers, communication system
multiform
5

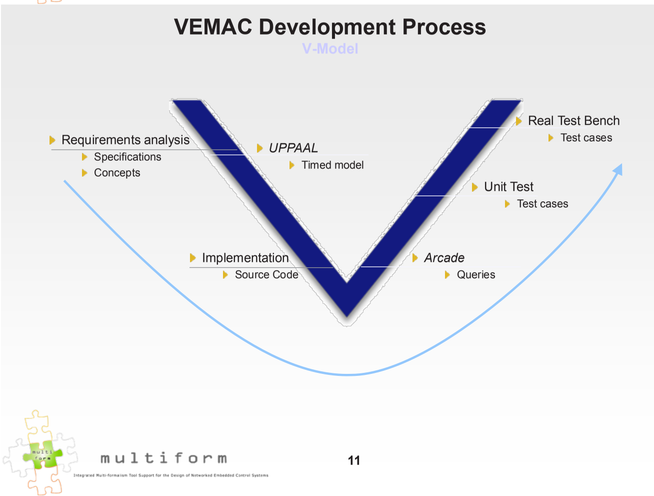

Challenges for Model-based Design (2)
Camera
Color stations
Product
Control PC
Mixing station
AGVs
Storage station
Charging stations
Coordination Control
Advanced Control
Sequence Control
Regulatory Control
Low-level Safety-related Control
System
Module
logistics, scheduling, coordination, quality control
advanced & recipe control, alarm handling, visualization
The control system spans the complete control hierarchy
Coordination control
Timed or hybrid models
Scheduling and performance optimization
Advanced control
Control of batch processes
AGV path planning
Continuous models
Regulatory control
AGV motion control
Docking control
Sequence control in the processing stations
Low-level continuous control
Discrete-event, hybrid, and continuous models
Low-level safety-related control
Discrete-event, timed, and hybrid models
multiform
6

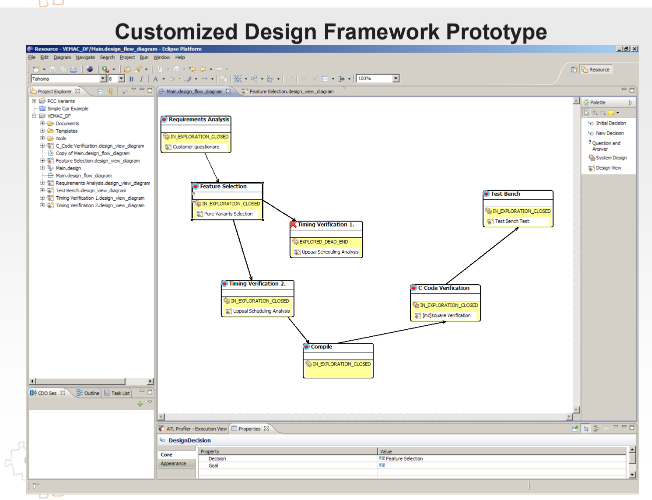

Design tasks
Models
Results
Requirements
Feasibility analysis
Plant layout design
Docking
AGV speed analysis
Validation
Controller design
Controller code generation
Boderc key drivers
Design parameters & assumptions
Timed Chi
Schedules
Uppaal
Plant layout & schedule trace
Docking time
Modelica
Speed & acceleration
Maximum Speed & acceleration
gPROMS
Model-splitting
Maximum speed & acceleration
Update of times
Modelica¹
Linear model
SpaceEx
Hybrid Chi
Controller only
CIF
Programming instructions
PLC Code
AGV ECUs
Arcade
First design parameters and assumptions
Feasible plant layouts (1 mixing, 2 filling or 2 mixing, 3 filling stations)
Cost optimal plant layout (1 mixing, 2 filling, 2 AGV) and scheduling trace
Docking time (10 s) and optimal station angle (90°)
Maximum speed (500 mm/s) and acceleration (500 mm/s²)
Visualization and validation
Docking controller validation
Controller for stations
Controller Code
Code validation
¹: Complete Modelica model based on Uppaal model; used as basis for in-depth Modelica and gPROMS models
Forwarded information
Feedback information
Model transformation
Model fragmentation / composition


Integrated Model-based Design
Integrated modeling and design of the system itself and of the multi-layered and networked control system
Including a structured approach to the management of specifications, design decisions, models, and results
Coverage of all layers of the automation and design hierarchy
Integrated tool support on all layers of the automation and design hierarchies
Current state: Islands of support for specific design and analysis tasks
Trans-level integration of model-based design approaches
Support of iterations in the design process
Propagation of faults and unexpected behaviors
Modifications over the life cycle without top-down redesign
Improvement of the tool support for the design steps
Tool integration and Design Framework
Exchange of models between tools via the CIF (Compositional Interchange Format)
multiform
8


MULTIFORM Tools and Tool Chains
Verification
Verification tool SpaceEx (successor of PHAVer)
Consistency checking methods using UPPAAL
Step-wise refinement based on HCIF
System Model
User Options
Specification
GUI - Editor
Reachability Analysis
Analysis Core
safe? (yes/no)
Visualization
Textual Output
GUI - Output
Logic Controller Design
Integrated controller design and analysis
Informal and vague specifications
Systematic analysis
Refinement
Formal and precise specifications
Plant model
Algorithmic Synthesis
Control System
Specification using DC/FT
Code Analysis
Code and requirements analysis for ECUs using Arcade and UPPAAL
consistency
multiform
9


The MULTIFORM Design Framework [ESI]
Consistent integration of design models into a common software framework
Support of a generic design flow model
– Design decisions
– System design
Consistency management
– Communication of design parameters
– Conflict detection
– Models and results management
Design Flow
design decision
design step
environment
system
formalism
model
analysis
INPUT
OUTPUT
tool
INFORMAL
FORMAL
DESIGN FLOW
DESIGN STEP
CONCRETE MODEL
multiform
10


VEMAC Development Process
V-Model
Requirements analysis
Specifications
Concepts
UPPAAL
Timed model
Implementation
Source Code
Arcade
Queries
Unit Test
Test cases
Real Test Bench
Test cases
multiform
11

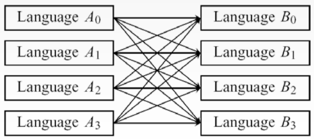

Customized Design Framework Prototype
Requirements Analysis
Feature Selection
Timing Verification 1.
Timing Verification 2.
Compile
C-Code Verification
Test Bench

Model Exchange Using the Compositional Interchange Format (CIF)

- Incompatibility of tools is one of the major obstacles for a broader acceptance of model-based design in industry

→ Achieve inter-operability by (algorithmic) model transformations

- One possibility: Bi-lateral transformations
  - Problems
    - Many transformations may be needed
    - The developer of a transformation must be familiar with many different formalisms



multiform 13

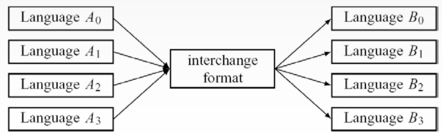
Model Exchange with the Compositional Interchange Format (CIF)

- Incompatibility of tools is one of the major obstacles for a broader acceptance of model-based design in industry

→ Achieve inter-operability by (algorithmic) model transformations

- One possibility: Bi-lateral transformations
- Interchange format
  - Generic and sufficiently rich modelling formalism
  - Only transformations from/to the interchange format are necessary

  → Reduction of the implementation effort



multiform 14

The Compositional Interchange Format (CIF) [Bert van Beek et al., TUE]

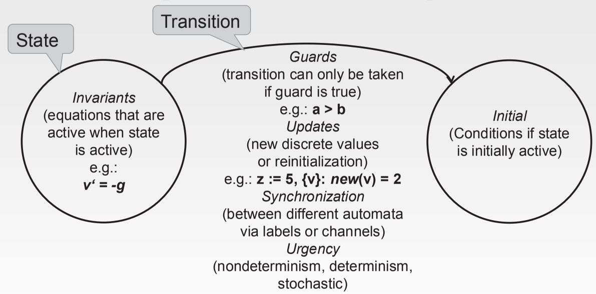
- Purposes
  - Establish inter-operability of a wide range of tools
  - Provide a generic formalism for general hybrid systems
- Major features
  - Formal and compositional semantics
    - Independent of implementation aspects
    - Mathematical correctness proofs of translations

    → Property-preserving model transformations possible
  - Fully implicit DAE dynamics (possibly discontinuous)
  - Hierarchy and re-usability
    - Parallel composition with different communication concepts
  - Model component interaction
    - Point to point communication, multi-component synchronization, broadcast communication, shared variables
  - Different urgency concepts

http://devel.se.wtb.tue.nl/trac/cif/

multiform 15

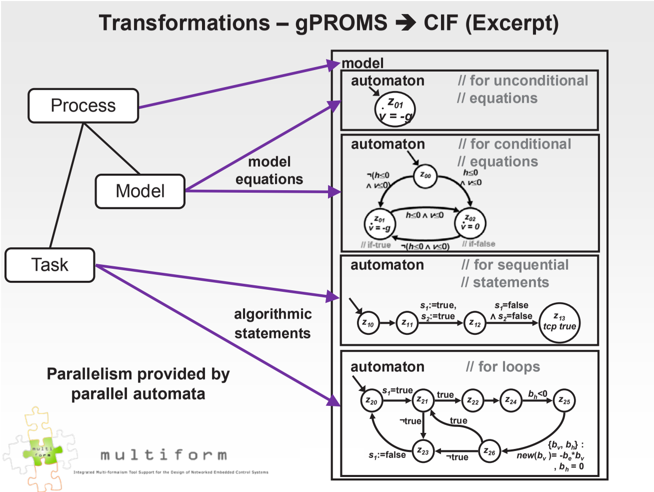
The Compositional Interchange Format (CIF) [Bert van Beek et al., TUE]

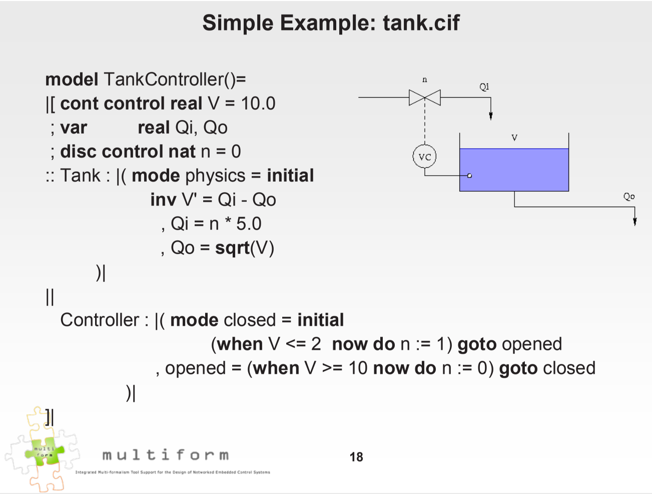


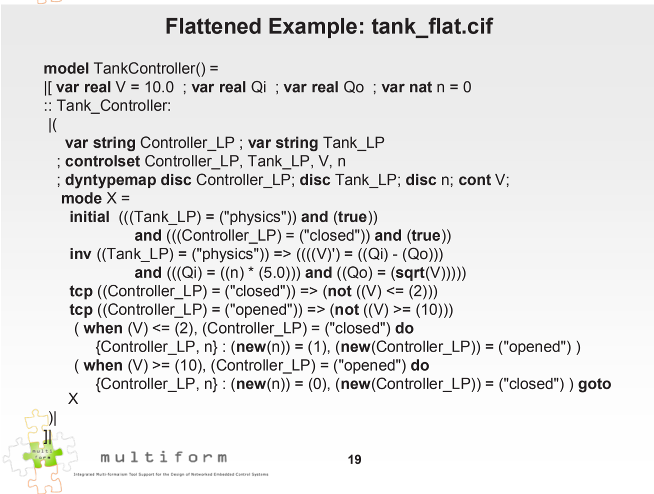
Formal definition by *Structural Operational Semantics* (SOS) rules, e.g.:

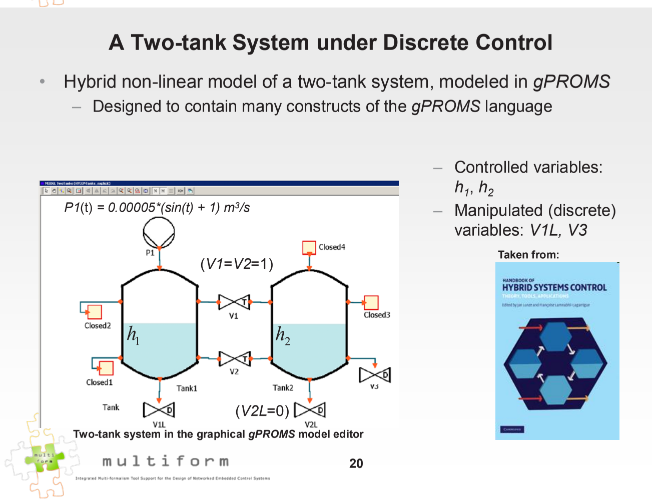
$$\frac{(\alpha_0, \sigma) \xrightarrow{a,\mathrm{true},X} (\alpha_0', \sigma'),\ (\alpha_1, \sigma) \xrightarrow{a,\mathrm{true},X} (\alpha_1', \sigma')}{(\alpha_0 \parallel \alpha_1, \sigma) \xrightarrow{a,\mathrm{true},X} (\alpha_0' \parallel \alpha_1', \sigma')}$$

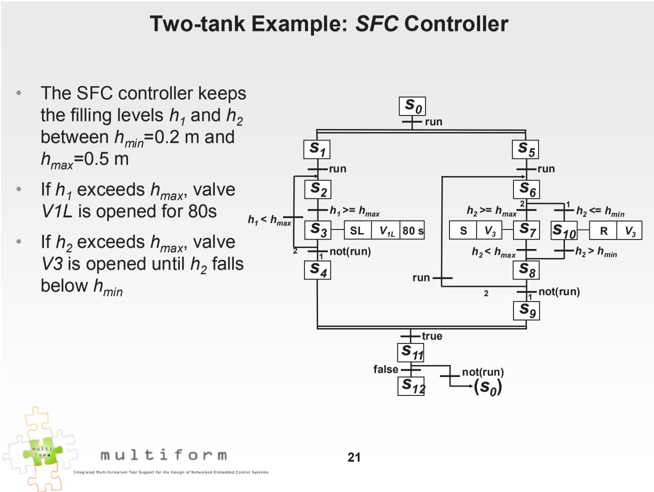
multiform 16

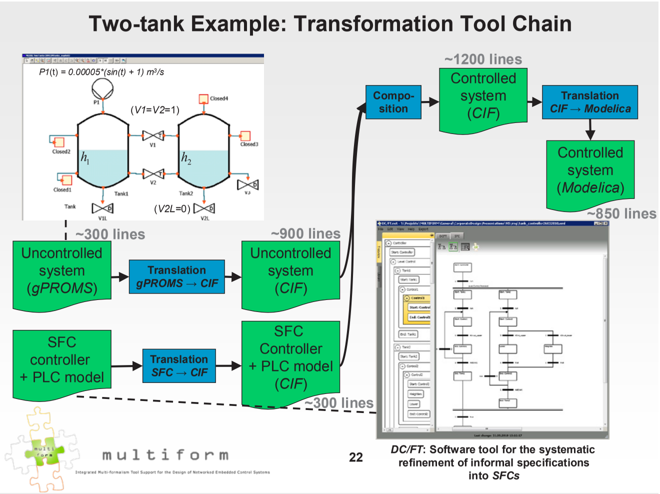

Transformations – gPROMS ➔ CIF (Excerpt)
Process
Model
Task
model equations
algorithmic statements
Parallelism provided by parallel automata
model
automaton // for unconditional // equations
automaton // for conditional // equations
automaton // for sequential // statements
automaton // for loops
multiform

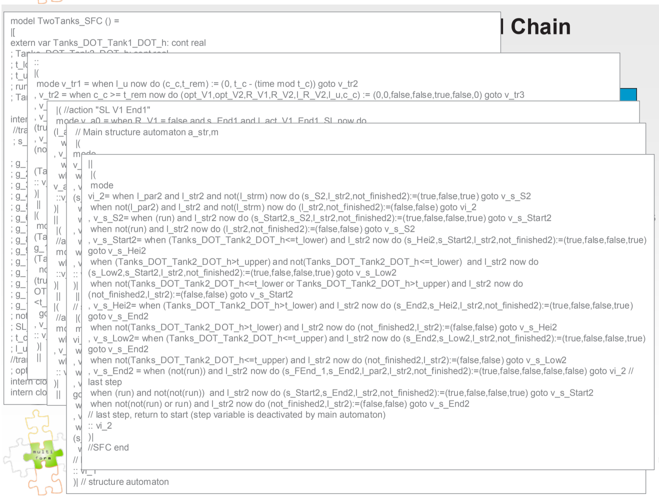

Simple Example: tank.cif
model TankController()=
|[cont control real V = 10.0
; var real Qi, Qo
; disc control nat n = 0
:: Tank : |(mode physics = initial
inv V' = Qi - Qo
, Qi = n * 5.0
, Qo = sqrt(V)
)|
||
Controller : |(mode closed = initial
(when V <= 2 now do n := 1) goto opened
, opened = (when V >= 10 now do n := 0) goto closed
)|
]|
n
Qi
V
VC
Qo
multiform
18

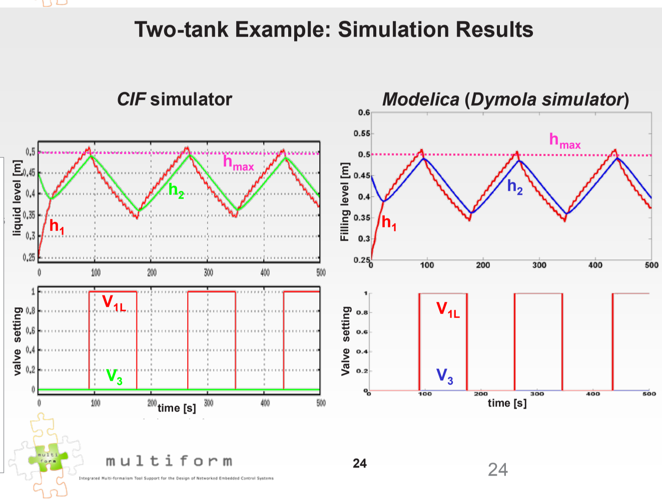

Flattened Example: tank_flat.cif
model TankController() =
|[var real V = 10.0 ; var real Qi ; var real Qo ; var nat n = 0
:: Tank_Controller:
|(
var string Controller_LP ; var string Tank_LP
; controlset Controller_LP, Tank_LP, V, n
; dyntypemap disc Controller_LP; disc Tank_LP; disc n; cont V;
mode X =
initial (((Tank_LP) = ("physics")) and (true))
and (((Controller_LP) = ("closed")) and (true))
inv ((Tank_LP) = ("physics")) => ((((V)') = ((Qi) - (Qo)))
and (((Qi) = ((n) * (5.0))) and ((Qo) = (sqrt(V)))))
tcp ((Controller_LP) = ("closed")) => (not ((V) <= (2)))
tcp ((Controller_LP) = ("opened")) => (not ((V) >= (10)))
(when (V) <= (2), (Controller_LP) = ("closed") do
{Controller_LP, n} : (new(n)) = (1), (new(Controller_LP)) = ("opened"))
(when (V) >= (10), (Controller_LP) = ("opened") do
{Controller_LP, n} : (new(n)) = (0), (new(Controller_LP)) = ("closed")) goto
X
)|
]|
multiform
19


A Two-tank System under Discrete Control
• Hybrid non-linear model of a two-tank system, modeled in gPROMS
– Designed to contain many constructs of the gPROMS language
– Controlled variables: h1, h2
– Manipulated (discrete) variables: V1L, V3
Taken from:
HYBRID SYSTEMS CONTROL
P1(t) = 0.00005*(sin(t) + 1) m3/s
(V1=V2=1)
(V2L=0)
Two-tank system in the graphical gPROMS model editor
multiform
20

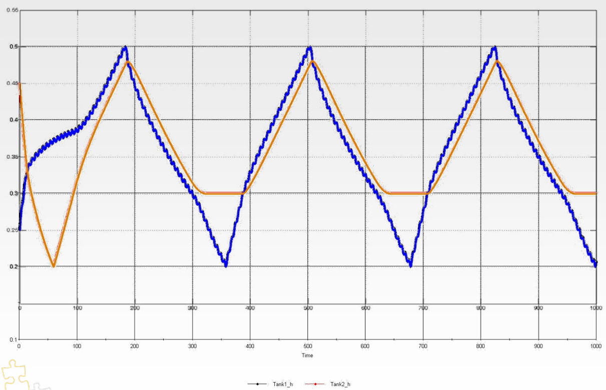

Two-tank Example: SFC Controller
The SFC controller keeps the filling levels h1 and h2 between hmin=0.2 m and hmax=0.5 m
If h1 exceeds hmax, valve V1L is opened for 80s
If h2 exceeds hmax, valve V3 is opened until h2 falls below hmin
S0
run
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
h1 >= hmax
h1 < hmax
SL
V1L
80 s
not(run)
S
V3
h2 >= hmax
h2 < hmax
h2 <= hmin
h2 > hmin
R
true
false
(S0)
multiform
21


Two-tank Example: Transformation Tool Chain
P1(t) = 0.00005*(sin(t) + 1) m³/s
(V1=V2=1)
(V2L=0)
~300 lines
Uncontrolled system (gPROMS)
Translation gPROMS → CIF
~900 lines
Uncontrolled system (CIF)
SFC controller + PLC model
Translation SFC → CIF
SFC Controller + PLC model (CIF)
~300 lines
Compo-sition
~1200 lines
Controlled system (CIF)
Translation CIF → Modelica
Controlled system (Modelica)
~850 lines
multiform
22
DC/FT: Software tool for the systematic refinement of informal specifications into SFCs


Chain
model TwoTanks_SFC () =
extern var Tanks_DOT_Tank1_DOT_h: cont real
mode v_tr1 = when l_u now do (c_c,t_rem) := (0, t_c - (time mod t_c)) goto v_tr2
, v_tr2 = when c_c >= t_rem now do (opt_V1,opt_V2,R_V1,R_V2,l_R_V2,l_u,c_c) := (0,0,false,true,false,0) goto v_tr3
|(//action "SL V1 End1"
// Main structure automaton a_str,m
mode
vi_2= when l_par2 and l_str2 and not(l_strm) now do (s_S2,l_str2,not_finished2):=(true,false,true) goto v_s_S2
when not(l_par2) and l_str2 and not(l_strm) now do (l_str2,not_finished2):=(false,false) goto vi_2
, v_s_S2= when (run) and l_str2 now do (s_Start2,s_S2,l_str2,not_finished2):=(true,false,false,true) goto v_s_Start2
when not(run) and l_str2 now do (l_str2,not_finished2):=(false,false) goto v_s_S2
, v_s_Start2= when (Tanks_DOT_Tank2_DOT_h<=t_lower) and l_str2 now do (s_Hei2,s_Start2,l_str2,not_finished2):=(true,false,false,true) goto v_s_Hei2
when (Tanks_DOT_Tank2_DOT_h>t_upper) and not(Tanks_DOT_Tank2_DOT_h<=t_lower) and l_str2 now do (s_Low2,s_Start2,l_str2,not_finished2):=(true,false,false,true) goto v_s_Low2
when not(Tanks_DOT_Tank2_DOT_h<=t_lower or Tanks_DOT_Tank2_DOT_h>t_upper) and l_str2 now do (not_finished2,l_str2):=(false,false) goto v_s_Start2
, v_s_Hei2= when (Tanks_DOT_Tank2_DOT_h>t_lower) and l_str2 now do (s_End2,s_Hei2,l_str2,not_finished2):=(true,false,false,true) goto v_s_End2
when not(Tanks_DOT_Tank2_DOT_h>t_lower) and l_str2 now do (not_finished2,l_str2):=(false,false) goto v_s_Hei2
, v_s_Low2= when (Tanks_DOT_Tank2_DOT_h<=t_upper) and l_str2 now do (s_End2,s_Low2,l_str2,not_finished2):=(true,false,false,true) goto v_s_End2
when not(Tanks_DOT_Tank2_DOT_h<=t_upper) and l_str2 now do (not_finished2,l_str2):=(false,false) goto v_s_Low2
, v_s_End2 = when (not(run)) and l_str2 now do (s_FEnd_1,s_End2,l_par2,l_str2,not_finished2):=(true,false,false,false,false) goto vi_2 //
last step
when (run) and not(not(run)) and l_str2 now do (s_Start2,s_End2,l_str2,not_finished2):=(true,false,false,true) goto v_s_Start2
when not(not(run) or run) and l_str2 now do (not_finished2,l_str2):=(false,false) goto v_s_End2
// last step, return to start (step variable is deactivated by main automaton)
:: vi_2
)|
//SFC end
)| // structure automaton


Two-tank Example: Simulation Results
CIF simulator
liquid level [m]
hmax
h2
h1
valve setting
V1L
V3
time [s]
Modelica (Dymola simulator)
Filling level [m]
Valve setting
multiform
24
24

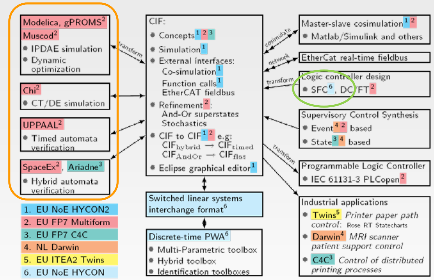
Output Identical



multiform

25

Compositional Interchange Format (CIF)



http://se.wtb.tue.nl/sewiki/cif/start

References:

Fischer, S.; Hüfner, M.; Sonntag, C.; Engell, S.: Systematic Generation of Logic Controllers in a Model-based Multi-formalism Design Environment. Proc. 18th IFAC World Congress, 28.08.-02.09.2011, 12490-12495.

Hendriks, D.; Schiffelers, R.; Hüfner, M.; Sonntag, C.: A Transformation Framework for the Compositional Interchange Format for Hybrid Systems. Proc. 18th IFAC World Congress, 28.08.-02.09.2011, 12509-12514.

Sonntag, C.; Hüfner, M.: On the Connection of Equation- and Automata-based Languages: Transforming the Compositional Interchange Format to Modelica. Proc. 18th IFAC World Congress, 28.08.-02.09.2011, 12515-12520.

multiform

26

Equation-based vs. Automaton-based Formalisms

- Simulation/Solver/Tool options encoded in model code (e.g. *EcosimPro*, *gPROMS*)
  - Tool specific options cannot be transformed
    ➔ Other tools might not find a solution for a difficult initialization problem
- Formal semantics not available ➔ Transformation not provably correct
- Equation-based models can be more restrictive than automata models
  - E.g. *Modelica* enforces globally and locally balanced models
    ➔ Automata models need to be preprocessed
    - Either by flattening of the model
    - Or by rebuilding the automata structure in an equation-based formalism

multiform

27

Summary

- There is a need for efficient model-based support of the design of complex automated systems with trans-level propagation and iteration, and re-use of models
- An all-encompassing mega-tool for the design of complex automated systems is not realistic, so several tools and modeling formalisms must be used in the design process.
- Three different routes to tool and model integration and design support were pursued in MULTIFORM:
  - Model exchange and tool chains via the *CIF*
  - Direct coupling of tools for testing of specifications
  - Propagation of parameters via the Design Framework

multiform

28

Lessons and Challenges from MULTIFORM

- The *CIF* and its tool set are stable and relatively mature
- Available under open source licence
- The effort for developing model transformations is high
- Transformations from the CIF in most cases can only be performed for subsets of the models which can be represented in the formalism.
  - A **formal** specification of the the supported *CIF* subset of a tool is needed
- It should be possible to trace elements of a model after the transformation
- Model blow-up is not as bad as could be expected

multiform
Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

29

Lessons and Challenges from MULTIFORM (2)

- The *CIF* is very expressive and well suited for model exchange between automata-based formalisms, but conceptually different from equation-oriented languages (e.g. *Modelica*, *gPROMS*, *EcosimPro*)
  - **Possible solution:** Use a *Modelica* subset as an exchange formalism for equation-oriented languages, bridge equation- and automata-oriented formalisms via the *CIF* ↔ *Modelica* transformation
- Often only some elements of a system are modeled precisely, and these models are formulated in different formalisms (*fragmented modeling*)
  - How can the interdependencies between model fragments be **formally** described and exploited?
- **Model ontology needed**
  - Specification of model formalism expressivity using a common formal vocabulary
  - Equipping model artifacts with meta data on their origin(s) → traceability
  - Description of relations of partial models to an overall model

multiform
Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

30

EQUATION-BASED MODELING AND CONTROL OF INDUSTRIAL PROCESSES

Johan Sjöberg, ABB AB, Corporate Research and Linköping university

Control systems are used in a vast number of different applications. However, despite the differences, the objective is similar. That is, to control the application in the most efficient manner possible. In order to find out what is most efficient, models are an important tool. The models can be derived using more or less physical relations, but there is a trend towards using more and more physical insights since it has the advantage that the model can be used in an extrapolating way. That is, it is possible to draw conclusions based on the model without having measurements from the considered operating region. This is a strength when optimizing the process and the best possible operating condition might very well be significantly different compared to usual operating conditions for which data is available.

The model building is a very large share of the work and it is therefore imperative to be able to reuse models developed for earlier applications. This makes equation-based modeling very interesting for companies such as ABB that develop control strategies for many different applications. ABB has therefore developed tools for equation-based modeling and control for a long time. These tools have also been successfully applied to many different processes, such as pulp and paper mills, power plants, cement industries, etc.

In this presentation, we focus on two different applications where equation-based modeling has been key in the process optimization. The first application is hot rolling mills, while the second application is harbor cranes.

Hot rolling mills are complicated industries where the overall model has to incorporate physical relations from many different domains, such as mechanics, thermodynamics and metallurgy. For hot rolling mills, ABB has developed a model that describes the material throughout the whole rolling including reheating, rolling and cooling. This model has been used to compute operating conditions that minimizes the energy consumption while maintaining the material properties.

For harbor cranes, ABB has developed a tool to improve the process of picking up coal from a ship and put it in a hopper at shore. In this context, process improvement normally means minimization of the cycle time, but today it is also important to limit the energy consumption.

Johan Sjöberg, ABB Corporate Research, Västerås, Sweden

Equation-based Modeling and Control of Industrial Processes

A Global Leader in Power and Automation Technologies
Leading Market Positions in Main Businesses

- 135,000 employees in about 100 countries
- $38 billion in revenue (2011)
- Formed in 1988 merger of Swiss and Swedish engineering companies
- Predecessors founded in 1883 and 1891
- Publicly owned company with head office in Switzerland

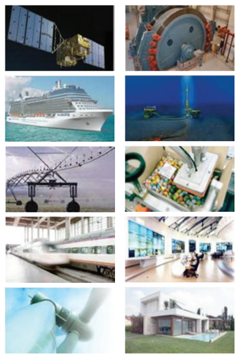
How ABB is organized
Five global divisions

Power Products	Power Systems	Discrete Automation and Motion	Low Voltage Products	Process Automation
$10.3 billion 35,000 employees	$7.7 billion 19,500 employees	$8.4 billion 27,500 employees	$5.0 billion 21,000 employees	$7.8 billion 28,500 employees

(2011 revenues, consolidated)

- ABB's portfolio covers:
  - Electricals, automation, controls and instrumentation for power generation and industrial processes
  - Power transmission
  - Distribution solutions
  - Low-voltage products
  - Motors and drives
  - Intelligent building systems
  - Robots and robot systems
  - Services to improve customers productivity and reliability

Power and automation are all around us
You will find ABB technology…



orbiting the earth and working beneath it,

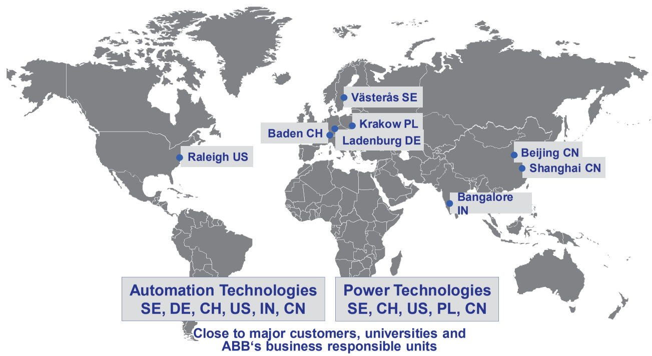
crossing oceans and on the sea bed,

in the fields that grow our crops and packing the food we eat,

on the trains we ride and in the facilities that process our water,

in the plants that generate our power and in our homes, offices and factories

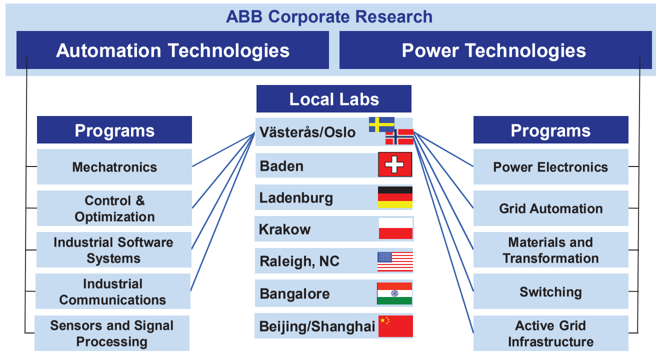
Global Labs…
… and Local Lab Locations



© ABB Group
October 24, 2012 | Slide 5

ABB

Global Labs, Corporate Programs and Locations
700 Researchers World-Wide – 250 in Västerås/Oslo



© ABB Group
October 24, 2012 | Slide 6

ABB

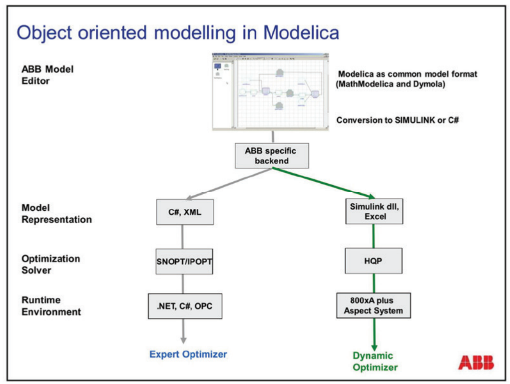
Examples of Industrial Systems modeled using an Equation-based Approach at ABB

- Pulp & paper
- Metals & minerals
- Mechatronical systems
- Power generation
- Power products
- Power systems
- …

© ABB Group
October 24, 2012 | Slide 7

ABB

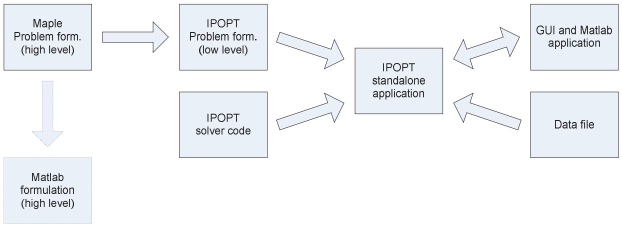
Tools for Equation-based Simulation and Control Modelica



© ABB Group
October 24, 2012 | Slide 8

ABB

Tools for Equation-based Simulation and Control
Maple to IPOPT



- **Maple (high level description)**

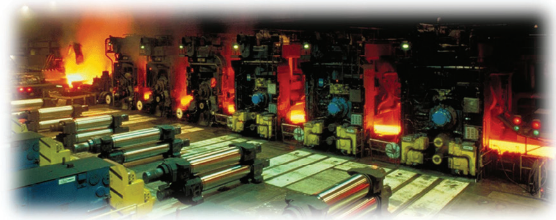
```
cact := [];
cact := [op(cact), HeightIn*WidthIn-AreaIn];

# Height out
cact := [op(cact), HeightOut-AGroove/MaxGrooveWidth-Gap];
```

- **IPOPT (low level description)**

```
g[i++] = x[2 + 170 * k] * x[3 + 170 * k] - x[25 + 170 * k];
g[i++] = x[20 + 170 * k] - x[6 + 170 * k] / x[5 + 170 * k] - x[170 * k];
```

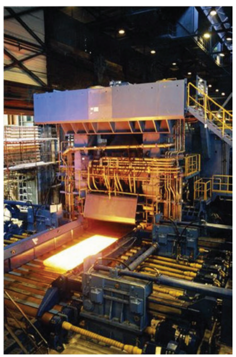

ABB

Optimization of Hot Rolling



ABB

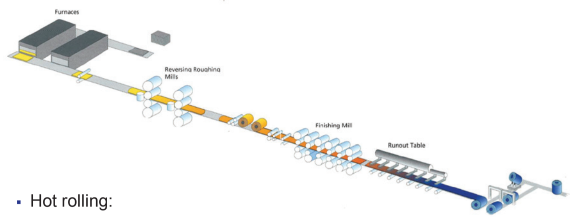
(Energy) Optimization in Hot Rolling Mills
Huge Potential



- Profile mills:
  - 7% reduction => 1.2 GWh/yr (>1000 profile mills globally)
- Flat mills:
  - 0,5% reduction => 1.6 GWh/yr (~400 flat mills globally)

ABB

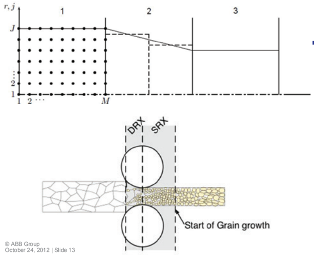
Introduction to Hot Rolling



- Hot rolling:
  - Temperature above the recrystallization temperature. (otherwise cold rolling).
- Profile rolling
  - To produce rods, bars, wires, etc.

ABB

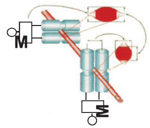
Process Model



- Covers room temperature to room temperature
- The model includes for instance
  - Mass balance, rolling geometries, power, groove utilization, temperature field, microstructure of material
- Model properties:
  - Large but sparse
  - Discontinuities due to switching behavior
  - Bad numerical scaling of certain equations

© ABB Group
October 24, 2012 | Slide 13

Optimization Formulation



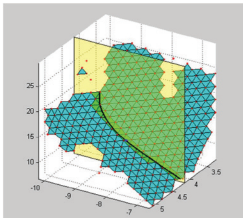
$$\min f(v, g, htc, T_0, \mathcal{X})$$
$$\text{s.t.} \quad \left.\begin{array}{l} c_i(v, g, htc, T_0, \mathcal{X}) = 0,\ i \in \mathcal{E} \\ c_i(v, g, htc, T_0, \mathcal{X}) \geq 0,\ i \in \mathcal{I} \end{array}\right\} \approx 700 + 45$$
$$A(htc, \mathcal{X})T^{m+1} = b(T^m, htc, \mathcal{X})\} \approx 10000 - 100000$$

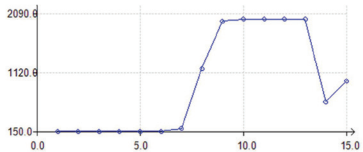
v = roll speed
g = gap
T_0 = furnace temperatures
htc = heat transfer coefficients
$\mathcal{X}$ = intermediate variables and parameters such as T_j, $\sigma, \ldots$

© ABB Group
October 24, 2012 | Slide 14

Multiobjective Optimization

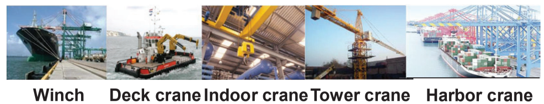
- Real life: Compromise between different objectives:
  - Total power
  - Austenite grain size (related to strength of material)
  - Production speed
- Pareto front analysis yields many insights, for instance, for the cooling.
  - Grain size reduction requires cooling. For low energy consumption, start from behind.

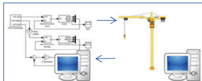




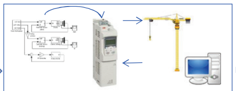
© ABB Group
October 24, 2012 | Slide 15

Optimization & Optimal control are important but…
Hardware-in-the-loop





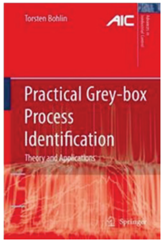
Control code development based on mathematical representation of application (e.g. tower crane) using Modelica



Control code verification using hardware (Drive, PLC) in the loop

Site test of new automation solutions

Optimization & Optimal control are important but…
System Identification - Grey-box identification



Book by Torsten Bohlin
Springer, 2006

- In an optimization project, modelling is by far the most time-consuming part
- Physical model should be gradually extended while testing statistical significance against experimental data
- Estimation of parameters using horizon estimation (HE) gives biased parameters without the right regularization in many cases.

$$\min_{x_k,\theta} \frac{1}{2}\sum_{k=0}^{N} w_k^T Q^{-1} w_k + v_k^T R^{-1} v_k + \log(\det(S_k(\theta)))$$

$$\text{s.t.}\ \ x_{k+1} = f(x_k, u_k) + w_k$$
$$y_k = h(x_k, u_k) + v_k$$
$$x(0) = x_0$$

ABB

Equation-based Modeling and Control for Industry
Conclusions

- Modeling
  - More and more use of first principle modeling
  - Requires considerable knowledge to succeed – process as well as theoretical
- Optimization
  - Optimization and decision support are slowly gaining ground
  - Increased competition will force more and more optimization solutions
  - More plant-wide & wider scope (production scheduling etc)
  - Optimal solution often used for comparison, not for the actual control

ABB

Equation-based Modeling and Control for Industry
Conclusions, cont'd.

- Identification
  - Parameter identification is not supported enough yet.
- Virtualization
  - Hardware-In-the-Loop / training simulators gain popularity
- Generally
  - Ease of use (incl. look-and-feel)
  - Model management
  - Integration (process, data, tool etc)

ABB

Power and productivity
for a better world™

ABB

Proposed grey-box scheme for nonlinear systems

1. Model process in Modelica
2. Discretize symbolically and export equations
3. Linearize model symbolically
4. Import and prepare data
5. Carefully introduce noise variables at equations motivated by physical insight
6. Solve by nonlinear programming (e.g. using IPOPT)

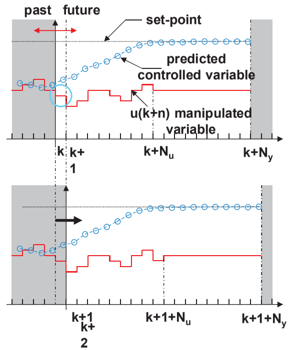
$$\min_{x_k,\theta} V = \frac{1}{2}\sum_{k=0}^{N}(w_k^T Q^{-1} w_k + v_k^T R^{-1} v_k + \log(\det(S_k(\theta)))$$

subject to

$$x_{k+1} = g(x_k, u_k) + w_k$$
$$y_k = h(x_k, u_k) + v_k$$

7. For every evaluation of V calculate the (time varying) linearized system along the trajectory to compute S_k
8. Test which parameters to make free (including noise parameters) by hypothesis testing using the chi-squared risk calculation
9. Repeat 5-8 until no further improvement

Model Predictive Control (MPC) Algorithm



Use model to

- Estimate where you are – state estimation
- Optimize future control signals over a time horizon

- Repeat at next sampling instant
- Shift horizon one step – receding horizon control

FMI: FUNCTIONAL MOCKUP INTERFACE FOR MODEL EXCHANGE AND CO-SIMULATION

Torsten Blochwitz, ITI GmbH Dresden, Germany

The Functional Mockup Interface (FMI) is a tool independent standard for the exchange of dynamic models and for co-simulation. The FMI was developed in a close collaboration between simulation tool vendors, research institutes and industrial users within the European joint research project MODELISAR. It is continued as Modelica Association Project since 2012. More than 30 tools support FMI, and it is heavily used in industrial and scientific projects, not only in the automotive sector. The presentation explains the technical concepts of FMI and demonstrates some industrial applications. Additionally an overview about version 2.0 of FMI is given that combines the formerly separated interfaces for Model Exchange and Co-Simulation in one standard.

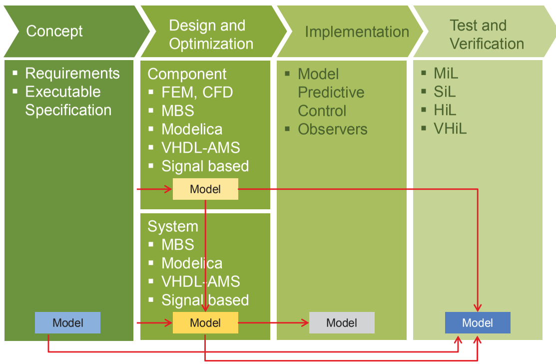
Functional Mockup Interface for Model Exchange and Co-Simulation

T. Blochwitz	ITI, Dresden
M. Otter	DLR, Oberpfaffenhofen
J. Akesson	Modelon, Lund,
M. Arnold	University of Halle
C. Clauß	Fraunhofer IIS EAS, Dresden
H. Elmqvist, H. Olsson	Dassault Systèmes, Lund
M. Friedrich,	Simpack AG, Gilching
A. Junghanns, J. Mauss	QTronic, Berlin
D. Neumerkel	Daimler AG, Stuttgart
A. Viel	LMS Imagine, Roanne

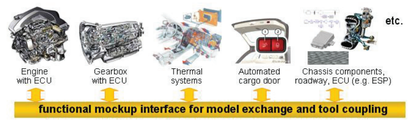
Contents

Motivation

Modeling and Simulation are applied in all stages of system design



Motivation



Challenges for Functional Mockup:
- Different tools and languages are involved
- No standards for model interface and co-simulation available
- Protection of model IP and know-how of supplier

Modelisar project:
- **Functional Mockup Interface for Model Exchange and Co-Simulation**

Functional Mockup Interface

EU project Modelisar (2008 – 2011, 26 Mill. €, 178 my)

- Initiated by Daimler AG, 28 European partners
  - Tool vendors
  - Users
  - Research organizations
- Proof of concept in industrial use cases

After 2011

- Continuation as Modelica Association Project
- Modelica Association changed its bylaws to become an umbrella organization for projects related to model based system design

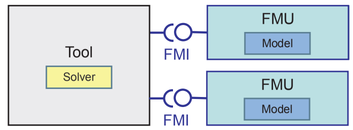
MODELISAR
(ITEA 2 ~ 07006)

Partners

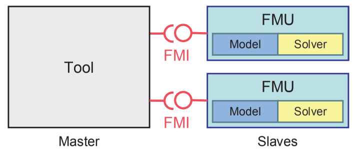
ARMINES
Arsenal Research
ATB
AVL
Berata
Daimler
Dassault Systèmes
David
DLR
Dynasim
Extessy
FhG First, IIS EAS, SCAI
Geensys
Halle University
IFP
Imagine
INSPIRE
SIMPACK AG
ITI
LMS International
QTronic
Schneider Electric
Trialog
Triphase
TWT
Verhaert
Volkswagen
Volvo

FMI – Main Design Idea

- FMI for Model Exchange

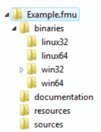


- FMI for Co-Simulation



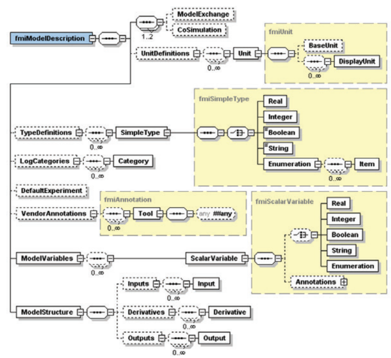
FMI – Main Design Idea

- A component which implements the interface is called a *Functional Mockup Unit (FMU)*
- Separation of:
  - Description of interface data (XML file)
  - Functionality (API in C)
- An FMU is a zipped file (*.fmu) containing:
  - modelDescription.xml
  - Implementation in source and/or binary form
  - Additional data and functionality
- One FMU can contain implementations of both interfaces



XML Model Description

Interface definition is stored in one xml-file:



Implementation and capability flags

Definition of units

Definition of variable types

Variables and their attributes

Dependency information

Example

```xml
<?XML version="1.0" encoding="UTF-8"?>
<fmiModelDescription
  XMLns:xsi="http://www.w3.org/2001/.."
  xsi:noNamespaceSchemaLocation="fmiModel.."
  fmiVersion="2.0"
  modelName="FMU_Coupling.DriveTrain_TorqueAtEnd"
  guid="{a4976b5c-b9f7-432a-9dd3-e80bafaac060}"
  ...>
  <ModelExchange
    modelIdentifier="FMU_0Coupling_..."
    canGetAndSetFMUstate="true"
    providesPartialDerivativesOf_DerivativeFunction_wrt_States="true"
    providesDirectionalDerivatives="true"/>
  <CoSimulation
    modelIdentifier="FMU_0Coupling_..."
    canHandleVariableCommunicationStepSize="true"
    canInterpolateInputs="true"
    .../>
  <UnitDefinitions>
    <Unit name="N.m">
      <BaseUnit kg="1" m="2" s="-2"/> </Unit>
  </UnitDefinitions>
  <TypeDefinitions>
    <SimpleType
      name="Modelica.SIunits.Torque">
      <Real quantity="Torque" unit="N.m"/>
    </SimpleType>
    ...
  </TypeDefinitions>
  <DefaultExperiment startTime="0.0"
    stopTime="1.0" tolerance="0.0001"/>
    ...
```


Example

```xml
  ...
  <ModelVariables>
    <ScalarVariable
      name="torque"
      valueReference="335544320"
      description="Torque in flange"
      causality="output">
      <Real
        declaredType=
         "Modelica.Blocks.Interfaces.RealOutput"
        unit="N.m"/>
      ...
  </ModelVariables>
  <ModelStructure>
    <Inputs>
      <Input name="phi"/>
      <Input name="w" derivative="1"/>
    </Inputs>
    <Derivatives>
      <Derivative
        name="der(inertia.phi)"
        state="inertia.phi"
        stateDependencies="2"
        inputDependencies=""/>
      <Derivative
        name="der(inertia.w)"
        state="inertia.w"/>
    </Derivatives>
    <Outputs>
      <Output name="torque"
       inputDependencies="1 2"
       inputFactorKinds="fixed fixed"/>
    </Outputs>
  </ModelStructure>
</fmiModelDescription>
```


C-Interface

- Instantiation:

```
fmiComponent fmiInstantiateModel(fmiString instanceName, ...)
fmiComponent fmiInstantiateSlave(fmiString instanceName, ...)
```

  - Returns an instance of the FMU. Returned `fmiComponent` is an argument of the other interface functions.
- Functions for initialization, termination, destruction
- Support of real, integer, boolean, and string inputs, outputs, parameters
- `Set` and `Get` functions for each type:

```
fmiStatus fmiSetReal    (fmiComponent c,
                          const fmiValueReference vr[], size_t nvr,
                          const fmiReal value[])
fmiStatus fmiSetInteger(fmiComponent c,
                          const fmiValueReference vr[], size_t nvr,
                          const fmiInteger value[])
```

- Identification by `valueReference`, defined in the XML description file for each variable

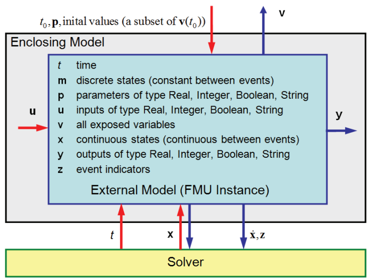
FMI for Model Exchange
Features

- Functionality of state of the art modeling methods can be expressed
- Support of continuous-time and discrete-time systems
- Model is described by differential, algebraic, discrete equations

- Interface for solution of Ordinary Differential Equations (ODE)
- Handling of time, state and step events, event iteration

- Discarding of invalid inputs, state variables
- No explicit function call for computation of model algorithm
  - FMU decides which part is to be computed, when a `fmiGetXXX` function is called
  - Allows for efficient caching algorithms

FMI for Model Exchange
Signals



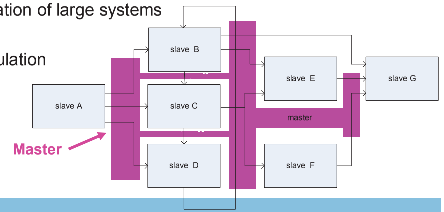
Co-Simulation

Definition:

- Coupling of several simulation tools
- Each tool treats one part of a modular coupled problem
- Data exchange is restricted to discrete communication points
- Subsystems are solved independently between communication points

Motivation

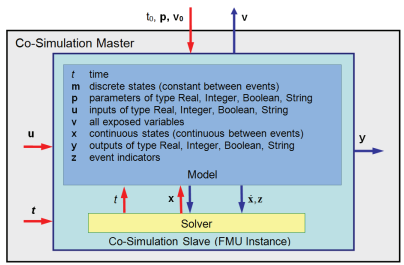
- Simulation of heterogeneous systems
- Partitioning and parallelization of large systems
- Multirate integration
- Hardware-in-the-loop simulation



FMI for Co-Simulation
Features

- State-of-the-Art Co-Simulation:
  - Fixed communication step size
- To improve accuracy and robustness:
  - Optional variable communication step size
  - Optional higher order approximation of inputs and outputs
  - Optional repetition of communication steps
- Capabilities of the slave are contained in the XML-file, for example:
  - `canHandleVariableCommunicationStepSize`
  - `canInterpolateInputs`
  - `canGetAndSetFMUstate`
- Master can decide which coupling algorithm is applicable
- Asynchronous execution (allows parallel execution)

FMI for Co-Simulation
Signals



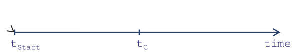
Additional:

- Status information
- Derivatives of inputs, outputs w.r.t. time for support of higher order approximation

FMI for Model Exchange and Co-Simulation
Sample Code

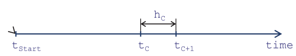
- Model Exchange:
  (One model evaluation)

```
/* Set inputs*/
fmiSetReal(m, id_u1, u1, nu1);
fmiSetTime(m, tC);
fmiSetContinuousStates(m, x, nx);
/* Get results */
fmiGetDerivatives(m, derx, nx);
fmiGetEventIndicators (m, z, nz);
fmiGetReal(m, id_u1, u1, nu1);
```

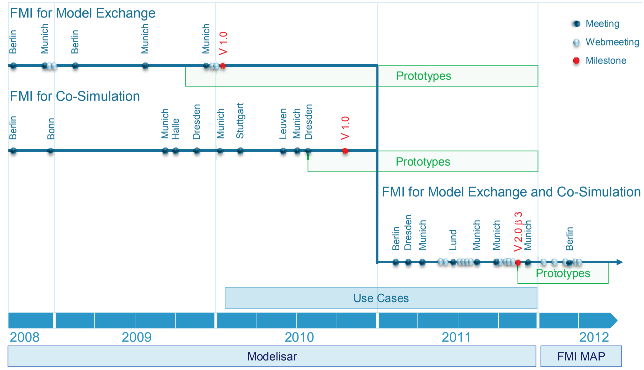


- Co-Simulation:
  (One communication step)

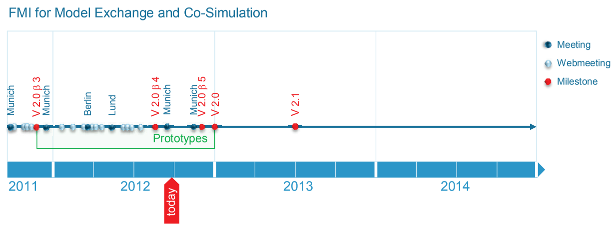
```
/* Set inputs*/
fmiSetReal(s, id_u1, u1, nu1);
/* Do computation*/
fmiDoStep(s, tC, hC, fmiTrue);
/* Get results */
fmiGetReal(s, id_u1, u1, nu1);
```



Development Process



Development Process



FMI 2.0 specification:

- Release December 2012
- Valid for several years
- Backwards-compatible enhancements in minor releases

FMI 2.0
New Features

- Motivation for FMI 2.0
  - Clarification of specification document
  - Ease usability
  - Increase performance for large models
- Unification of Model Exchange and Co-Simulation Standard
  - FMU can contain implementations of both interfaces
  - Distributed and tool based use cases now also for Model Exchange
- Many minor changes
  - Definition of log categories
  - Removement of alias and anti alias variables to ease usage
  - Continuous state variables are named and ordered
  - Improved unit handling

Current status of FMI 2.0

Clarification of specification:
- Instantiation
- Classification of variables
- Calling sequence

Features:
- Tunable parameters
- Improved unit handling
- Save and restore FMU state
- Detailed dependency information (inputs, outputs, derivatives)
- Efficient interface to partial derivatives — Contained in public Beta 4
- Improved handling of time events — Under Discussion

 23

FMI 2.0
Classification of interface variables

`causality`
- `parameter`
- `input`: output of another model
- `output`: input for another model
- `local`: not to be used by other models

`variability`
- `constant`
- `fixed`: constant after initialization
- `tunable`: constant between events
- `discrete`: changes at event instances
- `continuous`

- Combination of `causality` and `variability` allows clear classification of all kinds of variables
- New: distinction between `tunable` and `fixed parameters`
  - Stop simulation, set tunable parameters, resume simulation

 24

FMI 2.0
Save and Restore FMU State

- FMI 1.0: implicite save and restore depending on arguments of `fmiDoStep`
- FMI 2.0: explicite function calls

```
fmiStatus fmiGetFMUstate(fmiComponent c, fmiFMUstate* FMUstate)
fmiStatus fmiSetFMUstate(fmiComponent c, fmiFMUstate  FMUstate)
```

- Iterative co-simulation algorithms
  - Repeat more than one communication step
- Model Predictive Control
  - Simulate some steps starting from the same state with different sets of input values
  - Use the optimal set as control value for the real system
- FMU state can be serialized into a byte vector
  - Usage: start a training simulator from a certain scenario

 25

FMI 2.0
Dependency Information

- FMI 1.0:
  - Only dependencies of outputs on inputs can be indicated
- FMI 2.0:
  - Dependencies of outputs on continuous states
  - Dependencies of derivatives on continuous states and inputs
- Usage:
  - Detection of algebraic loops
  - Definition of sparsity pattern of Jacobian matrices

 26

FMI 2.0
Dependency Information

- Kind of dependency is also defined:
  - `nonlinear`: Jacobian entry is not constant
  - `fixed`: Jacobian entry is constant
  - `discrete`: Jacobian entry may change after events
- Allows optimizations:
  - Generate linear systems of equations for solution of algebraic loops if possible
  - Reduce number of Jacobian computations

FMI 2.0
Directional Derivatives (Jacobian Matrices)

- Jacobians are needed for:
  - Implicit integration methods
  - Solution of systems of equations resulting from algebraic loops
  - Linearization of FMU
  - Extended Kalman filters
- Numerical computation is expensive for large models
- Optional function for providing directional derivatives
  `fmiStatus fmiGetDirectionalDerivative(fmiComponent c,..)`
- Arguments define which derivative(s) w.r.t. which variable(s) are to be retrieved

FMI 2.0
Time Event Handling (under Development)

Requirements:

- Guarantee synchronicity of time events
- Support a subset of the synchronous extensions from Modelica 3.3 (time triggered clocks with constant and variable period)
- Allow backward compatible extensions
- Usable for tools without synchronous features

Main design idea:

- FMU exposes base rates and clocks in the XML model description
- Clock ticking is signaled by `fmiSetClock(..)` before `fmiEventUpdate(..)`
- Discrete variables can be associated with clocks (optional) in XML model description

FMI 2.1
Hierarchical Data, Buses, Physical Connectors (planned)

Requirements:

- Group variables to hierarchical structures, connectors
- Signal based tools must not be excluded
- Keep type information of connectors (e.g. `Modelica.Electrical.Analog.Interfaces.Pin`)
- Add connector type definition for reconstruction of connector type or mapping to existing types

Main design idea:

- Additional “layer” in XML model description
- Mark input/output variables as flow or across quantities
- Causality (input, output) is fixed

Roadmap

2012:
- Finalize time event handling
- October: FMI Meeting
- November: Release of public beta 5
- December: Release of FMI 2.0
- Coordinated prototype implementations by tool vendors

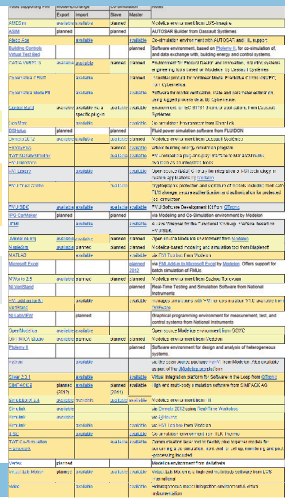
2013:
- Backwards-compatible extensions
- Support of arrays and hierarchical data
- Bus and physical connectors
- Graphical appearance
- …

FMI Support in Tools
fmi-standard.org/tools

- Tool support started immediately after release of FMI 1.0
- **32** tools support FMI, **9** intend to
- Within Modelisar project: **15**

FMI Support in Tools

- Authoring Tools: **12**
- Integration Tools: **20**
  (Co-Simulation master, HiL, optimization, control, analyses)
- Software Development Kits: **3**
  (C, Python, Java)

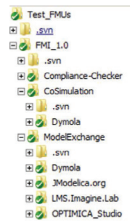


Quality of FMI Implementations

FMI Compliance Checker
- Open source implementation under contract of MA
- Checks XML model description
- Simulates single FMUs for Model Exchange and Co-Simulation
- https://svn.fmi-standard.org/fmi/branches/public/Test_FMUs/FMI_1.0/Compliance-Checker/

Repository of FMUs, generated by different tools
- https://svn.fmi-standard.org/fmi/branches/public/Test_FMUs

Public Error Tracking System
- https://trac.fmi-standard.org/



Applications outside of Automotive

Power plant simulation and control
- Siemens, ABB, EDF
- EU Project MODRIO (19 Mill. €, 150 man-years, 2012 – 2015)

Building simulation
- Situation is similar to automotive industry:
  - Heterogeneous systems (building, heating, air conditioning, …)
  - Components of different nature and from several suppliers

Research
- Co-Simulation master algorithms
- Model based control

FMI Modelica Association Project (MAP)

General conditions
- FMI project members need not to be Modelica Association (MA) members
- Project results are owned by the MA
- Project results are freely usable under copyleft license
- Meetings are open to the public

FMI Steering Commitee
- Defines FMI policy, strategy, feature roadmap, releases
- Voting rights

FMI Advisory Board
- Contribute to FMI design
- Access to FMI infrastructure (svn, trac, meeting minutes)

FMI Project Rules
How to participate

Steering Commitee
- Prove active FMI support by participation at 2 meetings in the last 24 months
- Support FMI or part of it in a commercial or open source tool, and/or active FMI usage in industrial projects
- Be accepted by Steering Commitee with qualified majority

Advisory Board
- Prove active FMI support by participation at 2 meetings in the last 24 months

Guests
- Send e-mail to contact@fmi-standard.org for registration in mailing list

FMI MAP Members

Steering Committee
- Atego, Daimler, Dassault Systèmes, IFP EN, ITI, LMS, Modelon, QTronic, Siemens, SIMPACK

Advisory Board
- Armines, DLR, Fraunhofer (IIS/EAS, First, SCAI), Open Modelica Consortium, TWT, University of Halle

Guests
- Altair Engineering, Berkeley University, Bosch, ETAS, Equa Simulation, IBM Research

Conclusions

FMI for Model Exchange and Co-Simulation is an established standard

- 32 tools currently support FMI 1.0, 9 intend to
- Is used in industrial and research applications
- Is maintained as Modelica Association Project

FMI project is open for non Modelica tool vendors and organizations

FMI 2.0 improves:

- Compatibility of implementations (clarified specification)
- Usability (tunable parameters, unit handling)
- Efficiency and robustness for large models (dependency information, directional derivatives)

 39

Outlook

FMI 2.0 Release planned for December 2012

Current tasks:

- Precise handling of time events for periodic and aperiodic sampled data systems

Ideas for FMI 2.1

- Arrays, hierarchical data, buses, physical ports
- Graphical appearance, connector placement

 40

VERTICAL INTEGRATION IN TOOL CHAINS FOR MODELING, SIMULATION AND OPTIMIZATION OF LARGE-SCALE SYSTEMS

Johan Åkesson, Modelon AB and Lund University, Lund, Sweden

In recent years, languages such as Modelica and VHDL-AMS have emerged as intuitive user and application-oriented high-level description formats suitable for modeling of physical systems. This trend has been further strengthened by the availability of software tools for modeling, simulation and optimization, which enable engineers to rapidly develop detailed models of complex systems composed from sub-systems from different physical domains. While the capabilities of such tools in terms of performance match the requirements in challenging industrial applications, tool interoperability has traditionally received little attention. Rather, tools have been designed as monolithic software environments, with dedicated interfaces to numerical algorithms. As a result, flexible creation of tool chains where several computational tools are assembled into a workflow tailored to a particular design process is often difficult. Driven by the observation that one single tool will not be able to provide the solution to the computational needs of the future, new challenges have emerged.

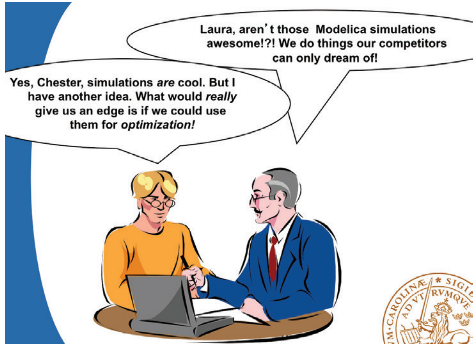
In order to meet these challenges, three aspects of computational tool chains need considered. Firstly, modeling languages are critical to provide comprehensive environments for engineering practitioners, as well as for enabling formal analysis of model properties. In addition, language extensibility, both in terms of the language itself and in terms of tools supporting it, requires attention in order to enable flexible tailoring of existing languages to specific needs arising when formulating different systems design problems. Secondly, open interfaces plays a key role in achieving tool interoperability. A recent example is the Functional Mock-up Interface (FMI), which has received considerable attention in the simulation tool community. Finally, symbolic and numerical algorithms designed to solve systems design problems need to be employed. In this presentation, challenges arising when integrating different tools into complete tools chains will be discussed. Particular attention will be given to the three mentioned aspects: languages, open interfaces and algorithms. Examples will be drawn from experiences from developing, integrating and using open source tools, notably CasADi and Jmodelica.org, in industrial projects where large-scale optimization techniques has been applied to Modelica models derived from first principles.

Vertical Integration in Tool Chains for Modeling Simulation and Optimization of Large-Scale Systems

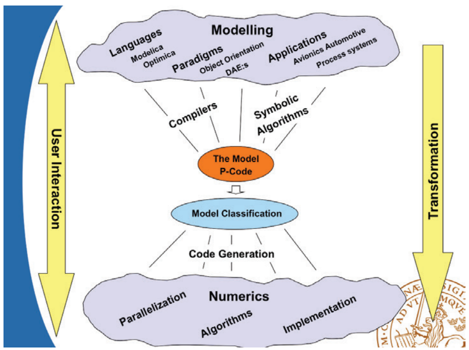
Johan Åkesson, Modelon AB/Lund University

Thanks to
Joel Andersson, Niklas Andersson, Magnus Gäfvert, Staffan Haugwitz, Görel Hedin, Per-Ola Larsson, Alexandra Lind, Kilian Link, Fredrik Magnusson, Elin Sällberg, Stephane Velut

In 2006…



The Landscape



Outline

- Modelica
- Application examples
- Extension example
- Interface example
- Towards a vertically integrated tool chain
- Challenges

What is Modelica?

- A language for modeling of complex heterogeneous physical systems
  - Open language
    - Modelica Association (www.modelica.org)
  - Several tools supporting Modelica
    - Dymola
    - OpenModelica (free)
    - MosiLab
    - Scilab/Scicos (free)
  - Extensive (free) standard library
    - Mechanical, electrical, thermal etc.

Key Features of Modelica

- Declarative equation-based modeling
  - Text book style equations
- Multi-domain modeling
  - Heterogeneous modeling
- Object oriented modeling
  - Inheritance and generics
- Software component model
  - Instances and (acausal) connections
- Graphical and textual modeling

A Simple Modelica model

Differential equation

$\dot{x}(t) = ax(t) + bu(t)$

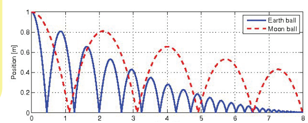
Class definition, **Parameter declaration**, **Variable declaration**, **Initialization**, **Derivative operator**, **Equation**

```
model FirstOrder
  input Real u;
  parameter Real b = 1;
  parameter Real a = -1;
  Real x(start=1);
equation
  der(x) = a*x + b*u;
end FirstOrder;
```

Hybrid modeling

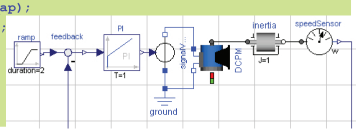
```
class BouncingBall //A model of a bouncing ball
  parameter Real g = 9.81; //Acceleration due to gravity
  parameter Real e = 0.9; //Elasticity coefficient
  Real pos(start=1); //Position of the ball
  Real vel(start=0); //Velocity of the ball
equation
  der(pos) = vel;   // Newtons second law
  der(vel) = -g;
  when pos <=0 then
    reinit(vel,-e*pre(vel));
  end when;
end BouncingBall;
```

```
class BBex
  BouncingBall eBall;
  BouncingBall mBall(g=1.62);
end BBex;
```

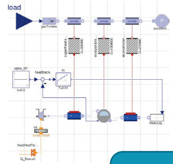


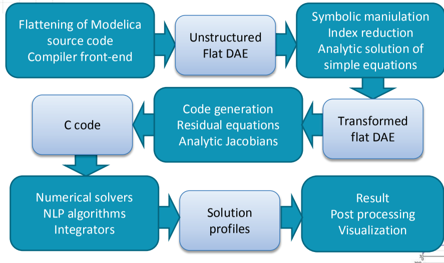
Graphical Modeling

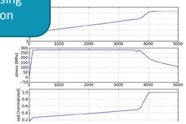
```
model MotorControl
  Modelica.Mechanics.Rotational.Inertia inertia;
  Modelica.Mechanics.Rotational.Sensors.SpeedSensor speedSensor;
  Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_PermanentMagnet DCPM;
  Modelica.Electrical.Analog.Basic.Ground ground;
  Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage;
  Modelica.Blocks.Math.Feedback feedback;
  Modelica.Blocks.Sources.Ramp ramp(height=100, startTime=1);
  Modelica.Blocks.Continuous.PI PI(k=-2);
equation
  connect(inertia.flange_b, speedSensor.flange_a);
  connect(DCPM.flange_a, inertia.flange_a);
  connect(speedSensor.w, feedback.u2);
  connect(ramp.y, feedback.u1);
  connect(signalVoltage.n, DCPM.pin_ap);
  connect(signalVoltage.p, ground.p);
  connect(ground.p, DCPM.pin_an);
  connect(feedback.y, PI.u);
  connect(PI.y, signalVoltage.v);
end MotorControl;
```



A Modelica-based Tool Chain







Industrial Application I
Power Plant Start-up Optimization

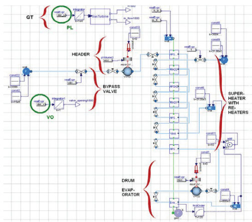
- Start-up optimization of combined cycle power plants
- Reduce start-up time
- Model-based optimization
- Siemens AG, LU, Modelon collaboration

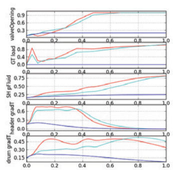
Continuous time states: **39**

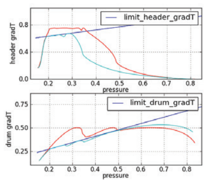
Scalar equations: **569**

Algebraic variables: **530**

NLP equations: **26824**







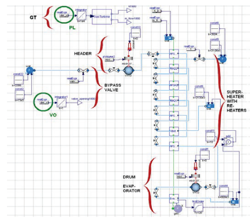
Industrial Application I
Power Plant Start-up Optimization

- ☺ Design-patterns from Modelica media model libraries applied to optimization-friendly models
- ☺ Intuitive high-level descriptions of dynamic optimization problem appreciated by users – a vehicle for communicating ideas

- ☹ Large effort to develop models suitable for optimization
- ☹ Scaling of problem significantly more challenging than in simulation
- ☹ Convergence and robustness of numerical algorithms

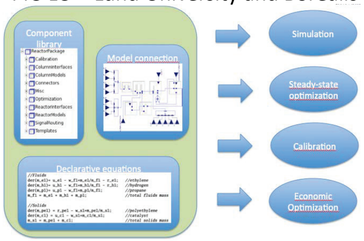
Lessons learnt

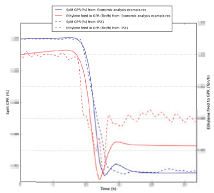
- Modeling for optimization is significantly different from modeling for simulation
- Numerical optimization algortihm is significantly less robust than simulation algorithm
- Scaling of problem and initial guesses have major impact



Industrial Application II
Grade Changes in Polyethylene Production

- Optimization of economics of polyethylene grade changes
- Model calibration to data
- Modeling with Modelica and Optimica
- Development of end-user GUI
- PIC-LU – Lund University and Borealis



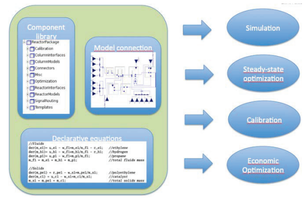


Industrial Application II
Grade Changes in Polyethylene Production

☺ Model reuse across different computations
☺ High-level model and optimization problem formulation enabled promoted focus on problem formulation
☺ Custom GUI in Python appreciated by end-users

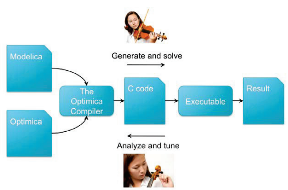
☹ Careful manual scaling of problem required for convergence
☹ Difficult to tailor collocation optimization formulation to problem description
☹ Non-standard economic cost difficult to handle

Lessons learnt
- Significant advantages from Modelica technology – same model used for steady-state, dynamic simulation, calibration and optimization
- Increased interaction with discretization sometimes important



Extension Example – Optimica

- High-level description of optimization problems
  - Steady-state
  - Dynamic
- Extension to Modelica
  - Optimization of physical models




```
optimization VDP_Opt(objective=cost(finalTime),
     startTime=0,
     finalTime(free=true, initialGuess=1))
  VDP vdp(u(free=true,initialGuess=0.0));
   Real cost (start=0);
equation
   der(cost) = 1;
constraint
   vdp.x1(finalTime) = 0;
   vdp.x2(finalTime) = 0;
   vdp.u >= -1; vdp.u <= 1;
end VDP_Opt;
```

$$\min_{u(t),\,p} \Psi(\bar{z}, p)$$

subject to the dynamic system

$$F(\dot{x}(t), x(t), y(t), u(t), p, t) = 0, \quad t \in [t_0, t_f]$$

and the constraints

$$c_{ineq}(x(t), y(t), u(t), p) \leq 0, \quad t \in [t_0, t_f]$$
$$c_{eq}(x(t), y(t), u(t), p) = 0, \quad t \in [t_0, t_f]$$
$$c^p_{ineq}(\bar{z}, p) \leq 0$$
$$c^p_{eq}(\bar{z}, p) = 0$$

where

$$\bar{z} = [x(t_1), \ldots, x(t_{N_t}), y(t_1), \ldots, y(t_{N_t}), u(t_1), \ldots, u(t_{N_t})]^T, \quad t_i \in [t_0, t_f]$$

Extension Example – Optimica

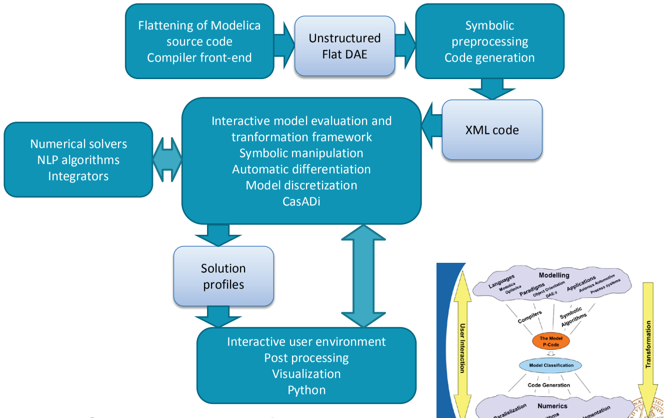
☺ High-level problem descriptions promote focus on formulation rather than encoding
☺ New users without optimization experience quickly gets up to speed
☺ Model reuse for different usages
☺ Automatic model transformation reduce user effort

☹ Tailoring of problem discretization difficult, but sometimes needed
☹ Power-users of dynamic optimization tools feel constrained

Lessons learnt
- High-level descriptions make optimization technology available to non-experts
- Automatic model transformation reduces design cycle times
- Modern compiler construction technology is accessible to non-experts (e.g., JastAdd)

```
optimization VDP_Opt(objective=cost(finalTime),
     startTime=0,
     finalTime(free=true, initialGuess=1))
  VDP vdp(u(free=true,initialGuess=0.0));
   Real cost (start=0);
equation
   der(cost) = 1;
constraint
   vdp.x1(finalTime) = 0;
   vdp.x2(finalTime) = 0;
   vdp.u >= -1; vdp.u <= 1;
end VDP_Opt;
```

Towards a vertically integrated toolchain

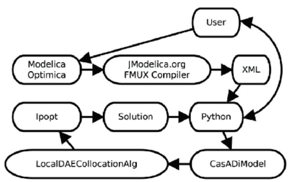


Interfacing Example –
Modelica, XML Models and CasADi

☺ Rapid prototyping with interactive model evaluation and transformation frameworks
☺ Flexibility to tailor model descretization to problem formulation
☺ Inspiration for future versions of Optimica

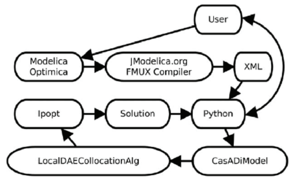
☹ Partial problem formulation in high-level format
☹ Some of the overview lost when parts of the problem is formulated in Modelica/Optimica some part is in scripting language

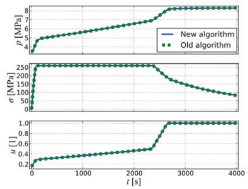
Lessons learnt
- Interactive model transformation powerful
- Symbolic model exchange format needed (standardization on-going)
- High performance and flexibility can be combined



Interfacing Example –
Modelica, XML Models and CasADi

- Replace C implementation of a collocation algorithm
- Intermediate symbolic model format in XML
- Decreased solution times by an order of magnitude
- Decreased implementation time by an order of magnitude
- Significantly increased flexibilty
- Tailoring to specific problems





	Off-line	On-line	Total	Iterations
New alg.	4.9	3.0	7.9	79
Old alg.	13.2	23.9	37.2	75

Challenges

- *How do we make advanced algorithms in systems design in general and in optimization in particular PhD-free?*
- *How do we combine declarative modeling languages with ideas from interactive model transformation/evaluation frameworks?*
- *How do we propagate consistent error/diagnostics through the tool chain?*
- *Open interfaces and interoperability, FMI and extensions*
- *Classify models applicable to different solution algorithms*

Conclusions

- In users' perception, current optimization algorithms for large-scale non-linear dynamic systems requires high level of expertise
- Very different cultures and best practices in simulation and optimization communities – expectation management
- Users sometimes need to/desire to to interact with both mathematical model and solution algorithm implementation
- Challenges in usability and robustness of numerical algorithms
- Challenges in vertically integrated tool chains – languages and open interfaces and tool decoupling

Thank you!

Questions, comments?

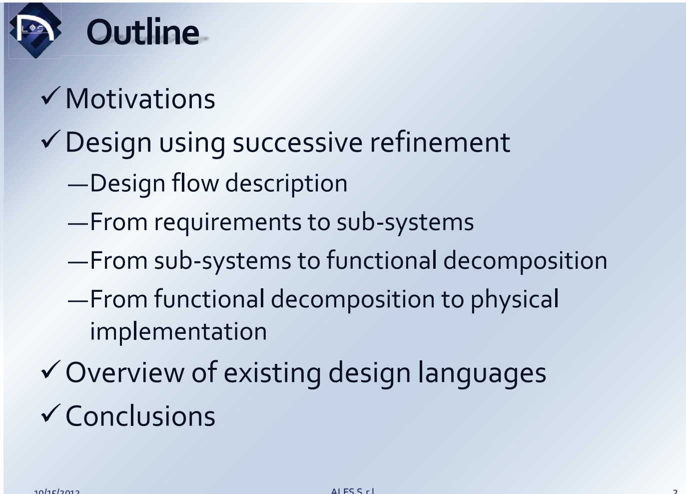
SYSTEM DESIGN – FROM REQUIREMENTS TO IMPLEMENTATION

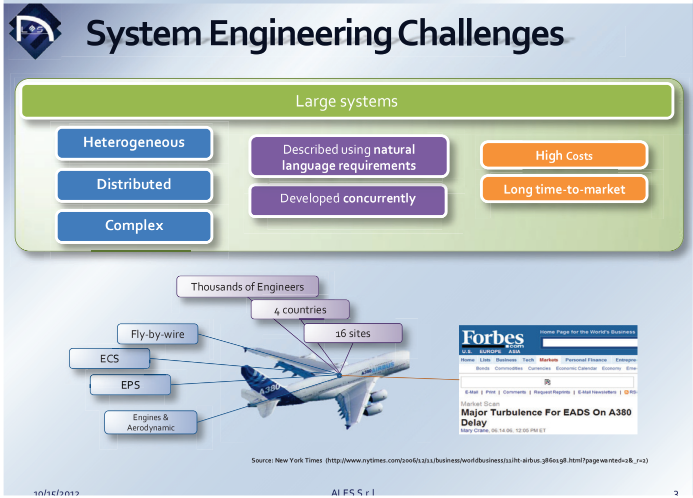
Alberto Ferrari, ALES S.r.l.

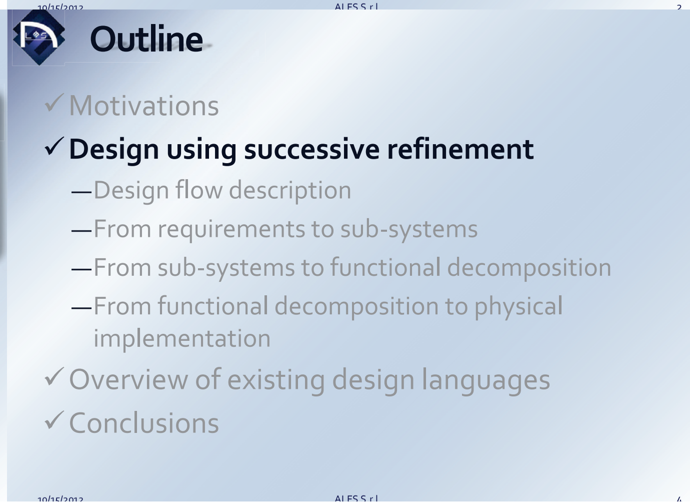
The design of cyber-physical systems by successive refinements starts from a set of requirements and incrementally adds design decisions till implementation is built. Equation based languages are essential to support with rigorousness these decisions and enable a formal exploration of the solution space.

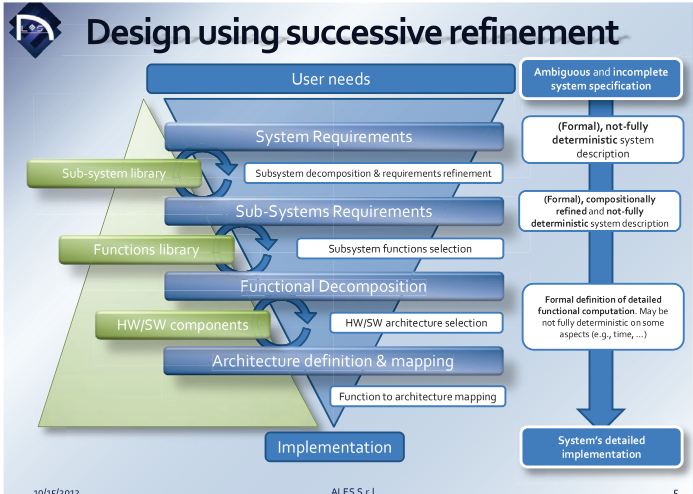
For real industrial cases, none of the current equation based language is capable of covering the entire design flow and different languages must be used. In this talk some of the current gaps and challenges will be described and partially addressed.


System Design: From
Requirements to Implementation
A.Ferrari
O.Ferrante, L.Mangeruca
Advanced Laboratory on Embedded Systems S.r.l.
A Research and Innovation Company

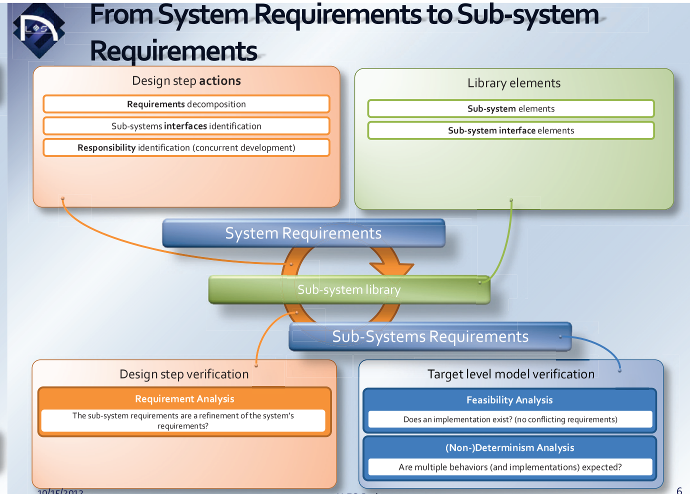

Outline
✓Motivations
✓Design using successive refinement
—Design flow description
—From requirements to sub-systems
—From sub-systems to functional decomposition
—From functional decomposition to physical implementation
✓Overview of existing design languages
✓Conclusions
2


System Engineering Challenges
Large systems
Heterogeneous
Distributed
Complex
Described using natural language requirements
Developed concurrently
High Costs
Long time-to-market
Thousands of Engineers
4 countries
16 sites
Fly-by-wire
ECS
EPS
Engines & Aerodynamic
Forbes
Major Turbulence For EADS On A380 Delay
Source: New York Times (http://www.nytimes.com/2006/12/11/business/worldbusiness/11iht-airbus.3860198.html?pagewanted=2&_r=2)
3

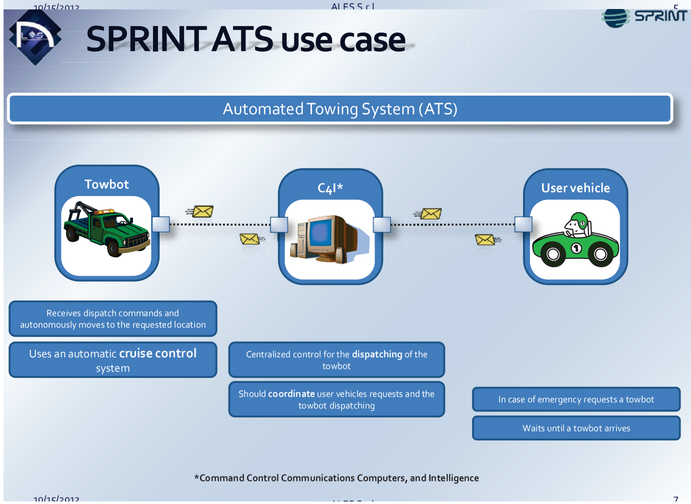

Outline
✓Motivations
✓Design using successive refinement
—Design flow description
—From requirements to sub-systems
—From sub-systems to functional decomposition
—From functional decomposition to physical implementation
✓Overview of existing design languages
✓Conclusions
4



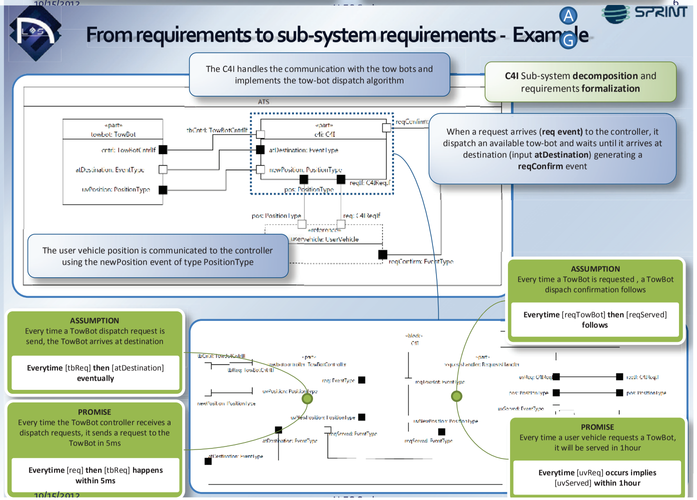
Design using successive refinement
User needs
Ambiguous and incomplete system specification
System Requirements
(Formal), not-fully deterministic system description
Sub-system library
Subsystem decomposition & requirements refinement
Sub-Systems Requirements
(Formal), compositionally refined and not-fully deterministic system description
Functions library
Subsystem functions selection
Functional Decomposition
Formal definition of detailed functional computation. May be not fully deterministic on some aspects (e.g., time, ...)
HW/SW components
HW/SW architecture selection
Architecture definition & mapping
Function to architecture mapping
Implementation
System's detailed implementation
10/15/2012
ALES S.r.l.
5



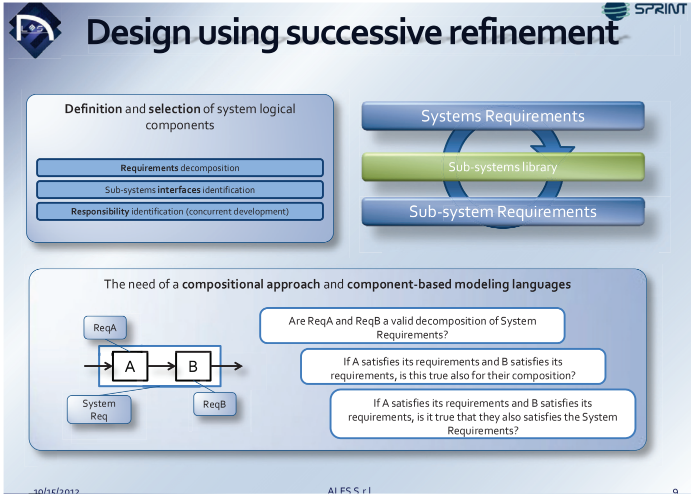
From System Requirements to Sub-system Requirements
Design step actions
Requirements decomposition
Sub-systems interfaces identification
Responsibility identification (concurrent development)
Library elements
Sub-system elements
Sub-system interface elements
System Requirements
Sub-system library
Sub-Systems Requirements
Design step verification
Requirement Analysis
The sub-system requirements are a refinement of the system's requirements?
Target level model verification
Feasibility Analysis
Does an implementation exist? (no conflicting requirements)
(Non-)Determinism Analysis
Are multiple behaviors (and implementations) expected?
10/15/2012
6

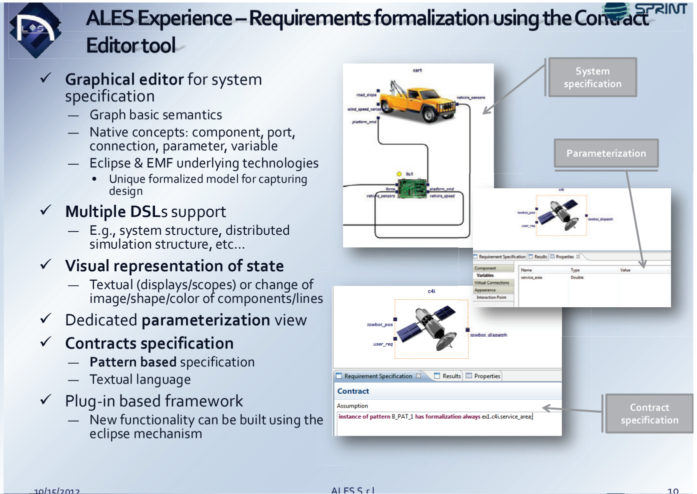


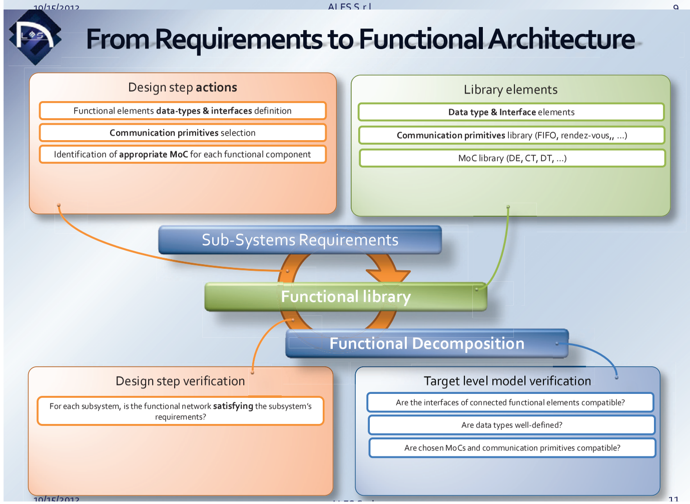
SPRINT ATS use case
SPRINT
Automated Towing System (ATS)
Towbot
C4I*
User vehicle
Receives dispatch commands and autonomously moves to the requested location
Uses an automatic cruise control system
Centralized control for the dispatching of the towbot
Should coordinate user vehicles requests and the towbot dispatching
In case of emergency requests a towbot
Waits until a towbot arrives
*Command Control Communications Computers, and Intelligence
10/15/2012
7

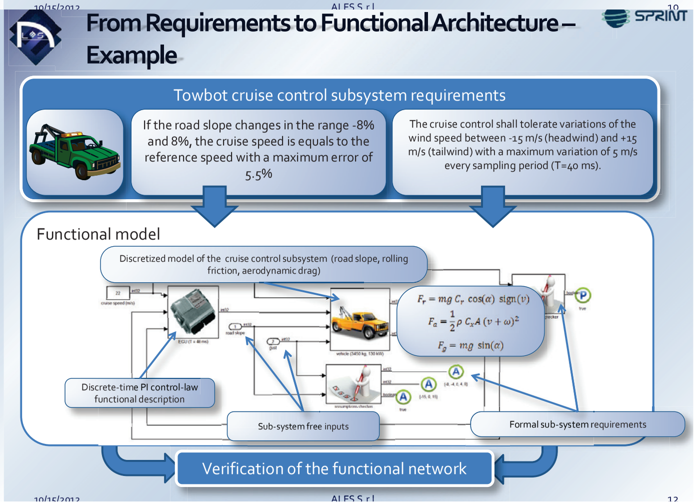


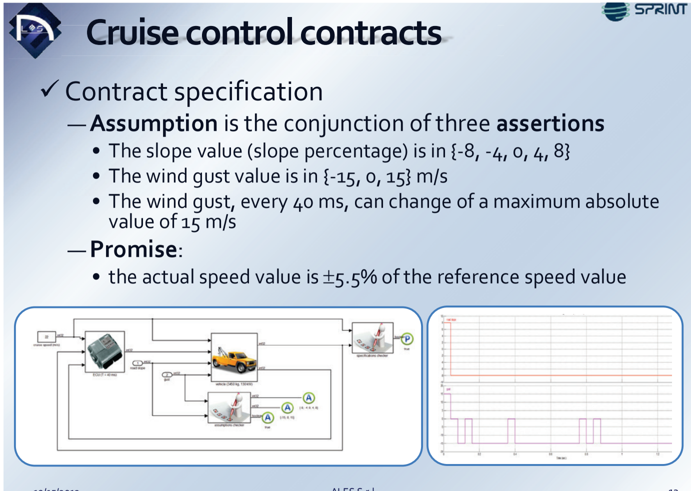
From requirements to sub-system requirements - Example
SPRINT
The C4I handles the communication with the tow bots and implements the tow-bot dispatch algorithm
C4I Sub-system decomposition and requirements formalization
When a request arrives (req event) to the controller, it dispatch an available tow-bot and waits until it arrives at destination (input atDestination) generating a reqConfirm event
The user vehicle position is communicated to the controller using the newPosition event of type PositionType
ASSUMPTION
Every time a TowBot is requested , a TowBot dispach confirmation follows
Everytime [reqTowBot] then [reqServed] follows
ASSUMPTION
Every time a TowBot dispatch request is send, the TowBot arrives at destination
Everytime [tbReq] then [atDestination] eventually
PROMISE
Every time the TowBot controller receives a dispatch requests, it sends a request to the TowBot in 5ms
Everytime [req] then [tbReq] happens within 5ms
PROMISE
Every time a user vehicle requests a TowBot, it will be served in 1hour
Everytime [uvReq] occurs implies [uvServed] within 1hour
10/15/2012

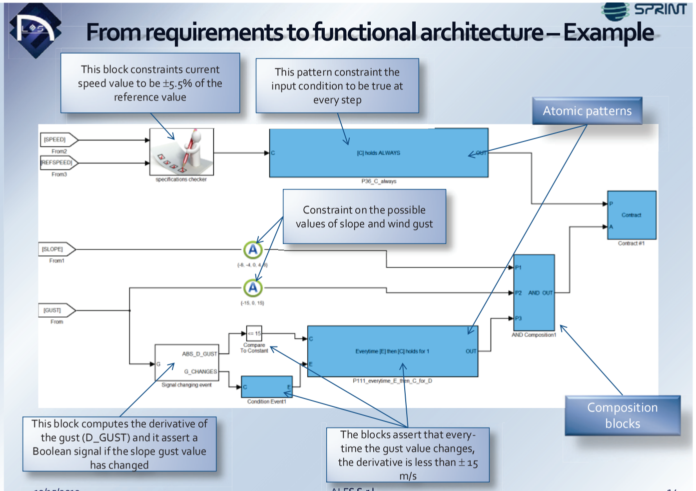

Design using successive refinement
SPRINT
Definition and selection of system logical components
Requirements decomposition
Sub-systems interfaces identification
Responsibility identification (concurrent development)
Systems Requirements
Sub-systems library
Sub-system Requirements
The need of a compositional approach and component-based modeling languages
ReqA
A
B
System Req
ReqB
Are ReqA and ReqB a valid decomposition of System Requirements?
If A satisfies its requirements and B satisfies its requirements, is this true also for their composition?
If A satisfies its requirements and B satisfies its requirements, is it true that they also satisfies the System Requirements?
9

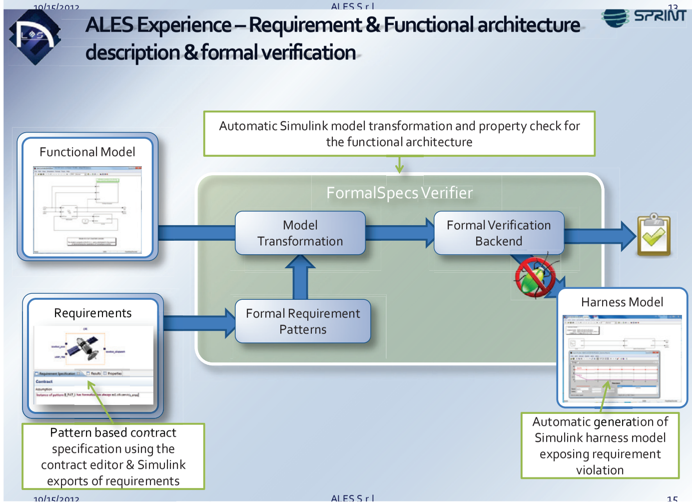

ALES Experience – Requirements formalization using the Contract Editor tool
SPRINT
✓ Graphical editor for system specification
— Graph basic semantics
— Native concepts: component, port, connection, parameter, variable
— Eclipse & EMF underlying technologies
• Unique formalized model for capturing design
✓ Multiple DSLs support
— E.g., system structure, distributed simulation structure, etc…
✓ Visual representation of state
— Textual (displays/scopes) or change of image/shape/color of components/lines
✓ Dedicated parameterization view
✓ Contracts specification
— Pattern based specification
— Textual language
✓ Plug-in based framework
— New functionality can be built using the eclipse mechanism
System specification
Parameterization
Contract specification
Contract
Assumption
10

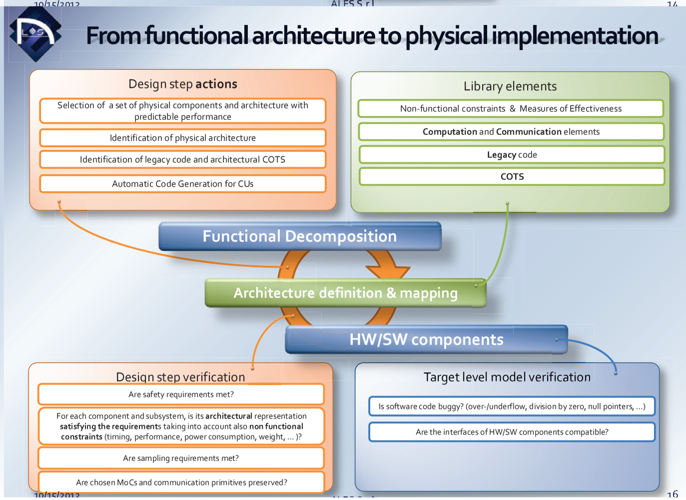

From Requirements to Functional Architecture
Design step actions
Functional elements data-types & interfaces definition
Communication primitives selection
Identification of appropriate MoC for each functional component
Library elements
Data type & Interface elements
Communication primitives library (FIFO, rendez-vous,, …)
MoC library (DE, CT, DT, …)
Sub-Systems Requirements
Functional library
Functional Decomposition
Design step verification
For each subsystem, is the functional network satisfying the subsystem's requirements?
Target level model verification
Are the interfaces of connected functional elements compatible?
Are data types well-defined?
Are chosen MoCs and communication primitives compatible?
11

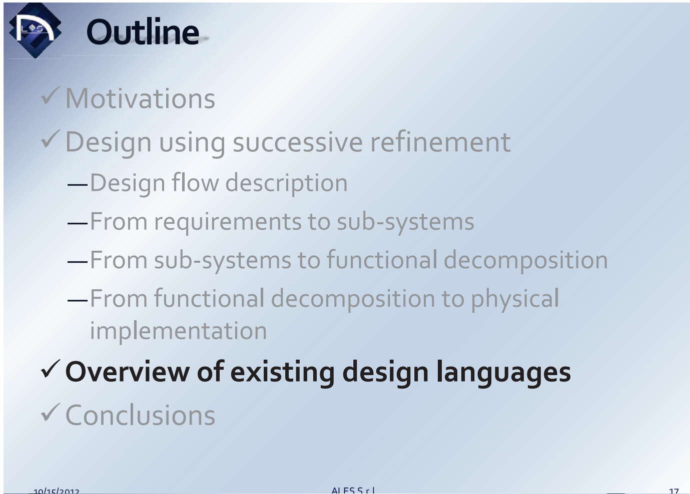

From Requirements to Functional Architecture – Example
SPRINT
Towbot cruise control subsystem requirements
If the road slope changes in the range -8% and 8%, the cruise speed is equals to the reference speed with a maximum error of 5.5%
The cruise control shall tolerate variations of the wind speed between -15 m/s (headwind) and +15 m/s (tailwind) with a maximum variation of 5 m/s every sampling period (T=40 ms).
Functional model
Discretized model of the cruise control subsystem (road slope, rolling friction, aerodynamic drag)
$F_r = mg\,C_r\cos(\alpha)\,\mathrm{sign}(v)$
$F_a = \frac{1}{2}\rho\,C_x A\,(v+\omega)^2$
$F_g = mg\,\sin(\alpha)$
Discrete-time PI control-law functional description
Sub-system free inputs
Formal sub-system requirements
Verification of the functional network
12

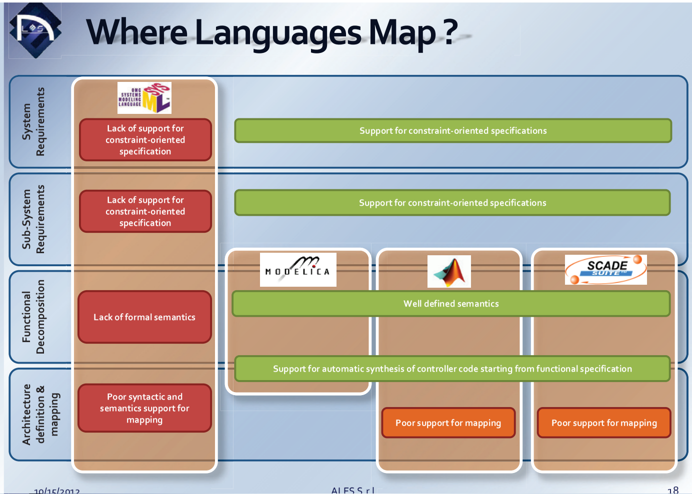

Cruise control contracts
SPRINT
✓ Contract specification
— Assumption is the conjunction of three assertions
• The slope value (slope percentage) is in {-8, -4, 0, 4, 8}
• The wind gust value is in {-15, 0, 15} m/s
• The wind gust, every 40 ms, can change of a maximum absolute value of 15 m/s
— Promise:
• the actual speed value is ±5.5% of the reference speed value
10/15/2012
ALES S.r.l.
13

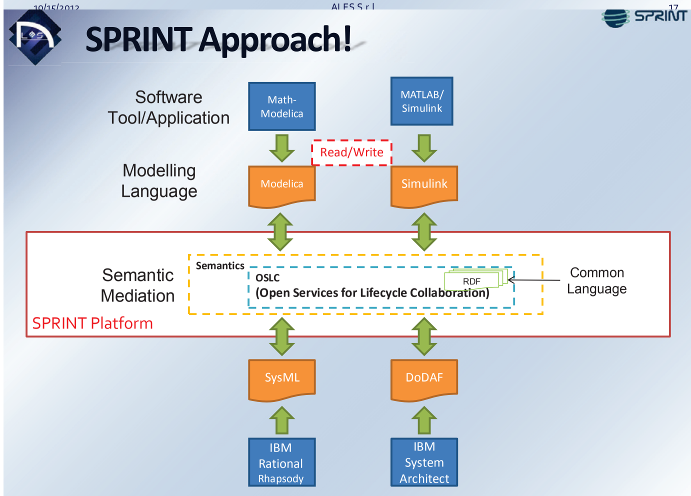

From requirements to functional architecture – Example
SPRINT
This block constraints current speed value to be ±5.5% of the reference value
This pattern constraint the input condition to be true at every step
Atomic patterns
Constraint on the possible values of slope and wind gust
This block computes the derivative of the gust (D_GUST) and it assert a Boolean signal if the slope gust value has changed
The blocks assert that every-time the gust value changes, the derivative is less than ±15 m/s
Composition blocks
10/15/2012
ALES S.r.l.
14

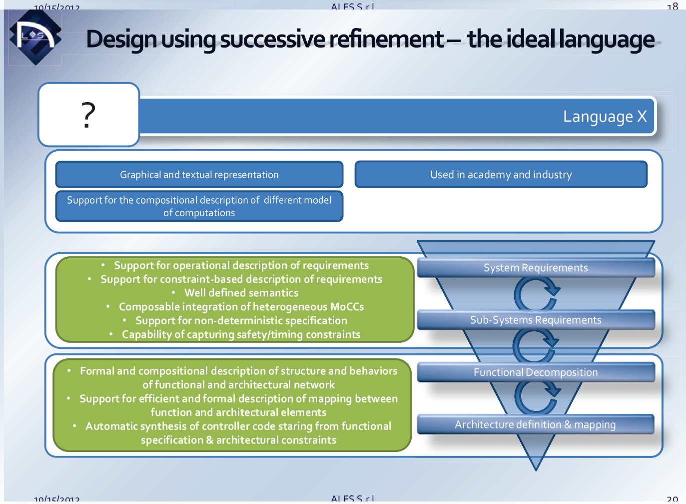

ALES Experience – Requirement & Functional architecture description & formal verification
SPRINT
Automatic Simulink model transformation and property check for the functional architecture
Functional Model
FormalSpecs Verifier
Model Transformation
Formal Verification Backend
Requirements
Formal Requirement Patterns
Harness Model
Pattern based contract specification using the contract editor & Simulink exports of requirements
Automatic generation of Simulink harness model exposing requirement violation
10/15/2012
ALES S.r.l.
15


From functional architecture to physical implementation
Design step actions
Selection of a set of physical components and architecture with predictable performance
Identification of physical architecture
Identification of legacy code and architectural COTS
Automatic Code Generation for CUs
Library elements
Non-functional constraints & Measures of Effectiveness
Computation and Communication elements
Legacy code
COTS
Functional Decomposition
Architecture definition & mapping
HW/SW components
Design step verification
Are safety requirements met?
For each component and subsystem, is its architectural representation satisfying the requirements taking into account also non functional constraints (timing, performance, power consumption, weight, ...)?
Are sampling requirements met?
Are chosen MoCs and communication primitives preserved?
Target level model verification
Is software code buggy? (over-/underflow, division by zero, null pointers, ...)
Are the interfaces of HW/SW components compatible?
10/15/2012
16


Outline
Motivations
Design using successive refinement
—Design flow description
—From requirements to sub-systems
—From sub-systems to functional decomposition
—From functional decomposition to physical implementation
Overview of existing design languages
Conclusions
10/15/2012
ALES S.r.l
17


Where Languages Map ?
System Requirements
Sub-System Requirements
Functional Decomposition
Architecture definition & mapping
Lack of support for constraint-oriented specification
Lack of support for constraint-oriented specification
Lack of formal semantics
Poor syntactic and semantics support for mapping
Support for constraint-oriented specifications
Support for constraint-oriented specifications
MODELICA
SCADE SUITE
Well defined semantics
Support for automatic synthesis of controller code starting from functional specification
Poor support for mapping
Poor support for mapping
10/15/2012
ALES S.r.l
18


SPRINT Approach!
SPRINT
Software Tool/Application
Modelling Language
Semantic Mediation
SPRINT Platform
Math-Modelica
MATLAB/ Simulink
Read/Write
Modelica
Simulink
Semantics
OSLC
(Open Services for Lifecycle Collaboration)
RDF
Common Language
SysML
DoDAF
IBM Rational Rhapsody
IBM System Architect


Design using successive refinement – the ideal language
?
Language X
Graphical and textual representation
Used in academy and industry
Support for the compositional description of different model of computations
Support for operational description of requirements
Support for constraint-based description of requirements
Well defined semantics
Composable integration of heterogeneous MoCCs
Support for non-deterministic specification
Capability of capturing safety/timing constraints
Formal and compositional description of structure and behaviors of functional and architectural network
Support for efficient and formal description of mapping between function and architectural elements
Automatic synthesis of controller code staring from functional specification & architectural constraints
System Requirements
Sub-Systems Requirements
Functional Decomposition
Architecture definition & mapping
10/15/2012
ALES S.r.l
20

Conclusion

✓ Summary

— Design flow using successive refinement

- From requirements to sub-system
- From sub-system to functional architecture
- From functional architecture to physical implementation

— Equation-based language

- Overview
- Limitations

SYNCHRONOUS CONTROL AND STATE MACHINES IN MODELICA

Hilding Elmqvist, Dassault Systèmes AB

The scope of Modelica has been extended from a language primarily intended for physical systems modeling to modeling of complete systems by allowing the modeling of control systems and by enabling automatic code generation for embedded systems. Much focus has been given to safe constructs and intuitive and well-defined semantics.

The presentation will describe the fundamental synchronous language primitives introduced for increased correctness of control systems implementation. The approach is based on associating clocks to the variable types. Special operators are needed when accessing variables of another clock. This enables clock inference and increased correctness of the code since many more checks can be done during translation. Furthermore, the sampling period of a clocked partition needs to be defined only at one place (either in absolute time or relatively to other clocked partitions). The principles of partitioning a system model into different clocks (continuous, periodic, non-periodic, multi-rate) will be explained.

The new language elements follow the synchronous approach. They are based on the clock calculus and inference system of Lucid Synchrone. However, the Modelica approach also uses multi-rate periodic clocks based on rational arithmetic and also non-periodic and event based clocks are supported.

Parallel and hierarchical state machines will be introduced including submodels within states. The supporting Modelica library will also be introduced.

Synchronous Control and State Machines in Modelica

Hilding Elmqvist
Dassault Systèmes

Sven Erik Mattsson, Fabien Gaucher, Francois Dupont
Dassault Systèmes

Martin Otter, Bernhard Thiele
DLR

Content

- Introduction
- Synchronous Features of Modelica
  - Synchronous Operators
  - Base-clock and Sub-clock Partitioning
- Modelica_Synchronous library
- State Machines
- Conclusions

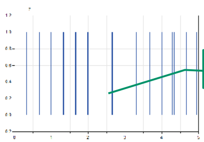
Slide 2

Introduction

- Why synchronous features in Modelica 3.3?

```
model Asynchronous_Modelica32
  Real x(start=0,fixed=true),
    y(start=0,fixed=true), z;
equation
  when sample(0,0.33) then
    x = pre(x)+1;
  end when;
  when sample(0,1/3) then
    y = pre(y)+1;
  end when;
  z = x-y;
end Asynchronous_Modelica32;
```

Implicit hold



z = x-y

```
model Asynchronous_Modelica33
  Real x(start=0,fixed=true),
    y(start=0,fixed=true), z;
equation
  when Clock(0.33) then
    x = previous(x)+1;
  end when;
  when Clock(1,3) then
    y = previous(y)+1;
  end when;
  z = x-y;
end Asynchronous_Modelica33;
```

Rational number 1/3

x and y must have the same clock

```
A subclock partition includes clocks that cannot be deduced to be equal.
Clock(1, 3)
Clock(0.33)
appears in the partition
when Clock_1 then
y = previous(y)+1;
end when;

when Clock_0 then
x = previous(x)+1;
end when;

z = x-y;
```

- Error Diagnostics for safer systems!

Slide 3

Introduction

- Scope of Modelica extended
- Covers complete system descriptions including controllers

- Clocked semantics
- Clock associated with variable type and inferred
- For increased correctness
- Based on ideas from Lucid Synchrone and other synchronous languages
- Extended with multi-rate periodic clocks, varying interval clocks and Boolean clocks

Slide 4

Synchronous Features of Modelica

- Plant and Controller Partitioning
- Boundaries between continuous-time and discrete-time equations defined by operators.
- sample(): samples a continuous-time variable and returns a clocked discrete-time expression
- hold(): converts from clocked discrete-time to continuous-time by holding the value between clock ticks
- sample operator may take a Clock argument to define when sampling should occur

MODELICA Slide 5

Mass with Spring Damper

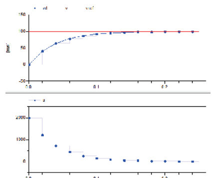
- Consider a continuous-time model

```
partial model MassWithSpringDamper
  parameter Modelica.SIunits.Mass m=1;
  parameter Modelica.SIunits.TranslationalSpringConstant k=1;
  parameter Modelica.SIunits.TranslationalDampingConstant d=0.1;
  Modelica.SIunits.Position x(start=1,fixed=true) "Position";
  Modelica.SIunits.Velocity v(start=0,fixed=true) "Velocity";
  Modelica.SIunits.Force f "Force";
equation
  der(x) = v;
  m*der(v) = f - k*x - d*v;
end MassWithSpringDamper;
```

MODELICA Slide 6

Synchronous Controller

- Discrete-time controller



```
model SpeedControl
  extends MassWithSpringDamper;
  parameter Real K = 20 "Gain of speed P controller";
  parameter Modelica.SIunits.Velocity vref = 100 "Speed ref.";
  discrete Real vd;
  discrete Real u(start=0);
equation
  // speed sensor
  vd = sample(v, Clock(0.01));

  // P controller for speed
  u = K*(vref-vd);

  // force actuator
  f = hold(u);
end SpeedControl;
```

Sample continuous velocity v with periodic Clock with period=0.01

The clock of the equation is inferred to be the same as for the variable vd which is the result of sample()

Hold discrete variable u between clock ticks

MODELICA Slide 7

Discrete-time State Variables

- Operator previous() is used to access the value at the previous clock tick (cf pre() in Modelica 3.2)
- Introduces discrete state variable
- Initial value needed

- interval() is used to inquire the actual interval of a clock

MODELICA Slide 8

Base-clocks and Sub-clocks

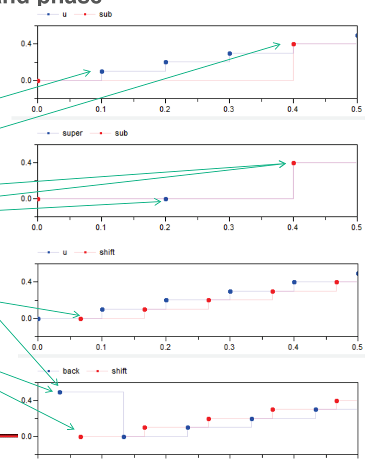
- A Modelica model will typically have several controllers for different parts of the plant.
- Such controllers might not need synchronization and can have different *base clocks*.
- Equations belonging to different base clocks can be implemented by asynchronous tasks of the used operating system.
- It is also possible to introduce sub-clocks that tick a certain factor slower than the base clock.
- Such sub-clocks are perfectly synchronized with the base clock, i.e. the definitions and uses of a variable are sorted in such a way that when sub-clocks are activated at the same clock tick, then the definition is evaluated before all the uses.
- New base type, Clock:

```
Clock cControl = Clock(0.01);
Clock cOuter = subSample(cControl, 5);
```

MODELICA Slide 9

Sub and super sampling and phase

```
model SynchronousOperators
  Real u;

  Real sub;
  Real super;

  Real shift(start=0.5);
  Real back;
equation
  u = sample(time, Clock(0.1));

  sub = subSample(u, 4);
  super = superSample(sub, 2);

  shift = shiftSample(u, 2, 3);
  back = backSample(shift, 1, 3);
end SynchronousOperators;
```



MODELICA

Exact Periodic Clocks

- Clocks defined by Real number period are not synchronized:

```
Clock c1 = Clock(0.1);

Clock c2 = superSample(c1,3);

Clock c3 = Clock(0.1/3);  // Not synchronized with c2
```

- Clocks defined by rational number period are synchronized:

```
Clock c1 = Clock(1,10);         // period = 1/10

Clock c2 = superSample(c1,3);  // period = 1/30

Clock c3 = Clock(1,30);         // period = 1/30
```

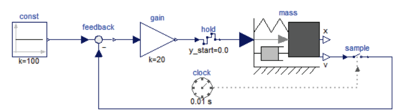
MODELICA Slide 11

Modelica_Synchronous library

- Synchronous language elements of Modelica 3.3 are "low level":

```
// speed sensor
vd = sample(v, Clock(0.01));

// P controller for speed
u = K*(vref-vd);

// force actuator
f = hold(u);
```

- Modelica_Synchronous library developed to access language elements in a convenient way graphically:



MODELICA Slide 12

Blocks that generate clock signals

periodicRealClock

0.02 s

Generates a periodic clock with a Real period

```
  parameter Modelica.SIunits.Time period;
  ClockOutput y;
equation
  y = Clock(period);
```

periodicExactClock

20 ms

Generates a periodic clock as an integer multiple of a resolution (defined by an enumeration).

y (year)
d (day)
h (hour)
min (minutes)
s (seconds)
ms (milli seconds)
us (micro seconds)
ns (nano seconds)

Code for 20 ms period:

```
y = superSample(Clock(20), 1000);
```

Clock with period 20 s — super-sample clock with 1000

period = 20 / 1000 = 20 ms

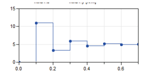
eventClock

Generates an event clock: The clock ticks whenever the continuous-time Boolean input changes from false to true.

```
y = Clock(u);
```

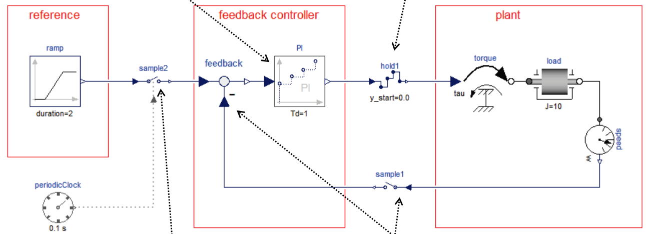
MODELICA Slide 13

Sample and Hold



Discrete-time PI controller

Holds a clocked signal and generates a continuous-time signal. Before the first clock tick, the continuous-time output y is set to parameter y_start `y = hold(u);`



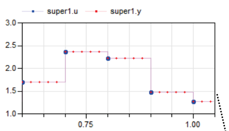
Samples a continuous-time signal and generates a clocked signal.

Purely algebraic block from Modelica.Blocks.Math

`y = sample(u, clock);` `y = sample(u);`

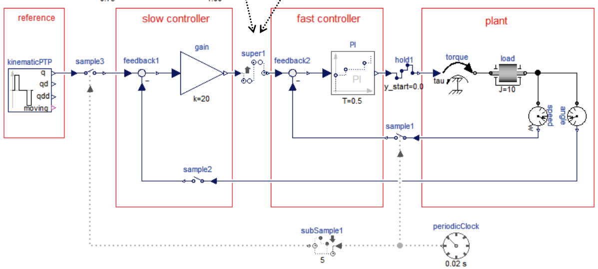
MODELICA Slide 14

Sub- and Super-Sampling

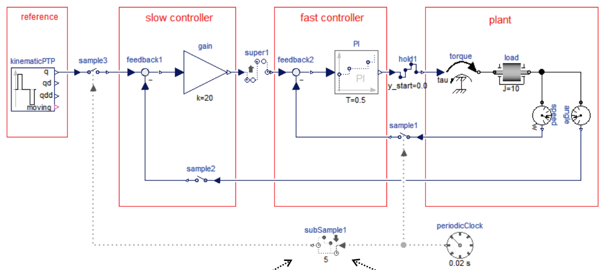


Defines that the output signal is an integer factor faster as the input signal, using a "hold" semantics for the signal. By default, this factor is inferred. It can also be defined explicitly.

`y = superSample(u);`

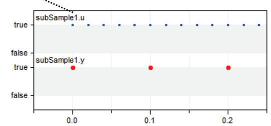


MODELICA Slide 15



Defines that the output signal is an integer factor slower as the input signal, picking every n-th value of the input.

`y = subSample(u,factor);`

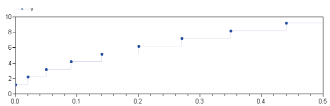


MODELICA Slide 16

Varying Interval Clocks

- The first argument of Clock(ticks, resolution) may be time dependent
- Resolution must not be time dependent
- Allowing varying interval clocks
- Can be sub and super sampled and phased

```
model VaryingClock
  Integer nextInterval(start=1);
  Clock c = Clock(nextInterval, 100);
  Real  v(start=0.2);
equation
  when c then
    nextInterval = previous(nextInterval) + 1;
    v = previous(v) + 1;
  end when;
end VaryingClock;
```

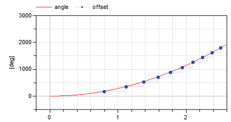


Slide 17

Boolean Clocks

- Possible to define clocks that tick when a Boolean expression changes from false to true.
- Assume that a clock shall tick whenever the shaft of a drive train passes 180º.

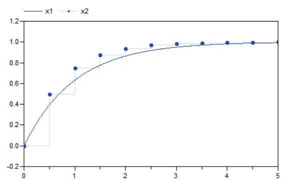
```
model BooleanClock
  Modelica.SIunits.Angle angle(start=0,fixed=true);
  Modelica.SIunits.AngularVelocity w(start=0,fixed=true);
  Modelica.SIunits.Torque tau=10;
  parameter Modelica.SIunits.Inertia J=1;
  Modelica.SIunits.Angle offset;
equation
  w = der(angle);
  J*der(w) = tau;
  when Clock(angle >= hold(offset)+Modelica.Constants.pi) then
    offset = sample(angle);
  end when;
end BooleanClock;
```



Slide 18

Discretized Continuous Time

- Possible to convert continuous-time partitions to discrete-time
- A powerful feature since in many cases it is no longer necessary to manually implement discrete-time components
- Build-up a inverse plant model or controller with continuous-time components and then sample the input signals and hold the output signals.
- And associate a solverMethod with the Clock.




```
model Discretized
  Real x1(start=0,fixed=true);
  Real x2(start=0,fixed=true);
equation
  der(x1) = -x1 + 1;

  der(x2) = -x2 + sample(1, Clock(Clock(0.5), solverMethod="ExplicitEuler"));
end Discretized;
```

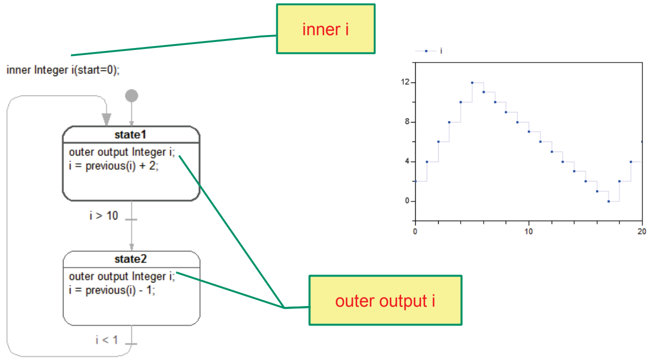
Slide 19

State Machines

- Modelica extended to allow modeling of control systems
- Any block without continuous-time equations or algorithms can be a **state** of a state machine.
- Transitions between such blocks are represented by a new kind of connections associated with transition conditions.
- The complete semantics is described using only 13 Modelica equations.
- A cluster of block instances at the same hierarchical level which are coupled by **transition** equations constitutes a state machine.
- All parts of a state machine must have the same clock. *(We will work on removing this restriction ,allowing mixing clocks and allowing continuous equations, in future Modelica versions.)*
- One and only one instance in each state machine must be marked as initial by appearing in an **initialState** equation.

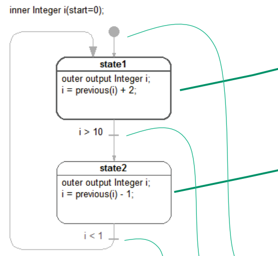
Slide 20

A Simple State Machine



Slide 21

A Simple State Machine – Modelica Text Representation




```
model StateMachine1
  inner Integer i(start=0);
  block State1
    outer output Integer i;
  equation
    i = previous(i) + 2;
  end State1;
  State1 state1;
  block State2
    outer output Integer i;
  equation
    i = previous(i) - 1;
  end State2;
  State2 state2;
equation
  initialState(state1);
  transition(state1, state2, i > 10, immediate=false);
  transition(state2, state1, i < 1, immediate=false);
end StateMachine1;
```

Slide 22

Merging Variable Definitions

- An **outer output** declaration means that the equations have access to the corresponding variable declared **inner**.
- Needed to maintain the single assignment rule.
- Multiple definitions of such outer variables in different mutually exclusive states of one state machine need to be merged.
- In each state, the outer output variables (v_j) are solved for ($expr_j$) and, for each such variable, a single definition is automatically formed:
- v := **if** activeState($state_1$) **then** $expr_1$
  elseif activeState($state_2$) **then** $expr_2$
  elseif … **else** last(v)

- **last**() is a special internal semantic operator returning its input. It is just used to mark for the sorting that the incidence of its argument should be ignored.
- A start value must be given to the variable if not assigned in the initial state.
- Such a newly created assignment equation might be merged on higher levels in nested state machines.

Slide 23

Defining a State machine

transition(from, to, condition, immediate, reset, synchronize, priority)

- This operator defines a transition from instance "from" to instance "to". The "from" and "to" instances become states of a state machine.
- The transition fires when condition = true if immediate = true (this is called an "immediate transition") or previous(condition) when immediate = false (this is called a "delayed transition").
- If reset = true, the states of the target state are reinitialized, i.e. state machines are restarted in initial state and state variables are reset to their start values.
- If synchronize = true, the transition is disabled until all state machines within the from-state have reached the final states, i.e. states without outgoing transitions.

- "from" and "to" are block instances and "condition" is a Boolean expression.
- "immediate", "reset", and "synchronize" (optional) are of type Boolean, have parametric variability and a default of true, true, false respectively.
- "priority" (optional) is of type Integer, has parametric variability and a default of 1 (highest priority). Defines the priority of firing when several transitions could fire.

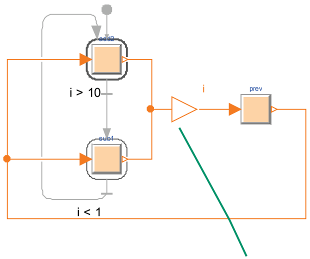
initialState(state)

- The argument "state" is the block instance that is defined to be the initial state of a state machine.

Slide 24

Conditional Data Flows

- Alternative to using **outer output** variables is to use conditional data flows.




```
block Increment
  extends Modelica.Blocks.Interfaces.PartialIntegerSISO;
  parameter Integer increment;
equation
  y = u + increment;
end Increment;

block Prev
  extends Modelica.Blocks.Interfaces.PartialIntegerSISO;
equation
  y = previous(u);
end Prev;
```

Merge of Conditional Data Flows

- It is possible to connect several outputs to inputs if all the outputs come from states of the same state machine.

  $u_1 = u_2 = \ldots = y_1 = y_2 = \ldots$

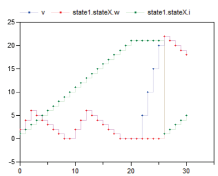
  with u_i inputs and y_i outputs.
- Let variable v represent the signal flow and rewrite the equation above as a set of equations for u_i and a set of assignment equations for v:
- v := **if** activeState($state_1$) **then** y_1 **else** last(v);
  v := **if** activeState($state_2$) **then** y_2 **else** last(v);
  …
  $u_1 = v$
  $u_2 = v$
  …
- The merge of the definitions of v is then made as described previously:

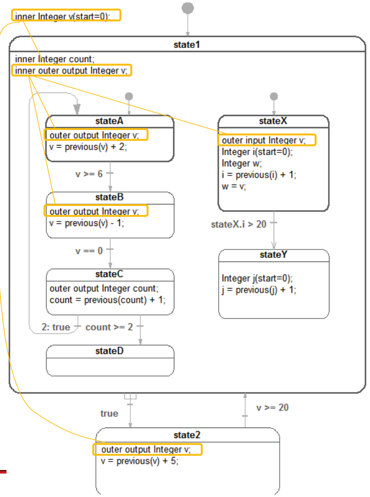
  v = **if** activeState($state_1$) **then** y_1
  elseif activeState($state_2$) **then** y_2
  elseif … **else** last(v)
  …

MODELICA Slide 25

Hierarchical State Machine Example

- stateA declares v as 'outer output'.
- state1 is on an intermediate level and declares v as 'inner outer output', i.e. matches lower level outer v by being inner and also matches higher level inner v by being outer.
- The top level declares v as inner and gives the start value.



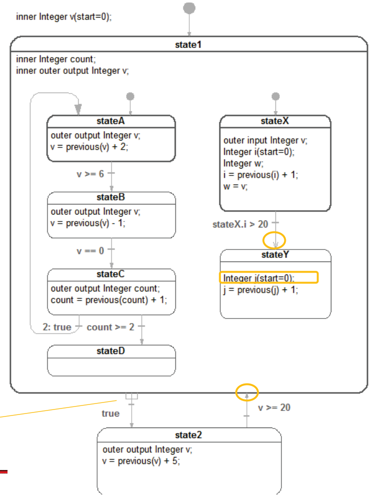


MODELICA

MODELICA Slide 26

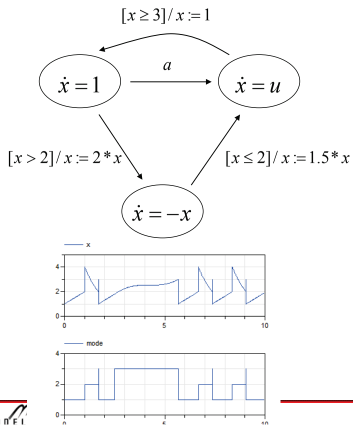
Reset and Synchronize

- count is defined with a start value in state1. It is reset when a reset transition (v>=20) is made to state1.
- stateY declares a local counter j. It is reset at start and as a consequence of the reset transition (v>=20) from state2 to state1.
- The reset of j is deferred until stateY is entered by transition (stateX.i>20) although this transition is not a reset transition.
- Synchronizing the exit from the two parallel state machines of state1 is done by using a synchronized transition.



MODELICA

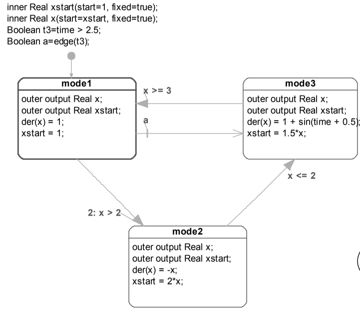
Hybrid Automata (Modelica 3.2-, 2006)

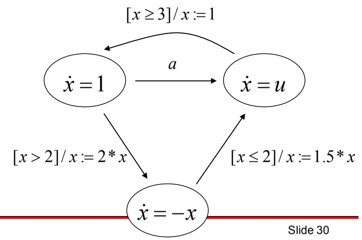



```
model Hybrid1
  Real x(start=1);
  Integer mode(start=1);
  Boolean a=time>2.5;
equation
  if mode == 1 then
    der(x) = 1;
  elseif mode==2 then
    der(x) = -x;
  else
    der(x) = 1+sin(time+0.5);
  end if;

algorithm
  when x>2 and mode==1 then
    mode :=2;
    reinit(x, 2*x);
  elsewhen edge(a) and mode==1 then
    mode :=3;
  elsewhen x<=2 and mode==2 then
    mode :=3;
    reinit(x, 1.5*x);
  elsewhen x>=3 and mode==3 then
    mode :=1;
    reinit(x, 1);
  end when;
end Hybrid1;
```

Slide 29

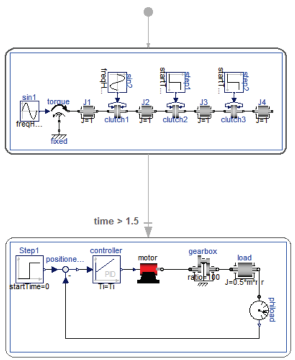
Hybrid Automata with Modelica 3.3+ (prototype)





Slide 30

Acausal Models in States – Modelica 3.3+

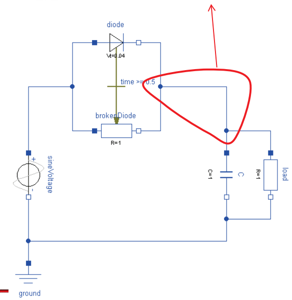


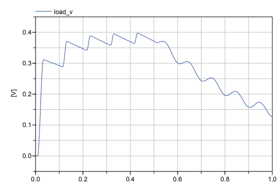
- The equations of each state is guarded by the activity condition
- Should time variable be stopped when not active?
- Should time be reset locally in state by a reset transition?
- Special Boolean operator exception() to detect a problem in one model and transition to another model

Slide 31

Multiple Acasual Connections

- // C_p_i+brokenDiode_n_i+diode_n_i+load_p_i = 0.0;
- Replaced by:
- C_p_i +
  (if activeState(brokenDiode) then brokenDiode_n_i else 0) +
  (if activeState(diode) then diode_n_i else 0) +
  load_p_i = 0.0;





Slide 32

Conclusions

- We have introduced synchronous features in Modelica 3.3.
- For a discrete-time variable, its clock is associated with the variable type and inferencing is supported.
- Special operators have to be used to convert between clocks.
- This gives an additional safety since correct synchronization is guaranteed by the compiler.

- We have described how state machines can be modeled in Modelica 3.3.
- Instances of blocks connected by transitions with one such block marked as an initial state constitute a state machine.
- Hierarchical state machines can be defined with reset or resume semantics, when re-entering a previously executed state.
- Parallel sub-state machines can be synchronized when they reached their final states.
- Special merge semantics have been defined for multiple outer output definitions in mutually exclusive states as well as conditional data flows.

EXTENSIBLE PROGRAMMING AND MODELING LANGUAGES

Eric Van Wyk, University of Minnesota

Extensible programming and modeling languages allow their users to import new features into their language. These may be new syntax (notations), new semantics (e.g. analysis for additional error checking), new optimizations, and new translations packaged as language extensions. Ideally, programmers and engineers with no knowledge of language design or implementation can direct tools to compose a "host" language with their chosen set of language extensions resulting in a custom translator or compiler for their extended language. To achieve this goal, languages and extensions are specified declaratively using context free grammars and attribute grammars. We describe a set of tools for generating translators and compilers from these specifications and a set of analyses that language extensions designers can use to verify that the composition of their extension and other similarly verified, independently developed, extensions will work as desired with the host language. These analyses ensure that the generated LR parser will be deterministic with no conflicts and that the attribute grammar will be complete, that is, has equations defining all needed attributes. Thus, the user is assured that their chosen language extensions will all work together. Example extensions to Java, C, Lustre, and Modelica will be discussed.

Extensible Programming and Modeling Languages

Ted Kaminski, Yogesh Mali, August Schwerdfeger
and *Eric Van Wyk*

University of Minnesota

September 20, 2012, Lund, Sweden

- Languages are not monolithic.
- But most language tools primarily support monolithic design and implementation.

Extensible Language Frameworks — ableP

- add features to a "host" language — Promela
- new language constructs - their syntax and semantics
  - `select (altitude: 1000 .. 10000);`
  - `select (altitude: 1000 .. 10000 step 100);`
  - `select (altQuality: High, Med, Low);`
  - DTSpin constructs: `timer t; t = 1; expire(t);`
- new semantics of existing constructs
  semantic analysis, translations to new target languages, ...
  - type checking
  - advanced ETCH-style type inference and checking

Various means for extending Promela

- `select (v: 1 .. 10)` added in SPIN version 6.
- DTSPIN features
  - as CPP macros — lightweight
  - or modifying the SPIN implementation — heavyweight
- ETCH, enhanced type checking
  - built their own scanner and parser using SableCC
- ableP — middleweight approach

An example

An altitude switch model that uses

- enhanced `select` statements
- DTSPIN-like constructs
- tabular Boolean expressions (*à la* RSML and SCR)

An instance of ableP parses and analyzes the model, then generates its translation to pure Promela.

```
% java -jar ableP.aviation.jar AltSwitch.xpml

% spin -a AltSwitch.pml
```

Our approach:

- Users choose (independently developed) extensions.
- Tools compose the extensions and Promela host language.
- Distinguish
  1. extension user
     - has no knowledge of language design or implementations
  2. extension developer
     - must know about language design and implementation

1. Tools and formalisms support automatic composition.
2. Modular analyses ensure the composition results in a working translator.

- Value easy composition over expressivity, accept some restrictions
  - on syntax
  - new constructs are translated to "pure" Promela
- ableP "instances" are smart pre-processors

Extending ableP with independently developed extensions

- Extension *user* directs underlying tools to
  - compose chosen extensions and the host language
  - and then create a custom translator for the extended language
- Silver grammar modules define sets of specifications
  - composition is set union, order does not matter
- Consider the Silver specification for this composition.

Developing language extensions

Two primary challenges:

1. composable syntax — enables building a scanner and parser
   - context-aware scanning [GPCE07]
   - modular determinism analysis [PLDI09]
   - Copper
2. composable semantics — analysis and translations
   - attribute grammars with forwarding, collections and higher-order attributes
   - set union of specification components
     - sets of productions, non-terminals, attributes
     - sets of attribute defining equations, on a production
     - sets of equations contributing values to a single attribute
   - modular well-definedness analysis [SLE12]
   - monolithic termination analysis [SLE12]
   - Silver

Context aware scanning

- Scanner recognizes only tokens valid for current "context"
- keeps embedded sub-languages, in a sense, separate
- Consider:
  - `chan in, out;`
    `for i in a { a[i] = i*i ; }`
- Two terminal symbols that match "`in`".
  - `terminal IN 'in' ;`
  - `terminal ID /[a-zA-Z_][a-zA-Z_0-9]*/`
    `submits to {promela_kwd };`
  - `terminal FOR 'for' lexer classes {promela_kwd };`

Allows parsing of embedded C code

```
c_decl {
  typedef struct Coord {
    int x, y; } Coord;      }

c_state "Coord pt" "Global"  /* goes in state vector */
int z = 3;                   /* standard global decl */

active proctype example()
{ c_code { now.pt.x = now.pt.y = 0; };

  do :: c_expr { now.pt.x == now.pt.y }
          -> c_code { now.pt.y++; }
     :: else -> break
  od;

  c_code { printf("values %d: %d, %d,%d\n",
              Pexample->_pid, now.z, now.pt.x, now.pt.y);
  };
}
```

Semantics for host language assignment constructs

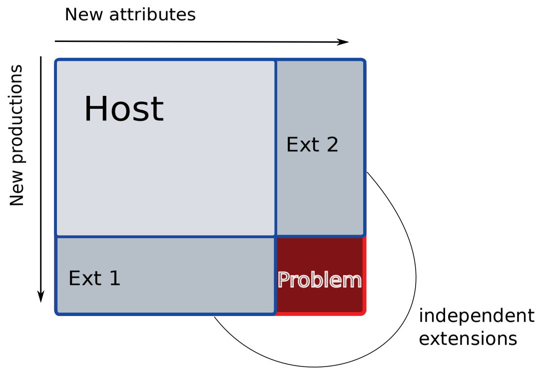
```
grammar edu:umn:cs:melt:ableP:host:core:abstractsyntax;

abstract production defaultAssign
s::Stmt ::= lhs::Expr rhs::Expr
{ s.pp = lhs.pp ++ " = " ++ rhs.pp ++ " ;\n" ;

  lhs.env = s.env;   rhs.env = s.env;
  s.defs = emptyDefs();

  s.errors := lhs.errors ++ rhs.errors ;
}
```

Adding extension constructs involves writing similar productions.

Adding ETCH-like semantic analysis.

```
grammar edu:umn:cs:melt:ableP:extensions:typeChecking ;

synthesized attribute typerep::TypeRep
  occurs on Expr, Decls ;

aspect production varRef
e::Expr ::= id::ID
{ e.typerep = ... retrieve from declaration
                  found in e.env ... ;   }

aspect production defaultAssign
s::Stmt ::= lhs::Expr rhs::Expr
{ s.errors <- if isCompatible(lhs.typerep, rhs.typerep)
              then [ ]
              else [ mkError ("Incompatible types ...") ];
}
```

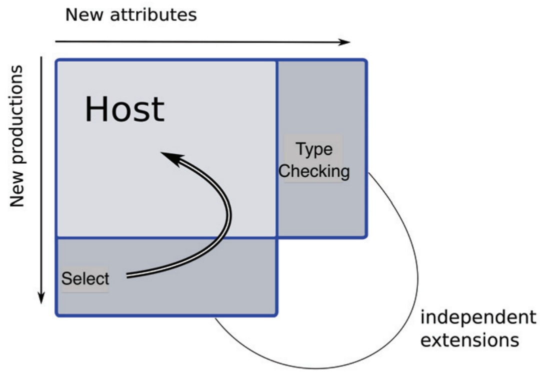
Extensibility: safe composability



Extensions get undefined semantics from host translation.

```
grammar edu:umn:cs:melt:ableP:extensions:enhancedSelect ;

abstract production selectFrom
s::Stmt ::= sl::'select' v::Expr es::Exprs
{
  s.pp = "select ( " ++ v.pp ++ ":" ++ es.pp ++ " ); \n" ;

  s.errors := v.errors ++ es.errors ++
     if  ... check that all expressions in 'es' have
             same type as 'v' ...
     then [ mkError ("Error: select statement " ++
                     "requires same type ... ") ]
     else [ ] ;

  forwards to ifStmt( mkOptions (v, es) ) ;
}
```

Extensibility: safe composability



Modular analysis

Ensuring that the composition will be successful.

Context free grammars

$$\boxed{G_H} \cup \boxed{G_E^1} \cup \boxed{G_E^2} \cup \ \ldots \boxed{G_E^i}$$

- $\cup$ of sets of nonterminals, terminals, productions
- Composition of all is an context free grammar.
- Is it non-ambiguous, useful for deterministic (LR) parsing?
- $conflictFree(G_H \cup G_E^1)$ holds
- $conflictFree(G_H \cup G_E^2)$ holds
- $conflictFree(G_H \cup G_E^i)$ holds
- $conflictFree(G_H \cup G_E^1 \cup G_E^2 \cup \ \ldots \ \cup G_E^i)$ may not hold

Attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup \boxed{AG_E^2} \cup \ \ldots \boxed{AG_E^i}$$

- $\cup$ of sets of attributes, attribute equations, occurs-on declarations
- Composition of all is an attribute grammar.
- Completeness: $\forall$ production, $\forall$ attribute, $\exists$ an equation
- $complete(AG_H \cup AG_E^1)$ holds
- $complete(AG_H \cup AG_E^2)$ holds
- $complete(AG_H \cup AG_E^i)$ holds
- $complete(AG_H \cup AG_E^1 \cup AG_E^2 \cup \ \ldots \ \cup AG_E^i)$ may not hold
- similarly for non-circularity of the AG

Detecting problems, ensuring composition

When can some analysis of the language specification be applied?
When ...

1. the host language is developed ?
2. a language extensions is developed ?
3. when the host and extensions are composed ?
4. when the resulting language tools are run ?

Libraries, and modular type checking

- Libraries "just work"
- Type checking is done by the library writer, modularly.
- Language extensions should be like libraries, composition of "verified" extensions should "just work."

Modular determinism analysis for grammars, 2009

$$\boxed{G_H} \cup \boxed{G_E^1} \cup \boxed{G_E^2} \cup \ldots \boxed{G_E^i}$$

- $isComposable(G_H, G_E^1) \wedge conflictFree(G_H \cup G_E^1)$ holds
- $isComposable(G_H, G_E^2) \wedge conflictFree(G_H \cup G_E^2)$ holds
- $isComposable(G_H, G_E^i) \wedge conflictFree(G_H \cup G_E^i)$ holds
- these imply $conflictFree(G_H \cup G_E^1 \cup G_E^2 \cup \ \ldots)$ holds
- $(\forall i \in [1, n].isComposable(G_H, G_E^i) \wedge conflictFree(G_H \cup \{G_E^i\}))$ $\Longrightarrow conflictFree(G_H \cup \{G_E^1, \ldots, G_E^n\})$
- Some restrictions to extension introduced syntax apply, of course.

So ...

- ableP supports the simple composition of language extensions
- This creates translators and analyzers for customized Promela-based languages.
  - extensions can be verified to (syntactically) compose, with other verified extensions — done by extension developers
  - adding (independently developed) extensions that add new features and new analysis on host features is supported
- Challenge: SPIN verification still occurs on the generated pure Promela specification.
- Future work
  - More extensions: multi-dimensional array, unit/dimension analysis, ...
  - Improve type analaysis
  - Semantic analysis of embedded C code?

Modular completeness analysis for attribute grammars

$$\boxed{AG_H} \cup \boxed{AG_E^1} \cup \boxed{AG_E^2} \cup \ldots \boxed{AG_E^i}$$

- $modComplete(AG_H \cup AG_E^1)$ holds
- $modComplete(AG_H \cup AG_E^2)$ holds
- $modComplete(AG_H \cup AG_E^i)$ holds
- these imply $complete(AG_H \cup AG_E^1 \cup AG_E^2 \cup \ \ldots)$ holds
- $(\forall i \in [1, n].modComplete(AG_H, AG_E^i))$ $\Longrightarrow complete(AG_H \cup \{AG_E^1, \ldots, AG_E^n\})$.
- similarly for non-circularity of the AG
- Again, some restrictions on extensions.

Thanks for your attention.

Questions?

http://melt.cs.umn.edu/
evw@cs.umn.edu

EXTENSIBLE COMPILER ARCHITECTURE – EXAMPLES FROM JMODELICA.ORG

Görel Hedin, Dept of Computer Science, Lund University, Sweden

The JModelica.org platform is built around an extensible compiler, implemented in reference attribute grammars (RAGs) using the JastAdd metacompiler. In this talk, I will give an overview of how extensible compiler architectures can be built using JastAdd and RAGs. Examples from the JModelica.org platform will be used for illustration

Extensible Compiler Architecture
Examples from JModelica.org

Uses of the JastAdd systems

Görel Hedin
Computer Science, Lund University

LCCC workshop, Lund, Sept 20, 2012

1

Background

JastAdd: an open source metacompiler for generating extensible compilers

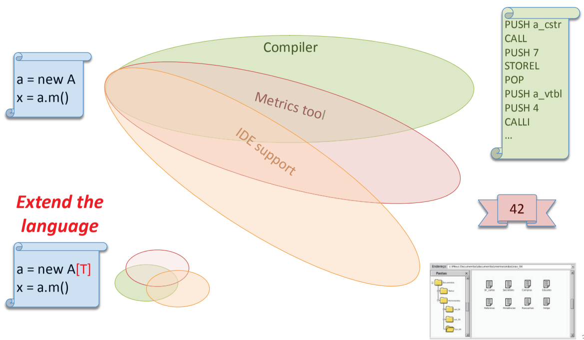
- Object-orientation (Java as host language)
- Aspect-oriented programming / Open classes
- Attribute grammars [Knuth 1968]
- Higher-order attributes [Vogt et al. 1989]
- Reference attributes [Hedin 2000]
- Context-dependent transformations [Ekman and Hedin 2004]
- ...

Applications

- JModelica.org, Optimica
- JastAddJ (extensible Java compiler)
- ...

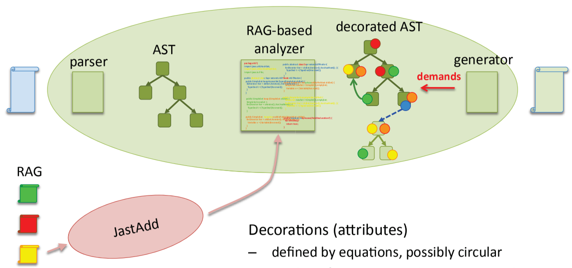
2

Why extensible compilers?



3

Modularizing the compiler using JastAdd



Decorations (attributes)

- defined by equations, possibly circular
- can be references—AST becomes a graph
- can be higher-order: new ASTs
- evaluated on demand—cached for efficiency
- arbitrary modularization of individual definitions

4

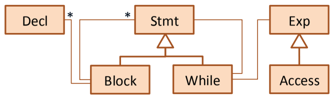
JastAdd programming mechanisms

OO: Classes, inheritance, overriding

```
abstract Stmt;
abstract Exp;
abstract Decl;
While : Stmt ::= Exp Stmt;
Block : Stmt ::= Decl* Stmt*;
Access: Exp ::= ID;
```

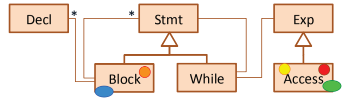
Open classes: Inter-type declarations

```
T1 Exp.f;
T2 Exp.m() { ... }
T2 Access.m() { ... }

Decl implements I;
```



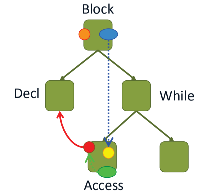
5

JastAdd programming mechanisms



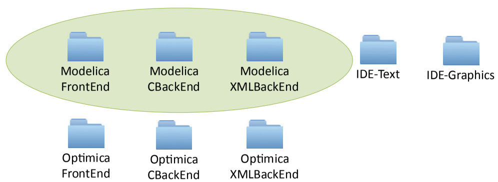
AGs: Synthesized and Inherited attributes

```
syn Decl Access.decl;
inh Decl Access.lookup(String s);
inh Decl Block.lookup(String s);
eq Access.decl = lookup(ID);
eq Block.stmts.lookup(String s) {
  res = decls.locals(s);
  if (res != null) return res;
  return lookup(s);
}
```



6

JModelica.org components



7

Compiling Modelica

```
model Bike
  Wheel frontwheel:
  Wheel backwheel;
end Bike;

model Wheel
  replaceable Brake brake;
end Wheel;

model Brake
end Brake;

model DiscBrake extends Brake
  Real discTemp;
end DiscBrake;

model DrumBrake extends Brake
end DrumBrake;

model MyBike extends Bike
  (frontwheel(redeclare DiscBrake brake),
  (backwheel(redeclare DrumBrake brake));
equation
  assert(frontwheel.brake.discTemp > 300,
   "Alarm: front wheel temperature too high");
end MyBike;
```

Key compilation analyses:
- Name analysis
- Type analysis
- Building the instance hierarchy

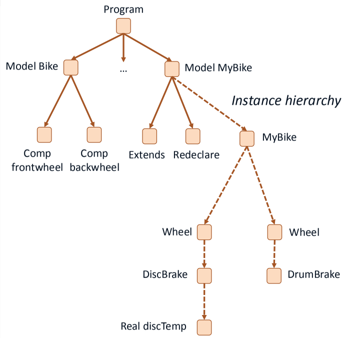
Challenge:
- Analyses are *interdependent*

JastAdd solution:
- instance tree — higher-order attributes
- automatic interleaving of the analyses

8

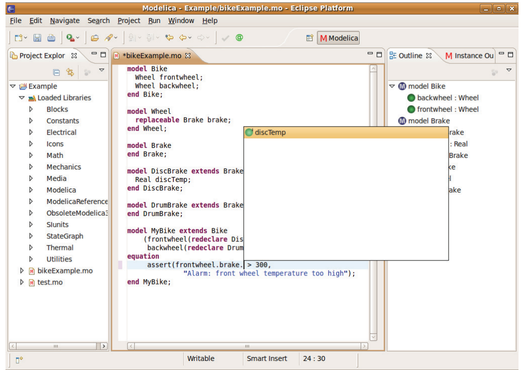
Compiling Modelica

```
model Bike
  Wheel frontwheel:
  Wheel backwheel;
end Bike;

model Wheel
  replaceable Brake brake;
end Wheel;

model Brake
end Brake;

model DiscBrake extends Brake
  Real discTemp;
end DiscBrake;

model DrumBrake extends Brake
end DrumBrake;

model MyBike extends Bike
  (frontwheel(redeclare DiscBrake brake),
  (backwheel(redeclare DrumBrake brake));
equation
  assert(frontwheel.brake.discTemp > 300,
   "Alarm: front wheel temperature too high");
end MyBike;
```



9

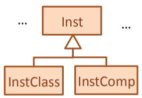
IDE name completion



10

Extending the compiler with name completion

Compiler



Modelica Name Completion

```
Inst implements CompletionNode;

public ArrayList Inst.completionProposals() {
 return completionNodes();
}

syn ArrayList Inst.completionNodes();

eq InstClass.completionNodes() =
 ... access compiler attributes ...

eq InstComp.completionNodes() =
 ... access compiler attributes ...
```

IDE framework

...

CompletionNode
completionProposals()

...

11

Optimica: an extended language

```
model Car
  Real x(start=0);
  Real v(start=0);
  input Real u;
equation
  der(x)=v;
  der(v)=u;
end Car;
```

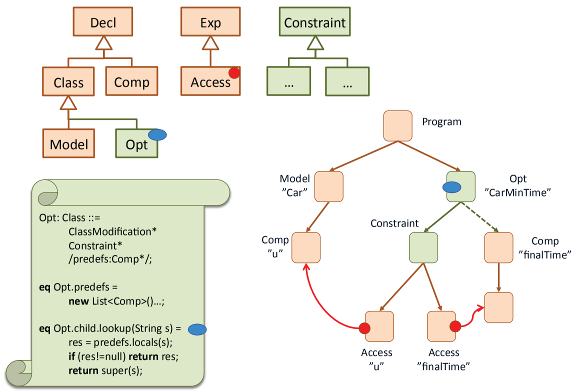
Modelica code

```
optimization CarMinTime (
      objective=finalTime ,
      startTime=0,
      finalTime(free=true, initialGuess=1))
   Car car(u(free=true, initialGuess=0.0));
constraint
   car.x(finalTime)=1;
   car.v(finalTime)=0;
   car.v<=0.5;
   car.u>=-1;
   car.u<=1;
end CarMinTime;
```

Optimica code
- extends Modelica with new syntax
- and changed semantics

12

Extending Modelica to Optimica



13

Ongoing and future work

- Incremental updating
- General IDE support
- Graphical editing
- Performance
- Higher-level specification

14

Conclusions

JModelica.org, a great case for JastAdd!

For more information, see jastadd.org

Thank you!

Questions?

15

CONSTRAINT SATISFACTION METHODS IN EMBEDDED SYSTEM DESIGN

Krzysztof Kuchcinski, Dept. of Computer Science, Lund University

Constraints can be used to define embedded systems parameters, requirements and specific design problem restrictions. They can be further formalized using Constraint Programming (CP) models. These models represent instances of Constraint Satisfaction Problem (CSP) and can be solved using CP solvers. CP is relatively young area that gains attention because of its flexibility to define different problems and possibility of using both complete and heuristic methods for their solving. Moreover, CP offers global constraint, such as scheduling constraints, that implement specific algorithms for efficient handling of a given class of problems. This provides an easy way to use several advanced algorithms in one problem that is difficult or time consuming in pure heuristic solutions. In this talk, we will concentrate on finite domain constraints and the related constraint programming framework. We will illustrate it with classical examples from embedded systems, such as scheduling, design mapping, register and memory allocation. Bandwidth auctions and their parallels to power.

Constraint satisfaction methods in embedded system design

Krzysztof Kuchcinski

Dept. of Computer Science,
Lund University, Sweden

Outline

1. Motivation an Example
2. CP Basics
3. Advanced Example- Sub-graph Isomorphism
4. Summary and Conclusions

Outline

1. Motivation an Example
2. CP Basics
3. Advanced Example- Sub-graph Isomorphism
4. Summary and Conclusions

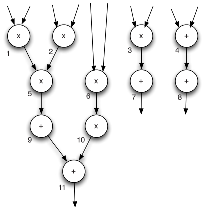
Why constraints?

- Examples of combinatorial optimization problems in embedded systems
  - Scheduling, allocation and assignment,
  - Partitioning,
  - Memory and register assignment,
  - Instruction selection.
- Different constraints:
  - timing,
  - resource,
  - power consumption, etc.
- Constraint programming over finite domain– combinatorial optimization problems!!!
- Constraint programming offers a *unified* approach to model and solve problems with *heterogeneous* constraints.

LCCC workshop 2012

Scheduling example

Simple data-flow graph

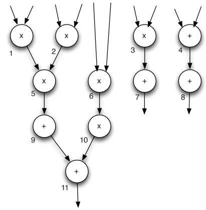


Krzysztof Kuchcinski 5(21)

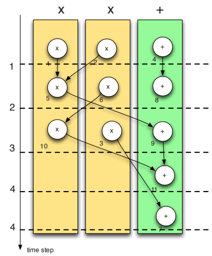
LCCC workshop 2012

Scheduling example

Simple data-flow graph



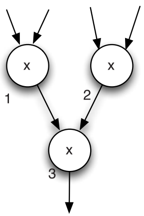
Simple schedule



Krzysztof Kuchcinski 5(21)

LCCC workshop 2012

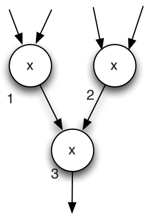
Scheduling Constraints



Krzysztof Kuchcinski 6(21)

LCCC workshop 2012

Scheduling Constraints



Variables

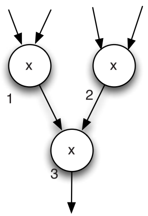
Operation start
$t_1 :: \{0..10\}, t_2 :: \{0..10\}, t_3 :: \{0..10\}$

Assigned resource
$r_1 :: \{1..2\}, r_2 :: \{1..2\}, r_3 :: \{1..2\}$

Krzysztof Kuchcinski 6(21)

LCCC workshop 2012

Scheduling Constraints



Variables

Operation start
$t_1 :: \{0..10\}, t_2 :: \{0..10\}, t_3 :: \{0..10\}$

Assigned resource
$r_1 :: \{1..2\}, r_2 :: \{1..2\}, r_3 :: \{1..2\}$

Constraints

Precedence constraints
$t_1 + d_1 \leq t_2 \wedge$
$t_2 + d_2 \leq t_3 \wedge$

Resource constraints
$(t_1 + d_1 \leq t_2 \vee t_2 + d_2 \leq t_1 \vee r_1 \neq r_2)$

Krzysztof Kuchcinski 6(21)

LCCC workshop 2012

Global Constraints

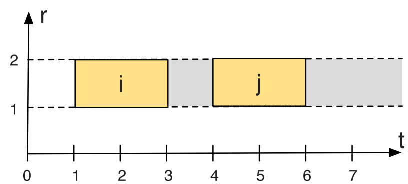
$\forall i, j$ where $i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$

Krzysztof Kuchcinski 7(21)

LCCC workshop 2012

Global Constraints

$\forall i, j$ where $i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$



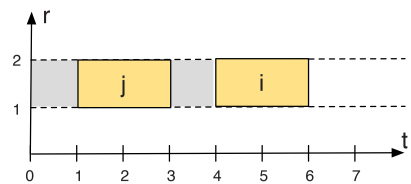
Diff2 constraint (non-overlapping rectangles)

Krzysztof Kuchcinski 7(21)

LCCC workshop 2012

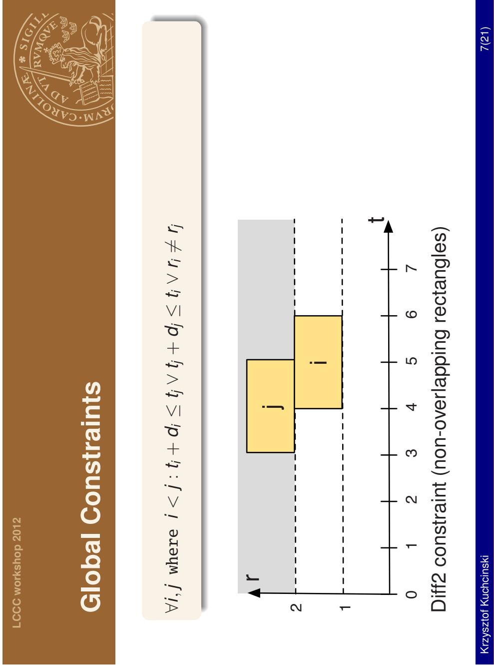
Global Constraints

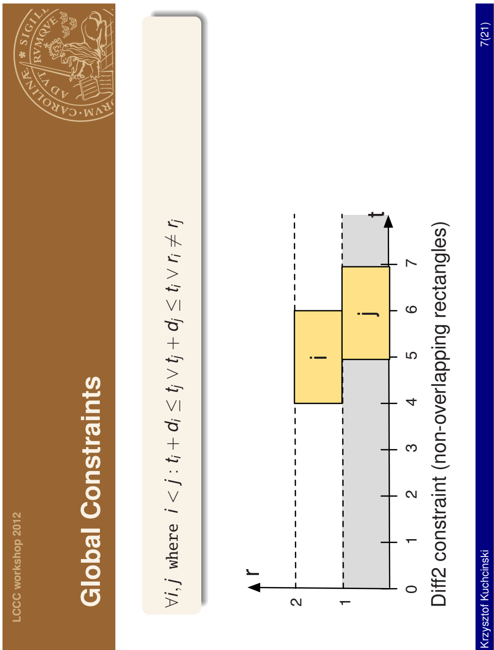
$\forall i, j$ where $i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$

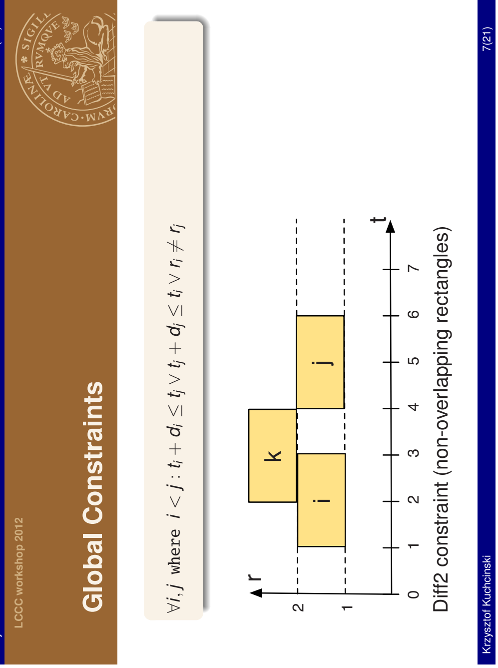


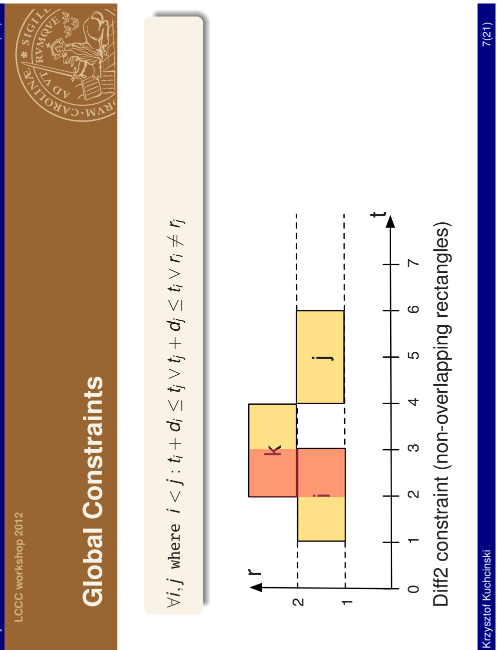
Diff2 constraint (non-overlapping rectangles)

Krzysztof Kuchcinski 7(21)


LCCC workshop 2012
Global Constraints
$\forall i,j$ where $i<j : t_i+d_i \le t_j \lor t_j+d_j \le t_i \lor r_i \ne r_j$
r
t
2
1
0
1
2
3
4
5
6
7
i
j
Diff2 constraint (non-overlapping rectangles)
Krzysztof Kuchcinski
7(21)


LCCC workshop 2012
Global Constraints
$\forall i,j$ where $i<j : t_i+d_i \le t_j \lor t_j+d_j \le t_i \lor r_i \ne r_j$
r
t
2
1
0
1
2
3
4
5
6
7
i
j
Diff2 constraint (non-overlapping rectangles)
Krzysztof Kuchcinski
7(21)


LCCC workshop 2012
Global Constraints
$\forall i,j$ where $i<j : t_i+d_i \le t_j \lor t_j+d_j \le t_i \lor r_i \ne r_j$
r
t
2
1
0
1
2
3
4
5
6
7
k
i
j
Diff2 constraint (non-overlapping rectangles)
Krzysztof Kuchcinski
7(21)


LCCC workshop 2012
Global Constraints
$\forall i,j$ where $i<j : t_i+d_i \le t_j \lor t_j+d_j \le t_i \lor r_i \ne r_j$
r
t
2
1
0
1
2
3
4
5
6
7
k
i
j
Diff2 constraint (non-overlapping rectangles)
Krzysztof Kuchcinski
7(21)

LCCC workshop 2012

Final Model

```
array[1..n] of var 0..100 : t;
array[1..n] of var 1..2 : r;

% precedence constraints
constraint
   t[1] + 2 =< t[6] /\ t[2] + 2 =< t[6] /\ t[3] + 2 =< t[7] /\
   t[4] + 2 =< t[8] /\ t[5] + 1 =< t[9] /\ t[6] + 2 =< t[10] /\
   t[7] + 2 =< t[11] /\ t[10] + 1 =< t[11];

constraint
   % resource constraints for adders
   diff2([[t[5],r[5],1,1], [t[8],r[8],1,1], [t[9],r[9],1,1],
          [t[10],r[10],1,1], [t[11],r[11],1,1] ])
   /\
   % resource constraints for multipliers
   diff2([[t[1],r[1],2,1], [t[2],r[2],2,1], [t[3],r[3],2,1],
          [t[4],r[4],2,1], [t[6],r[6],2,1], [t[7],r[7],2,1]]);
```

Krzysztof Kuchcinski 8(21)

LCCC workshop 2012

Model Advantages

- Separation of a model and solving method
- Time-constrained and resource-constrained scheduling
- Easy to add new constraints
- Non-linear constraints
- Combination of consistency algorithms (e.g., diff2 and cumulative constraints)
- Standard and heuristic methods for solving the model

Krzysztof Kuchcinski 9(21)

LCCC workshop 2012

Outline

1. Motivation an Example
2. **CP Basics**
3. Advanced Example- Sub-graph Isomorphism
4. Summary and Conclusions

Krzysztof Kuchcinski 10(21)

LCCC workshop 2012

CP basics

- Finite domain variables, e.g., $t :: 0..10$
- Constraints; defined by their consistency methods (propagators)
- Primitive constraints
  - $a + b < c$, $x \cdot y = z$, $A \cup B = C$, etc.
  - bounds and domain consistency
- Global constraints
  - diff2, alldifferent, etc.
  - can be decomposed to primitive constraints BUT
  - specialized algorithms from operation research, graph theory, computational geometry, etc. are more efficient

Krzysztof Kuchcinski 11(21)

LCCC workshop 2012

Propagators

Propagator for $x + y = z$ (bounds consistency)

$$x \textbf{ in } \{\min(z) - \max(y) \mathinner{..} \max(z) - \min(y)\}$$
$$y \textbf{ in } \{\min(z) - \max(x) \mathinner{..} \max(z) - \min(x)\}$$
$$z \textbf{ in } \{\min(x) + \min(y) \mathinner{..} \max(x) + \max(y)\}$$

Krzysztof Kuchcinski 12(21)

LCCC workshop 2012

Propagators

Propagator for $x + y = z$ (bounds consistency)

$$x \textbf{ in } \{\min(z) - \max(y) \mathinner{..} \max(z) - \min(y)\}$$
$$y \textbf{ in } \{\min(z) - \max(x) \mathinner{..} \max(z) - \min(x)\}$$
$$z \textbf{ in } \{\min(x) + \min(y) \mathinner{..} \max(x) + \max(y)\}$$

Example

$x :: \{1..10\}$, $y :: \{1..10\}$ and $z :: \{1..10\}$
yields
$x :: \{1..9\}$, $y :: \{1..9\}$ and $z :: \{2..10\}$.

Krzysztof Kuchcinski 12(21)

LCCC workshop 2012

Global Constraints

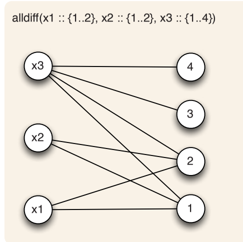
- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.

Krzysztof Kuchcinski 13(21)

LCCC workshop 2012

Global Constraints

- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.

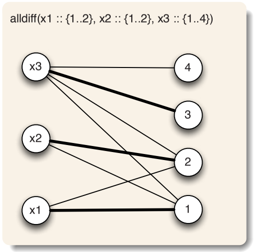


Krzysztof Kuchcinski 13(21)

LCCC workshop 2012

Global Constraints

- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.

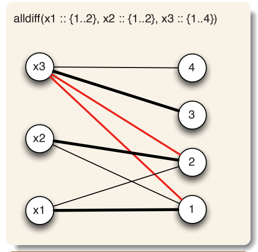


Krzysztof Kuchcinski 13(21)

LCCC workshop 2012

Global Constraints

- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.



Berge, 1973

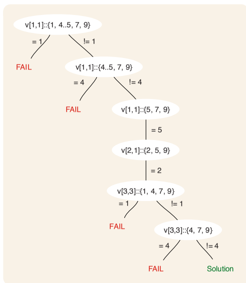
An edge belongs to a maximum matching iff for some maximum matching, it belongs to either an even alternating path which begins at a free node, or to an even alternating cycle.

Krzysztof Kuchcinski 13(21)

LCCC workshop 2012

Solving

- Systematically assign values to variables and check if the problem is still consistent
- Implemented usually as depth-first-search
- Other methods can be used instead of assigning values, i.e., constraints on tasks ordering
- Heuristics can be incorporated



Krzysztof Kuchcinski 14(21)

LCCC workshop 2012

Outline

1. Motivation an Example
2. CP Basics
3. **Advanced Example- Sub-graph Isomorphism**
4. Summary and Conclusions

Krzysztof Kuchcinski 15(21)

Subgraph Isomorphism Constraint

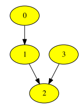
Definition (Subgraph isomorphism)

Target $G_t = (N_t, E_t)$ and pattern $G_p = (N_p, E_p)$ graphs are subgraph isomorphic iff there exist an injective function $f : N_p \to N_t$ respecting $(u, v) \in E_p \Leftrightarrow (f(u), f(v)) \in E_t$.

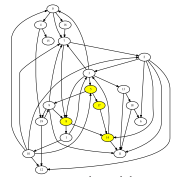
Subgraph Isomorphism Constraint

Definition (Subgraph isomorphism)

Target $G_t = (N_t, E_t)$ and pattern $G_p = (N_p, E_p)$ graphs are subgraph isomorphic iff there exist an injective function $f : N_p \to N_t$ respecting $(u, v) \in E_p \Leftrightarrow (f(u), f(v)) \in E_t$.

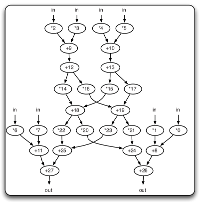


pattern graph

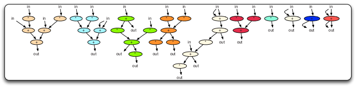


target graph with matching

Instruction Identification and Selection



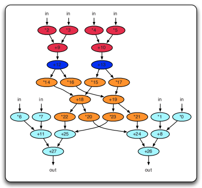
Data-flow graph



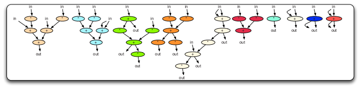
Computational patterns

- Computational patterns - connected components of the graph

Instruction Identification and Selection (cont'd)



Covered data-flow graph



Computational patterns

- Find sub-graph isomorphism that fulfills additional constraints (e.g., shortest schedule)

LCCC workshop 2012

Outline

1. Motivation an Example
2. CP Basics
3. Advanced Example- Sub-graph Isomorphism
4. **Summary and Conclusions**

LCCC workshop 2012

Our Solver

Java **Co**nstraint **P**rogramming

- constraint programming paradigm implemented in Java.
- provides different type of constraints
  - *primitive constraints*, such as arithmetical constraints (+, *, div, mod, etc.), equality (=) and inequalities (<, >, =<, >=, !=).
  - *logical, reified and conditional constraints*
  - *global constraints*.
  - *set constraints*, such as =, $\bigcup$, $\bigcap$.
  - *stochastic variables and constraints*.
- High-level language, minizinc, interface
- `http://www.jacop.eu`
- `http://sourceforge.net/projects/jacop-solver/`

LCCC workshop 2012

Conclusions

- Easy way of modeling problems with heterogeneous constraints
- Easy to extend the problem with new constraints
- Can handle non-linear constraints
- Combination of different algorithms through global constraints
- Separation between modeling and solving
- Both complete and heuristic methods can be used for finding solutions

DYNAMICAL MODELS FOR INDUSTRIAL CONTROLS: USE CASES AND CHALLENGES

Fernando D'Amato, GE Global Research Center

This presentation will introduce a few cases of industrial model-based controls in which model development has been critical for implementation success. Then, a brief description will be given of desired model properties for advanced controls, especially when dealing with formal optimization processes. Finally, a sample of the main challenges faced by modelling practices for industrial controls will be discussed.

Dynamical models for industrial controls: use cases and challenges

Fernando D'Amato
Principal Engineer, Controls, Electronics & Signal Processing
General Electric Global Research

LCCC workshop: Systems Design Meets Equation-Based Languages

Lund, September 2012

imagination at work

Outline

- Overview of controls at General Electric
- Train trip optimization example
- Power plant predictive control example
- From control system challenge to model challenge
- Conclusions

imagination at work 2

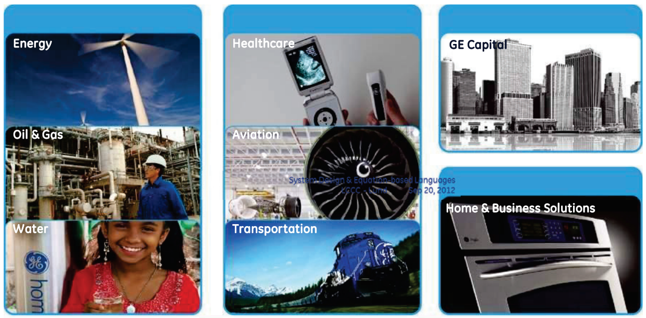
GE … a heritage of innovation

- Founded in 1892
- 300,000 employees worldwide
- $150 billion in annual revenues
- Only company in Dow Jones index originally listed in 1896

imagination at work 3

System Design & Equation-based Languages

GE today



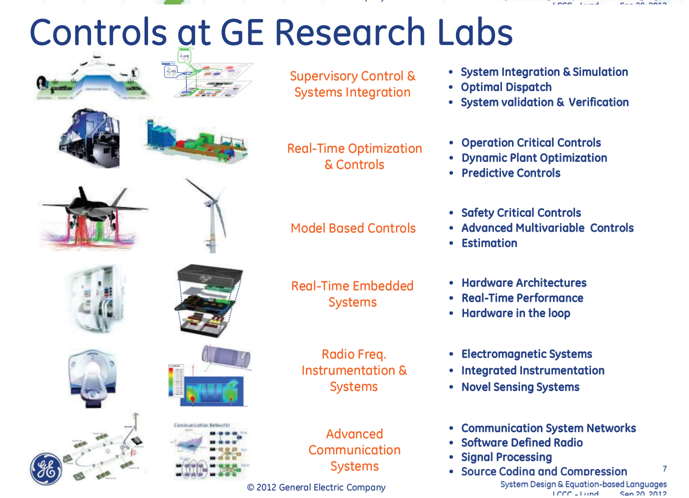
Aligned for growth

imagination at work 4

System Design & Equation-based Languages


Expanding global presence in research
3000 technologists worldwide
AMSTC
Ann Arbor, MI
Global Research HQ
Niskayuna, NY
Global Research - Europe
Munich, Germany
China Technology Center
Shanghai, China
Brazil Technology Center
Rio de Janeiro, Brazil
John F. Welch Technology Center
Bangalore, India
imagination at work
© 2012 General Electric Company
System Design & Equation-based Languages

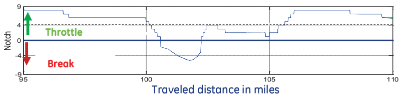

Products with Controls
IGCC
Wind
Nuclear
Power Gen
4000 Controls Technologists Supporting GE's diverse applications
Platforms
Healthcare
Oil & Gas
Aviation
Transportation
Water
imagination at work
6
© 2012 General Electric Company
System Design & Equation-based Languages


Controls at GE Research Labs
Supervisory Control & Systems Integration
• System Integration & Simulation
• Optimal Dispatch
• System validation & Verification
Real-Time Optimization & Controls
• Operation Critical Controls
• Dynamic Plant Optimization
• Predictive Controls
Model Based Controls
• Safety Critical Controls
• Advanced Multivariable Controls
• Estimation
Real-Time Embedded Systems
• Hardware Architectures
• Real-Time Performance
• Hardware in the loop
Radio Freq. Instrumentation & Systems
• Electromagnetic Systems
• Integrated Instrumentation
• Novel Sensing Systems
Advanced Communication Systems
• Communication System Networks
• Software Defined Radio
• Signal Processing
• Source Coding and Compression
7
© 2012 General Electric Company
System Design & Equation-based Languages


Transportation:
Optimal train control
2005
2005
Optimize fuel utilization
in every trip

The Problem

Online calculation of optimal acceleration and breaking for fuel efficiency



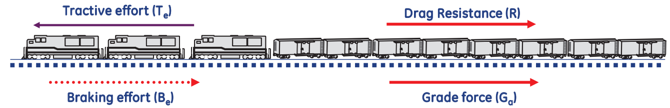
Constraints

- Arrival timing
- Speed limits (mile per mile)
- Fuel reserves
- Maximum internal forces

Uncertainty/Variability

- Train weight
- Track conditions
- Other trains operation

Approach: Online optimal control



Input Data

1. Trip details
- Departure and arrival locations
- Objective: min time or set pace
- Train manifest (load, consist)

2. Track database
- Speed restrictions
- Grade & Curvature
- GPS coordinates

3. Loco and Train Physics
- Power: TE, SFC, Braking
- Train: Weight, length, drag
- Equations of motion

Physics based optimization

$$\min_{u(t)} \alpha_1 \int_0^{t_f} Fuel(u(t))\,dt + \alpha_2 \int_0^{t_f} \dot{v}(t)^2\,dt + \alpha_3 t_f$$

subject to:

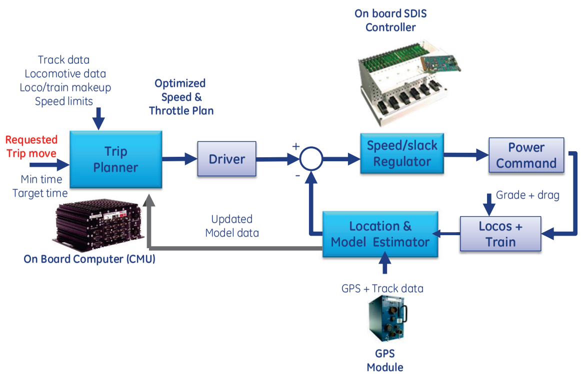
$$\dot{x}(t) = v(t)$$
$$\dot{v}(t) = \frac{1}{M}\left(T_e(u,v) - B_e(u,v) - G_e(x) - R(v)\right)$$
$$\dot{u}(t) = \frac{du(t)}{dt}$$
$$x(0) = x_0;\ x(t_f) = x_f$$
$$v(0) = v_0;\ v(t_f) = v_f$$
$$u(0) = u_0$$
$$0 \le v \le \text{speed limit}$$
$$0 \le t \le t_f$$

T_e : Tractive Effort
B_e : Braking Effort
R : Drag Resistance
G : Gradient Force
x_0, x_f : initial and final distance
v_0, v_f : initial and final speed
u_0 : initial notch
tf : maximum trip time

Calculated magnitudes

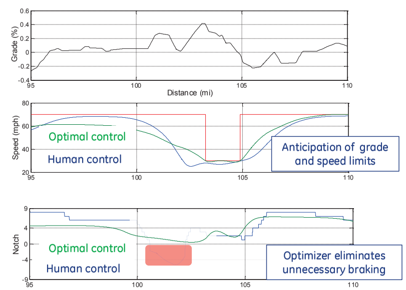
Driving plan vs. distance
- Optimal speed
- Optimal notch
- Expected arrival time
- Expected fuel use

Implementation

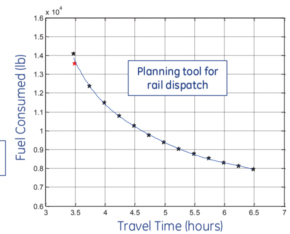


Results

Improvements from optimal control

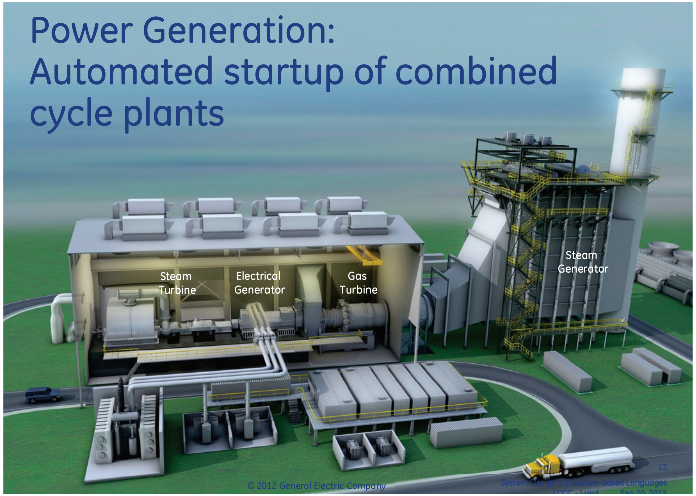


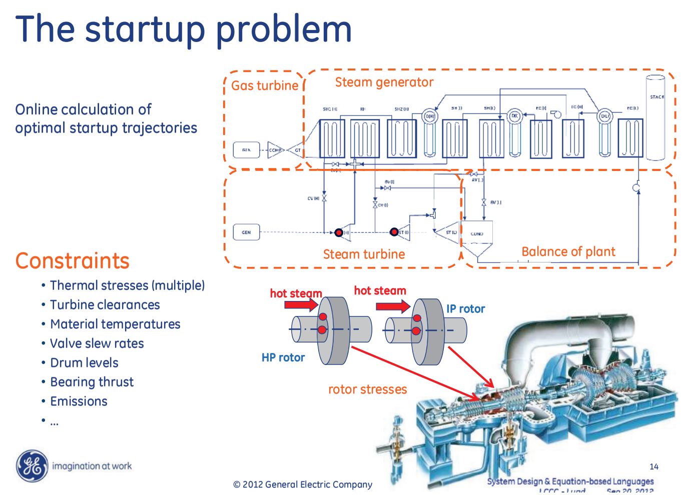
Entitlement curve

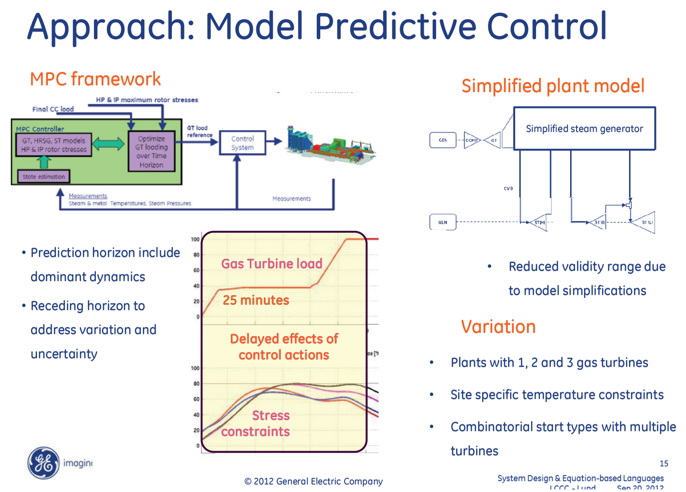


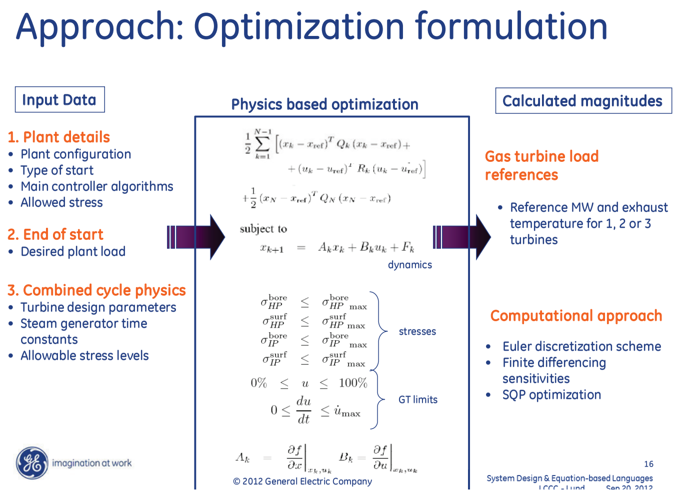
Impact

- Runs on BSNF, CP, CSX, CN, coal, grain & general merchandise
- 97 Subdivisions, 17000 Track Miles
- **10+ % system-wide average fuel savings, no velocity impact**


Power Generation:
Automated startup of combined cycle plants
Steam Turbine
Electrical Generator
Gas Turbine
Steam Generator
© 2012 General Electric Company
13


The startup problem
Online calculation of optimal startup trajectories
Gas turbine
Steam generator
Steam turbine
Balance of plant
Constraints
• Thermal stresses (multiple)
• Turbine clearances
• Material temperatures
• Valve slew rates
• Drum levels
• Bearing thrust
• Emissions
• ...
hot steam
hot steam
IP rotor
HP rotor
rotor stresses
imagination at work
© 2012 General Electric Company
14
System Design & Equation-based Languages


Approach: Model Predictive Control
MPC framework
Final CC load
HP & IP maximum rotor stresses
MPC Controller
GT, HRSG, ST models HP & IP rotor stresses
Optimize GT loading over Time Horizon
State estimation
GT load reference
Control System
Measurements
• Prediction horizon include dominant dynamics
• Receding horizon to address variation and uncertainty
Gas Turbine load
25 minutes
Delayed effects of control actions
Stress constraints
Simplified plant model
Simplified steam generator
• Reduced validity range due to model simplifications
Variation
• Plants with 1, 2 and 3 gas turbines
• Site specific temperature constraints
• Combinatorial start types with multiple turbines
© 2012 General Electric Company
15
System Design & Equation-based Languages


Approach: Optimization formulation
Input Data
1. Plant details
• Plant configuration
• Type of start
• Main controller algorithms
• Allowed stress
2. End of start
• Desired plant load
3. Combined cycle physics
• Turbine design parameters
• Steam generator time constants
• Allowable stress levels
Physics based optimization
subject to
dynamics
stresses
GT limits
Calculated magnitudes
Gas turbine load references
• Reference MW and exhaust temperature for 1, 2 or 3 turbines
Computational approach
• Euler discretization scheme
• Finite differencing sensitivities
• SQP optimization
imagination at work
© 2012 General Electric Company
16
System Design & Equation-based Languages

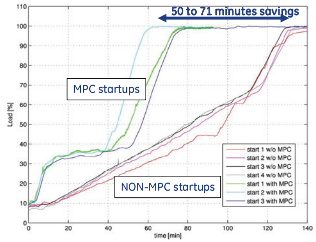
Implementation

imagination at work

© 2012 General Electric Company

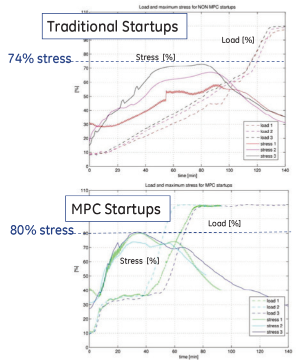
System Design & Equation-based Languages LCCC - Lund Sep 20, 2012 17

Results



Typical benefits per start

- Time savings: 1 hour
- Fuel (NG) savings: 70,000 lbm
- Fuel cost reduction: $10K
- NOx reduction: 140 lbm



Virtually no impact on life

imagination at work

© 2012 General Electric Company

System Design & Equation-based Languages LCCC - Lund Sep 20, 2012 18

Trends

Calculations getting faster & cheaper

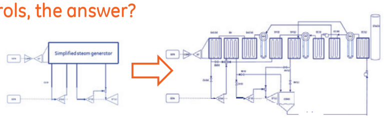
- Computing HW performance ↑
- Algorithms performance ↑
- Computing cost ↓

Increasing performance demands

- Competitiveness in market place
- Increased operation flexibility
- Transient efficiency
- Environmental regulations

Advanced Model Based Controls, the answer?

- More detailed physical models
- Rely more on optimization



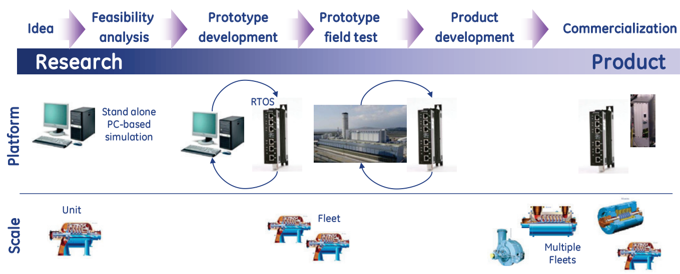
Significant challenges ahead ...

imagination at work

© 2012 General Electric Company

System Design & Equation-based Languages LCCC - Lund Sep 20, 2012 19

Industrial Control Development



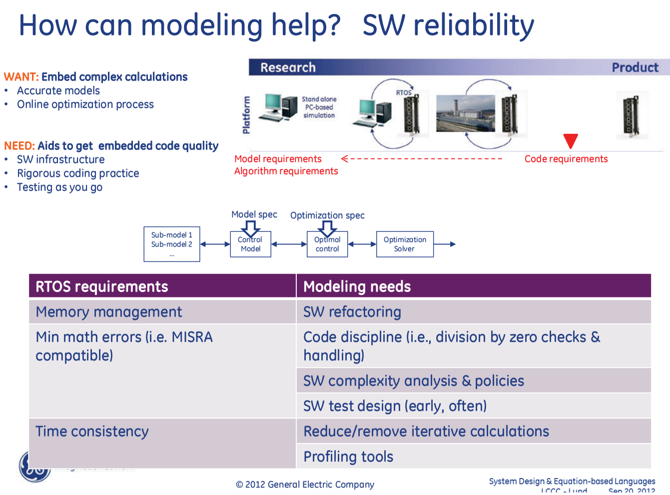
Challenges for model-based control products

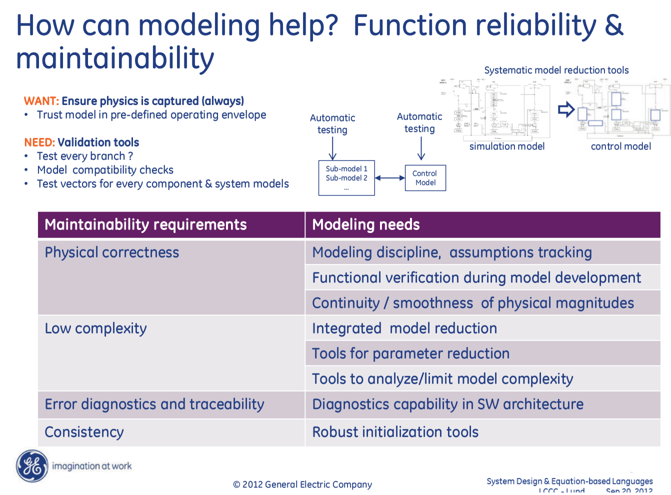
- Time to market
- Cost & complexity → development, deployment, maintenance

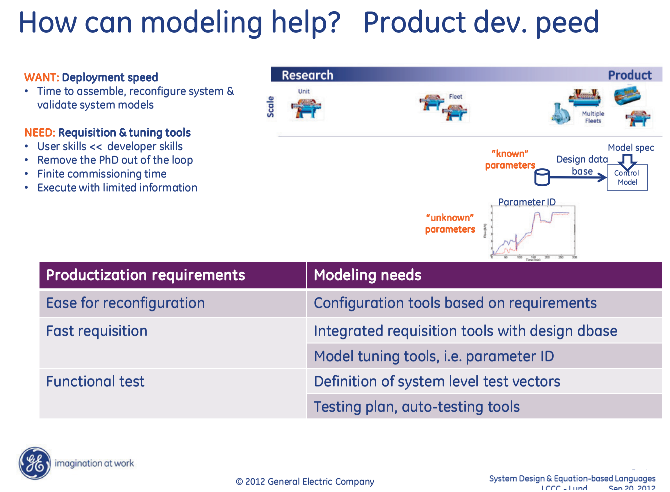
imagination at work

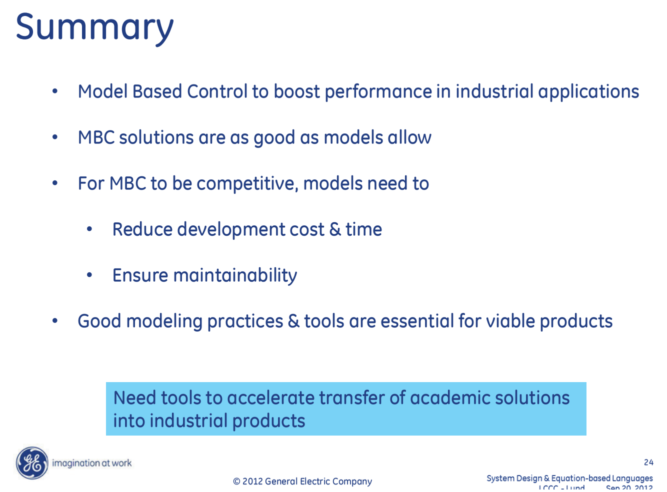
© 2012 General Electric Company

System Design & Equation-based Languages LCCC - Lund Sep 20, 2012


How can modeling help? SW reliability
WANT: Embed complex calculations
• Accurate models
• Online optimization process
NEED: Aids to get embedded code quality
• SW infrastructure
• Rigorous coding practice
• Testing as you go
Research
Product
Platform
Stand alone PC-based simulation
RTOS
Model requirements
Algorithm requirements
Code requirements
Model spec
Optimization spec
Sub-model 1
Sub-model 2
...
Control Model
Optimal control
Optimization Solver
RTOS requirements | Modeling needs
Memory management | SW refactoring
Min math errors (i.e. MISRA compatible) | Code discipline (i.e., division by zero checks & handling)
SW complexity analysis & policies
SW test design (early, often)
Time consistency | Reduce/remove iterative calculations
Profiling tools
© 2012 General Electric Company
System Design & Equation-based Languages


How can modeling help? Function reliability & maintainability
WANT: Ensure physics is captured (always)
• Trust model in pre-defined operating envelope
NEED: Validation tools
• Test every branch ?
• Model compatibility checks
• Test vectors for every component & system models
Automatic testing
Automatic testing
Sub-model 1
Sub-model 2
...
Control Model
Systematic model reduction tools
simulation model
control model
Maintainability requirements | Modeling needs
Physical correctness | Modeling discipline, assumptions tracking
Functional verification during model development
Continuity / smoothness of physical magnitudes
Low complexity | Integrated model reduction
Tools for parameter reduction
Tools to analyze/limit model complexity
Error diagnostics and traceability | Diagnostics capability in SW architecture
Consistency | Robust initialization tools
imagination at work
© 2012 General Electric Company
System Design & Equation-based Languages


How can modeling help? Product dev. peed
WANT: Deployment speed
• Time to assemble, reconfigure system & validate system models
NEED: Requisition & tuning tools
• User skills << developer skills
• Remove the PhD out of the loop
• Finite commissioning time
• Execute with limited information
Research
Product
Scale
Unit
Fleet
Multiple Fleets
"known" parameters
Design data base
Model spec
Control Model
Parameter ID
"unknown" parameters
Productization requirements | Modeling needs
Ease for reconfiguration | Configuration tools based on requirements
Fast requisition | Integrated requisition tools with design dbase
Model tuning tools, i.e. parameter ID
Functional test | Definition of system level test vectors
Testing plan, auto-testing tools
imagination at work
© 2012 General Electric Company
System Design & Equation-based Languages


Summary
• Model Based Control to boost performance in industrial applications
• MBC solutions are as good as models allow
• For MBC to be competitive, models need to
• Reduce development cost & time
• Ensure maintainability
• Good modeling practices & tools are essential for viable products
Need tools to accelerate transfer of academic solutions into industrial products
imagination at work
24
© 2012 General Electric Company
System Design & Equation-based Languages

ORIGINS OF EQUATION-BASED MODELING LANGUAGES

Karl Johan Åström, Department of Automatic Control, LTH, Lund University Lund, Sweden

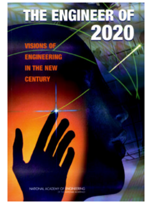
Modeling and simulation are indispensable tools for design and operation of complex engineered systems. Models are used in the design phase to select system architecture and configuration and for optimization of the design. Simulation is used to investigate dynamic behavior and to explore control architecture and control design. Simulation can be combined with real hardware in hardware-in-the-loop simulation for system testing. Models are also integral parts of feedback systems. Dynamic models are used during operation for control, dynamic optimization, supervision and fault diagnosis. They are also used in simulators for operator training. Models and simulation can also be used for decision support systems. Modeling is a rich field. It covers large parts of natural science and engineering. Statistics, design of experiments and parameter estimation are also essential ingredients. Numerical mathematics is important for simulation of a model and for optimization. Computer algebra is indispensable for safe transformation of models and for dealing with models of complex systems. Concepts and tools from computer science and software engineering are necessary to deal with large systems. Development of modeling and simulation worked hand in hand with emergence of computing starting with analog computers. The paper presents some of the ideas that lead to equation-based languages like Modelica.

Origins of Equation-Based Modeling

Karl Johan Åström
Department of Automatic Control LTH
Lund University

Modeling is Important

There will be growth in areas of simulation and modeling around the creation of new engineering “structures”. Computer-based design-build engineering ... will become the norm for most product designs, accelerating the creation of complex structures for which multiple subsystems combine to form a final product.



NAE The Engineer of 2020

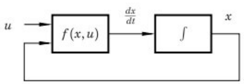
1. Introduction
2. **Block diagram modeling**
3. Equation-based modeling
4. Summary

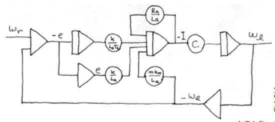
Vannevar Bush 1927

Engineering can progress no faster than the mathematical analysis on which it is based. Formal mathematics is frequently inadequate for numerous problems, a mechanical solution offers the most promise.

Analog Computing

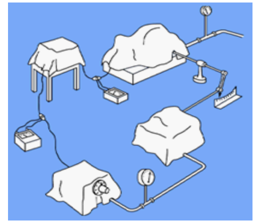
- Use a feedback loop to solve ODEs
- Integrators and function generation
- Linear systems integrators, +, -, *
- Parallelism
- Algebraic loop (loop without integrator)
- Scaling and alarms for out of scale!!





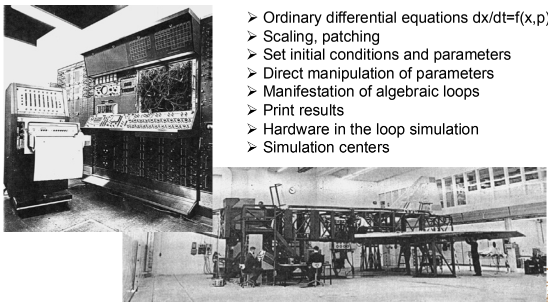
Block Diagram Modeling

- Information hiding
- Very useful abstraction
- Essential for control
- Causal inputs-output models
- Blocks described by ODE
- Base for analog computing
- BUT not for serious physical modeling



Oppelt 1954

Analog Simulation - HIL



- Ordinary differential equations dx/dt=f(x,p)
- Scaling, patching
- Set initial conditions and parameters
- Direct manipulation of parameters
- Manifestation of algebraic loops
- Print results
- Hardware in the loop simulation
- Simulation centers

Digital Emulators

- Precompilers to FORTRAN
- MIMIC Wright-Patterson 1965
- CSMP IBM 1962
- Babels tower > 30 emulators by 1965
- CSSL Simulation Council 1967
- ACSL Gauthier and Mitchell 1975
- SIMNON Elmqvist 1975
- MATLAB Cleve Moler 1980
- System Build, MatrixX 1984
- LabView 1986
- PC Matlab 1984, Simulink 1991

LTH in the 70s

- New control department at LTH (1965) in new school (1961) close to an old university
- Research program in Control Department: Optimization, Computer Control, System Identification, Adaptive Control, Applications:, Computer Aided Control Engineering (CACE)
- Embedded systems taught in the control department from 1970
- Interactive computing Wieslander: INTRAC, SYNPAC, IDPAC, MODPAC. FORTRAN based widely distributed
- A nonlinear simulator was missing

Simnon Elmqvist 1972

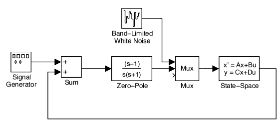
A block diagram language and an interactive simulator
Formal syntax in Bachus Naur format
Six basic commands: SYST, PAR, INIT SIMU, PLOT, AXES
Seven auxiliary: STORE, SHOW, DISP, SPLIT, HCOPY, ALGOR, ERROR

```
CONTINUOUS SYSTEM proc
Input u
Output y
State x
Der dx
dx=sat(u,0.1)
END

CONNECTING SYSTEM
yr(reg)=1; y(reg)=y(proc)
u(proc)=u(reg)
END
```

```
DISCRETE SYSTEM reg
Input yr y
Output u
State I
New nl
Tsamp ts
ts=t+h
v=k*e+I
u=sat(v.0.1)
nl=I+k*h*e/Ti+u-v
k:1
h:0.1
END
```


Simulink 1991 the Ultimate Block Diagram Tool

- Mimics the analog computer with more general blocks
- Each block a state model



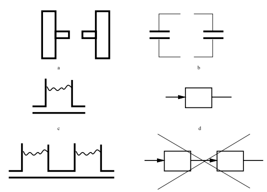
- MATLAB, Stateflow
- Granularity and Structuring
- Graphical aggregation and disaggregation
- Much manual manipulation from physics to blocks
- Neither formal syntax nor formal semantics

But!!

States may disappear when system are interconnected – warning algebraic loop!

Composition does not work!

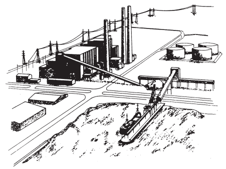
Much manual labor to go from physics to block diagrams

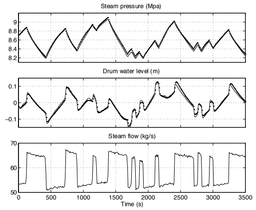


Lesson 1: Block diagrams not suitable for physical modeling
Lesson 2: Don't stick to a paradigm based on old technology when new technology emerges!!

1. Introduction
2. Block diagram modeling
3. **Equation-based modeling**
4. Summary

Boiler Control at LTH





- Experiments, modeling, system identification
- Eklund Linear DrumBoiler-Turbine Models 1971
- Lindahl Design and Simulation of a Coordinated Drum Boiler-Turbine Controller Dec 1976

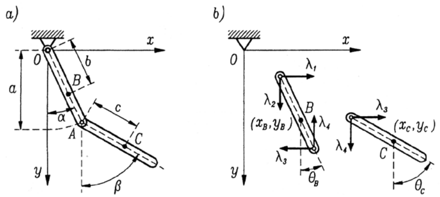
Inspiration

- Bond Graphs Henry Paynter MIT 1961
  Excellent if there is one dominating balance equation. Difficult to deal with many balances.
- Circuit theory
  Two ports systems: Kirchoffs current and voltage law
  Differential algebraic systems DAE Gear 1971 & Petzold
  Spice Peterson Berkeley 1973
  Good solution for circuits. Attempts at generalizations: System dynamics, through and across variables
- Multi-body systems: Adams, SolidWorks, ….
- Chemical Engineering: Complex plants, no dynamics, optimization

Good Old Physical Modeling

- Divide a system into subsystems
- Define interfaces and account for interactions
- Write mass, momentum and energy balances
- Add constitutive material equations
- Lumped parameters models DAE not ODE
- Symbolic computations DAE
- Connecting subsystems (many trivial equations)

Mechanical Systems

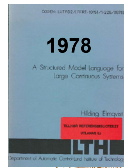


- Split into subsystems (free body diagrams)
- Write equations of motion for each subsystem
- Add constraints to describe connections

Origins of Equation-based Modeling LCCC Sept 2012

Elmqvist's PhD Thesis

- Strong industrial interest in SIMNON, demands for extensions, matrices, hierarchies. Is this a good thesis topic? Transpiration/inspiration?
- More interesting to make a modeling language
- Modeling paradigm – balance equations
- Object orientation (Simula)
- Symbolic computations DAE
- Boiler model worked
- Great ideas but premature
- Demanding application useful



www.control.lth.se/Publication/elm78dis.html

Origins of Equation-based Modeling LCCC Sept 2012

Model Manipulations

- Eliminate redundant variables
- Use graph algorithms to reduce to lower block diagonal form LBD
- Solve linear blocks analytically
- Use tearing to generate iterative solution for nonlinear blocks
- Generate code for finding equilibria
- Generate code for DAE solvers
- Connect to optimizers
- Generate inverse models for feedforward control (reverse causality) e.g. computed torque
- Generate linear models for control design

Origins of Equation-based Modeling LCCC Sept 2012

Omola-Omsim

- Work on CACE stopped around 1980 because of FORTRAN and MATLAB
- New research project 1990 Object Oriented Modeling and Simulation: Sven Erik Mattsson, Mats Andersson, Bernt Nilsson, Dag Bruck, Jonas Eborn, Hubertus Tummescheit, Johan Åkesson
- Experiments with OO in Lisp & KEE
- C++ for object orientation
- Language (Omola) and simulator (OmSim)
- Extensive symbolic manipulation (Mattsson)
- Jmodelica.org Optimica

Origins of Equation-based Modeling LCCC Sept 2012

Modelica

- Intensive interaction with Dynasim 1991
- ESPRIT Simulation in Europe, Lund Sept 1996
- COSY meeting Lund Sept 5-7, 1996
- European groups: 23 participants, 17 talks by groups from Dynasim Lund, ETH Zurich, INRIA Paris, DLR Munich, VTT Helsinki, Imperial College London,LTH Lund, RWTH Aachen and universities in Barcelona,, Groningen, Valencia, Wien
- Formation of the Modelica language group
- First Modelica language specification Sept 1997
- 7 Modelica compilers at 9th Modelica conf 2012

Original Language Team

Hilding Elmqvist, Dynasim AB, Lund, Sweden
Fabrice Boudaud, Gaz de France,
Jan Broenink, University of Twente, Netherlands
Dag Bruck, Dynasim AB, Lund, Sweden
Thilo Ernst, GMD-FIRST, Berlin, Germany
Peter Fritzon, Linköping University, Sweden
Alexandre Jeandel, Gas de France
Kaj Juslin, VTT, Finland
Matthias Klose, Technical University of Berlin, Germany
Sven Erik Mattsson, Lund University, Sweden
Martin Otter, DLR, Oberpfaffenhofen,Germany
Per Sahlin, BrisData, Stockholm, Sweden
Hubertus Tummescheit, DLR Cologne, Germany
Hans Vangheluwe, University of Gent, Belgium

1. Introduction
2. Block diagram modeling
3. Equation-based modeling
4. **Summary**

Many Views on Modeling

- Engineering: Free body diagrams, circuit diagrams, block diagrams, P&I diagrams
- Behavioral systems Willems 1981 (CSM 2007)
- Physics: Mass, energy, momentum balances constitutive material equations
- Mathematics: ODE, DAE, PDE
- Computer Science: Languages, datastructures, programming, imperative, declarative
- Block Diagram Modeling: Causal modeling, imperative
- Equation-Based Modeling: Acausal, declarative,

Equation-based Modeling

- Has come a long way
- Serious industrial use
- 9th Modelica conference, several commercial compilers
- Strong potential for education
- Lower the entrance barrier
- Many challenges
- Much work remains
- Step back and think!
- This workshop and …

Challenges

- Is it time to sit it back and think about fundamentals?
- Make Modelica an international standard, compliance checking!
- Make it widely used!
- More than simulation
- Embedded systems
- Lower entrance barrier
- The tool chain

Modeling

- Solomon Golomb: Mathematical models – Uses and limitations. Aeronautical Journal 1968

Solomon Wolf Golomb (1932) mathematician and engineer and a professor of electrical engineering at the University of Southern California. Best known to the general public and fans of mathematical games as the inventor of polyominoes, the inspiration for the computer game Tetris. He has specialized in problems of combinatorial analysis, number theory, coding theory and communications.

Golomb On Modeling

- Don't apply a model until you understand the simplifying assumptions on which it is based and can test their applicability. Validity ranges
- Distinguish at all times between the model and the real world. You will never strike oil by drilling through the map!
- Don't expect that by having named a demon you have destroyed him
- The purpose of notation and terminology should be to enhance insight and facilitate computation – not to impress or confuse the uninitiated

ASSIMULO – A PYTHON PACKAGE FOR SOLVING DIFFERENTIAL EQUATION WITH INTERFACE TO EQUATION BASED LANGUAGES

Claus Führer, Centre of Mathematical Sciences, Lund University

We present a Python package which gives access to state-of-the art industrial differential equation algorithms in C or FORTRAN and which is open for experimental methods in Python. The interesting feature of Assimula is that it comes with a specially designed problem class to import models (=differential equations) from JMODELICA. Such an equation based modeling language can provide much more information to the solver than just the problem description itself. Equation coupling information, information about equation type, discontinuities and others can be used to improve and control efficiently the solution process. The talk includes even a wish-list for additional language constructions.

Assimulo - a Python package for solving differential equations with interface to equation based languages

Lunds universitet · Modelon

Christian Andersson, Claus Führer

Johan Åkesson

LCCC workshop Lund
September 2012

Let's move the focus ...

Problem — User friendly description → Mathematical description:

$\dot{x} = f(t, x)$

$F(t, x, \dot{x}) = 0$

$x(0) = x_0$

$\dot{x}(0) = \dot{x}_0$

$t \in [t_0, t_e]$

→ Solver:

$$\sum \alpha_i x_{n-1} - h \sum \beta_i \dot{x}_{n-1} = 0$$

ODE and DAE solvers in two disjoint worlds

Industrial Simulation Tasks

- highly complex models
- high robustness standards
- high documentation standards
- long life cycle

⟶ *one or two ODE/DAE packages meet these requirements.*

Academic Simulation Tasks

- a few, low scale test models
- lab standard quality (validation of concept)
- good analyzed algorithms, poor code documentation
- short life cycle, often coupled on individual career steps.

⟶ *dozen of codes produced (and forgotten) this way.*

ODE and DAE solvers in two disjoint worlds

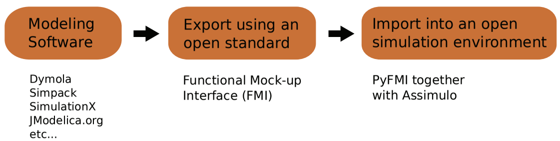
The harsh requirement that a useful numerical method must permit an efficient, robust, and reliable implementation has withered the beautiful flowers which bloomed on thousands of journal pages.

Hans Stetter in:
Mathematics of computation, 1943-1993: a half-century of computational mathematics : Mathematics of Computation 50th Anniversary Symposium, August 9-13, 1993, Vancouver, British Columbia

... highly valid still today.

Motivation

- Give the academic world access to complex models ⟶ FMI
- Give the industrial world access to a variety of ODE/DAE codes (even experimental ones): ⟶ ASSIMULO
- Give students in scientific computing an intuitive access to industrial standard solvers: ⟶ ASSIMULO



Functional Mock-up Interface (FMI)

FMI is an open interface for model exchange with the idea that tools may generate and exchange dynamic system models.

The **FMI** supports model defined as discontinuous ordinary differential equations.

- **Model interface** The equations are evaluated and the model interaction is performed by standardized *C* functions.
- **Model description** The variable information of the model is contained in an XML-file.
- **Additional data** Model data, such as tables and maps may also exists.

⇒ Talk by Torsten Blochwitz on Wednesday.

Assimulo is written in Python, why?

Benefits of using Python:

- Open-source language
- Interpreted
- Object-oriented
- Many freely available packages
  - NumPy
  - SciPy
  - Matplotlib
  - Cython
- Highly flexible for interfacing to *C*, FORTRAN ...
- Ideal in teaching.

ASSIMULO

Python workbench for simulation of ordinary differential equations.

The intention is to provide a common high-level interface for a variety of different solvers.

Supports

- problems formulated as first or second order ordinary differential equations
- problems formulated as implicit ordinary differential equations including overdetermined problems.

ASSIMULO, problem formulations

- Explicit hybrid ODEs

$$\dot{y} = f(t, y, sw), \quad y(t_0) = y_0, \quad sw(t_0) = sw_0$$

- Implicit hybrid ODEs (also called DAEs)

$$F(t, y, \dot{y}, sw) = 0, \quad y(t_0) = y_0, \quad \dot{y}(t_0) = \dot{y}_0, \quad sw(t_0) = sw_0$$

- Mechanical systems in second order explicit ODE form

$$\ddot{p} = M(p)^{-1} f(t, p, \dot{p})$$

- Mechanical systems in (overdetermined) implicit ODE form

$$\begin{aligned} \dot{p} &= v \\ M(p)\dot{v} &= f(t, p, v) - G^T(p)\lambda \\ 0 &= g_{\text{constr}}(p) \\ 0 &= G(p)v \end{aligned}$$

- Delay (retarded) differential equations.

ASSIMULO, overview

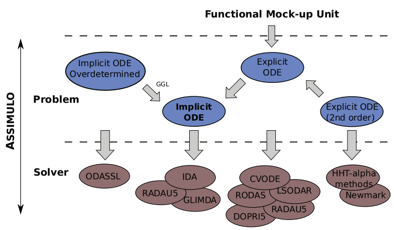


Figure : Connection between the different problem formulations and the different solvers available in ASSIMULO. The connection of the Functional Mock-up Interface to ASSIMULO is also shown.

ASSIMULO, solvers

Currently, solvers written in Python, FORTRAN and *C* are available.

- **IDA** - Multistep method for DAEs
- **CVode** - Multistep methods for ODEs
- **ODASSL** - Multistep methods for overdetermined DAEs
- **RADAU5** - Runge–Kutta method for DAEs
- **GLIMDA** - General linear methods methods for DAEs
- and we are working on a **"solver museum"** (oldest code in restoration 1983).

IDA and CVode are production quality solvers from the **SUNDIALS** suite.

Simple example workflow

Make a problem

```
def rhs(t,y):
  A = array([[0, 1], [-2, -1]])
  yd = N.dot(A, y)
  return yd

y0 = array([1.0, 1.0])
t0 = 0.0

linmodel = Explicit_Problem(rhs, y0, t0)
```

Create a solver instance

```
sim = CVode(linmodel)
```

... and simulate

```
t, y = sim.simulate(tfinal)
```

Assimulo can be quite verbose...

```
Final Run Statistics: Linear Test ODE

 Number of Error Test Failures             = 4
 Number of F-Eval During Jac-Eval          = 0
 Number of Function Evaluations            = 153
 Number of Jacobian Evaluations            = 0
 Number of Nonlinear Convergence Failures  = 0
 Number of Nonlinear Iterations            = 149
 Number of Root Evaluations                = 0
 Number of Steps                           = 84

Solver options:

 Solver                  :  CVode
 Linear Multistep Method :  Adams
 Nonlinear Solver        :  FixedPoint
 Maxord                  :  12
```

Controlling the method

```
sim.atol=N.array([1.0,0.1])*1.e-5
sim.rtol=1.e-8
sim.maxord=3
sim.discr='BDF'
sim.iter='Newton'
```

Discontinuities – a Continuous Challenge

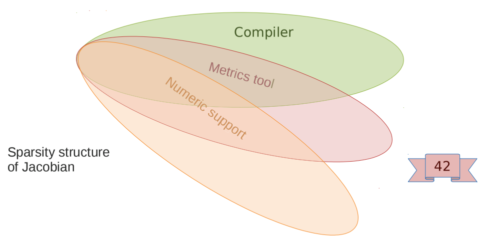
```
class Extended_Problem(Explicit_Problem):
  #Sets the initial conditions directly into the problem
  y0 = [0.0, -1.0, 0.0]
  sw0 = [False,True,True]

  #The right-hand-side function (rhs)
  def rhs(self, t, y, sw):
      ....

  #The event function
  def state_events(self, t, y, sw):
     event_0 = y[1] - 1.0
      ...
     return array([event_0,event_1,event_2])

  #Responsible for handling the events.
  def handle_event(self, solver, event_info):
    event_info = event_info[0]
    while True: #Event Iteration
      self.event_switch(solver, event_info) #Turns the swi
       ...
```

Languages have the potential to inform

- Are there discontinuities?
- State/Time events?
- Are there linear components?
- What are differential, what are algebraic variables? ("loop closure" conditions versus algebraic equations)
- Derivatives?

The compiler might know more

Why extensible compilers?



(Sorry Görel for changing your slide ...)

Thank you!

... and feel free to try it out!

- **Assimulo** www.assimulo.org
- **PyFMI** www.pyfmi.org

Plans/ideas/wishes for the future

- Would like to stimulate to open the **FMI** for a wider range of problem formulations - higher index DAES(?)
- Continue to expand the solvers available in ASSIMULO
  - Work on the museum.
  - Introduce problem formulation for delay differential equations
  - Generalize solvers for discontinuity handling
- Potentials of language/compiler aided numerics.
- Automatic differentiation: a separate tool or an integrated part of the language-solver chain?

CASADI: A TOOL FOR AUTOMATIC DIFFERENTIATION AND SIMULATION-BASED NONLINEAR PROGRAMMING

Moritz Diehl, Electrical Engineering Department and Optimization in Engineering Center OPTEC KU Leuven

We present CasADi, an open-source symbolic environment for simulation based nonlinear programming and automatic differentiation (AD). Casadi offers a level of abstraction that is higher than conventional AD tools and is in particular designed to enable calls to solvers of initial-value problems in differential-algebraic equations (DAE) within nonlinear programming formulations, with derivative information efficiently calculated through automatic formulation of the corresponding forward and adjoint sensitivity equations.

In this talk, we give an overview of the tool, with a focus on the AD approach and the symbolic environment. This environment allows users to formulate problems in a high-level language such as Python, but solve it with the speed of optimized C-code thanks to fast interpreters and just-in-time compilation. We also show how optimal control problems formulated in the physical modelling language Modelica can be imported into the symbolic environment. Joint work with Joel Andersson, Joris Gillis, and Johan Akesson.

CasADi: A Tool for Automatic Differentiation and Simulation-Based Nonlinear Programming

Moritz Diehl*
with Joel Andersson*, Joris Gillis*, Johan Akesson**
*OPTEC, KU Leuven, Belgium
**Lund University / Modelon

LCCC, Sept 20, 2012

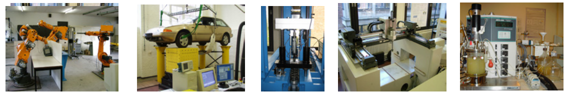
KATHOLIEKE UNIVERSITEIT LEUVEN · OPTEC K.U.Leuven Optimization in Engineering Center · EMBOCON · ERC HIGHWIND SIMULATION, OPTIMIZATION & CONTROL OF HIGH-ALTITUDE WIND POWER GENERATORS · ITN SADCO Initial Training Network Sensitivity Analysis for Deterministic Controller Design

OPTEC - Optimization in Engineering Center

***Center of Excellence of KU Leuven**, since 2005*
*70 people, working jointly on **methods and applications of optimization,** in 5 departments:*

- Electrical Engineering
- Mechanical Engineering
- Chemical Engineering
- Computer Science
- Civil Engineering

Many real world applications at OPTEC...



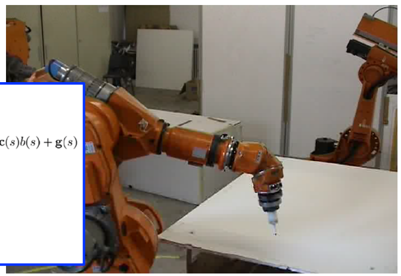
OPTEC Research Example: Time Optimal Robot Motion

Robot shall write as fast as possible.
Global solution found in 2 ms due to ***convex reformulation***

IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 54, NO. 10, OCTOBER 2009

Time-Optimal Path Tracking for Robots: A Convex Optimization Approach

Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl



$$
\begin{aligned}
\min_{a(\cdot),b(\cdot),\boldsymbol{\tau}(\cdot)} \quad & \int_0^1 \frac{1}{\sqrt{b(s)}}ds \\
\text{subject to} \quad & \boldsymbol{\tau}(s) = \mathbf{m}(s)a(s) + \mathbf{c}(s)b(s) + \mathbf{g}(s) \\
& b(0) = \dot{s}_0^2 \\
& b(1) = \dot{s}_T^2 \\
& b'(s) = 2a(s) \\
& b(s) \geq 0 \\
& \underline{\boldsymbol{\tau}}(s) \leq \boldsymbol{\tau}(s) \leq \overline{\boldsymbol{\tau}}(s) \\
& \text{for } s \in [0,1].
\end{aligned}
$$

Overview

- Optimization in Engineering Center OPTEC
- **State of the Art in Optimal Control Algorithms (ACADO)**
- CasADi: A Framework to WRITE Optimal Control Algorithms

Optimal Control Problem in Continuous Time

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad \int_0^T L(x(t),u(t))\,dt \; + \; E(x(T))$$

subject to

$$\begin{aligned} x(0) - x_0 &= 0, && && \text{(fixed initial value)} \\ \dot{x}(t) - f(x(t),u(t)) &= 0, && t \in [0,T], && \text{(ODE model)} \\ h(x(t),u(t)) &\geq 0, && t \in [0,T], && \text{(path constraints)} \\ r(x(T)) &= 0 && && \text{(terminal constraints).} \end{aligned}$$

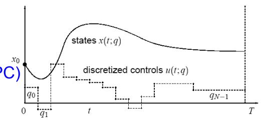
How to solve these nonlinear problems reliably and fast?

Sequential Approach (Single Shooting): Eliminate States

$$\underset{u}{\text{minimize}} \quad \sum_{i=0}^{N-1} L_i(\tilde{x}_i(u), \tilde{z}_i(u), u_i) \; + \; E(\tilde{x}_N(u))$$

subject to

$$\begin{aligned} h_i(\tilde{x}_i(u), \tilde{z}_i(u), u_i) &\leq 0, \quad i = 0, \ldots, N-1, \\ r(\tilde{x}_N(u)) &\leq 0. \end{aligned}$$



Pros:
- Only control degrees of freedom (for NMPC)
- Can couple with "Vanilla NLP" solver

Cons:
- Sparsity of problem lost
- Unstable systems cannot be treated

Historically first "direct" approach ("single shooting", Sargent&Sullivan 1978)

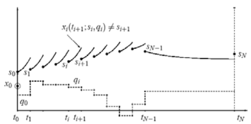
Simultaneous Approach: Keep States in NLP

INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL
9TH WORLD CONGRESS
BUDAPEST, HUNGARY
JULY 2-6 1984

A MULTIPLE SHOOTING ALGORITHM FOR DIRECT SOLUTION OF OPTIMAL CONTROL PROBLEMS*

Hans Georg Bock and Karl J. Plitt

Institut für Angewandte Mathematik, SFB 72, Universität Bonn, 5300 Bonn, Federal Republic of Germany



Variants:
Direct Multiple Shooting and Collocation

Pros:
- Sparsity of problem kept
- Unstable systems can be treated, nonlinearity reduced

Cons:
- Large scale problems
- Need to develop (or use) structure exploiting NLP solver

Nonlinear Program (NLP) in Multiple Shooting

$$\underset{x,z,u}{\text{minimize}} \quad \sum_{i=0}^{N-1} L_i(x_i, z_i, u_i) \; + \; E(x_N)$$

subject to

$$\begin{aligned} x_0 - \bar{x}_0 &= 0, \\ x_{i+1} - f_i(x_i, z_i, u_i) &= 0, \quad i = 0, \ldots, N-1, \\ g_i(x_i, z_i, u_i) &= 0, \quad i = 0, \ldots, N-1, \\ h_i(x_i, z_i, u_i) &\leq 0, \quad i = 0, \ldots, N-1, \\ r(x_N) &\leq 0. \end{aligned}$$

Structured parametric Nonlinear Program
- Initial Value $\bar{x}_0$ is often not known beforehand ("online data" in NMPC)
- Discrete time dynamics from ODE simulation (we will need sensitivities!)

Sequential Convex Programming (SCP)

- Summarize problem as $\min_{x \in \mathbb{R}^n} f(x)$ s.t. $g(x) + M\xi = 0,\ x \in \Omega,$ with convex f and Ω

Step 1: Linearize nonlinear constraints at x^k to obtain convex problem:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad g(x^k) + g'(x^k)(x - x^k) + M\xi = 0,\ x \in \Omega.$$

Step 2: Solve convex problem to obtain next iterate.
Obtain new value of parameter ξ and go to step 1)

- Convergence to (and tracking of) local minima under mild assumptions [1]

[1] Tran Dinh, Savorgnan, Diehl: Adjoint-based predictor-corrector SCP for parametric nonlinear optimization. *SIAM Journal on Optimization* (in print)

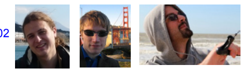
ACADO Toolkit [1]

- ACADO = Automatic Control and Dynamic Optimization

- Open source (LGPL) C++: www. acadotoolkit. org
- Implements direct multiple shooting [2] and real-time iterations [3]
- User interface close to mathematical syntax
- *Automatic C-Code Export for Microsecond Nonlinear MPC* [4]

- Developed at OPTEC by B. Houska, H.J. Ferreau, M. Vukov, ...
- ~3000 downloads since first release in 2009

[1] Houska, Ferreau, D., *OCAM*, 2011
[2] Bock, Plitt, *IFAC WC*, 1984
[3] D., Bock, Schloder, Findeisen, Nagy, Allgower, *JPC*, 2002
[4] Houska, Ferreau, D., *Automatica*, 2011



Rocket Example in ACADO Language

Mathematical Formulation:

$$\underset{s(\cdot),v(\cdot),m(\cdot),u(\cdot),T}{\text{minimize}} \quad T$$

subject to

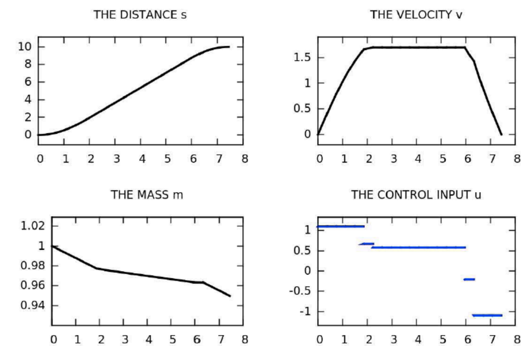
$$\begin{aligned} \dot{s}(t) &= v(t) \\ \dot{v}(t) &= \frac{u(t) - 0.2\,v(t)^2}{m(t)} \\ \dot{m}(t) &= -0.01\,u(t)^2 \end{aligned}$$

$$\begin{aligned} s(0) &= 0 \quad s(T) = 10 \\ v(0) &= 0 \quad v(T) = 0 \\ m(0) &= 1 \end{aligned}$$

$$\begin{aligned} 0 &\le v(t) \le 1.7 \\ -1.1 &\le u(t) \le 1.1 \\ 5 &\le T \le 15 \end{aligned}$$

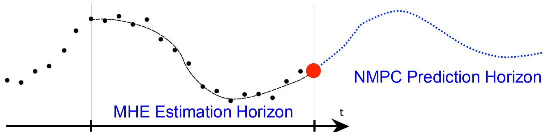
```
DifferentialState                s,v,m;
Control                              u;
Parameter                            T;
DifferentialEquation      f( 0.0, T );

OCP ocp( 0.0, T );
ocp.minimizeMayerTerm( T );

f << dot(s) == v;
f << dot(v) == (u-0.2*v*v)/m;
f << dot(m) == -0.01*u*u;
ocp.subjectTo( f );

ocp.subjectTo( AT_START, s ==  0.0 );
ocp.subjectTo( AT_START, v ==  0.0 );
ocp.subjectTo( AT_START, m ==  1.0 );
ocp.subjectTo( AT_END  , s == 10.0 );
ocp.subjectTo( AT_END  , v ==  0.0 );

ocp.subjectTo(  0.0 <= v <=  1.7   );
ocp.subjectTo( -1.1 <= u <=  1.1   );
ocp.subjectTo(  5.0 <= T <= 15.0   );

OptimizationAlgorithm algorithm(ocp);
algorithm.solve();
```

ACADO Results Plot (after few milliseconds)



NMPC Practice: Estimation AND Optimization

- Moving Horizon Estimation (MHE): Get State by Least Squares Optimization
- Nonlinear Model Predictive Control (NMPC): Solve Optimal Control Problem

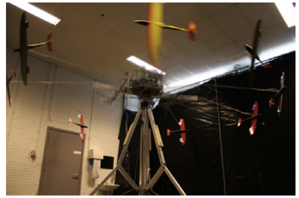


Gauss-Newton in ACADO:

ocp.minimizeMayerTerm()→ ocp.minimizeLSQ();

ACADO Code Generation for Tethered Airplanes

- 22 states, nonlinear, unstable
- 2 controls
- 1 s horizons in past / future

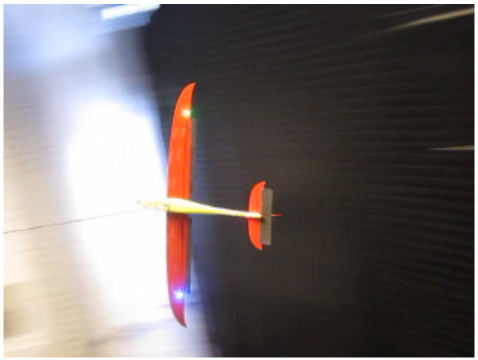


4 ms execution time for one optimization problem (on i7 2.5 GHz)

[Note: NMPC today 100 000x faster than 1997]

ERC HIGHWIND SIMULATION, OPTIMIZATION & CONTROL OF HIGH-ALTITUDE WIND POWER GENERATORS

MHE+NMPC Experiments (Aug 22, 2012)



Overview

- Optimization in Engineering Center OPTEC
- State of the Art in Optimal Control Algorithms (ACADO)
- **CasADi: A Framework to WRITE Optimal Control Algorithms**

Optimal Control Problem (OCP) Solvers

Two implementation approaches

- Write/use a general-purpose OCP solver
  - Examples: MUSCOD-II, ACADO Toolkit, DyOS, DIRCOL
  - + Easy to set up for the average user
  - + Can be very efficient for medium size problems
  - – Many OCPs cannot be formulated
- Write special-purpose OCP solvers
  - OCP→NLP using algebraic modelling language
  - + Full control of NLP formulation, easier to *extend*
  - – So far only for collocation methods
- Both approaches taken at OPTEC using two in-house software tools
  - ACADO Toolkit: A general-purpose OCP solver for NMPC
  - CasADi: A *framework for writing OCP solvers*

Computer Algebra System for Algorithmic Differentiation

CasADi

What is CasADi?

A framework for C++, Python and Octave for quick, yet efficient, implementation of algorithms for numeric optimization

In particular

Facilitates OCP→NLP transcription for collocation methods *and* shooting methods (e.g. single-shooting method in 30 lines of code)

Permissive open-source license (LGPL)

`www.casadi.org`

CasADi

Main components of CasADi

- A symbolic framework with state-of-the-art algorithmic differentiation (all eight flavours of AD)
- Interfaces to other tools; NLP solvers, ODE/DAE integrators, ...
- In-house tools; NLP solvers, ODE/DAE integrators, ...
- Framework for import and symbolic reformulation of OCPs from Modelica

Implementation

- Written in self-contained C++ code
- Full-featured front-ends to Python and Octave using SWIG

CasADi

Main developers


Joel Andersson


Joris Gillis

CasADi

An illustrating example

Drive a Van der Pol oscillator to the origin with minimal control effort:

$$\begin{aligned}
\underset{v,p,u}{\text{minimize:}} \quad & \int_0^{t_f} u(t)^2\,dt \\
\text{subject to:} \quad & \dot{x}(t) = \begin{bmatrix} \dot{v} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} (1-p^2)\,v - p + u \\ v \end{bmatrix}, \quad t \in [0, t_f] \\
& v(0) = 0, \quad p(0) = 1, \\
& v(t_f) = 0, \quad p(t_f) = 0 \\
& -0.75 \le u(t) \le 1.0, \quad t \in [0, t_f]
\end{aligned}$$

Solve with a direct-single shooting method.

CasADi

Step 1: Formulate symbolic expression ODE in CasADi

- The ODE:

$$\begin{bmatrix} \dot{v} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} (1-p^2)\,v - p + u \\ v \end{bmatrix},$$

- Can be formulated in CasADi–Python:

```
# Declare variables
u  = ssym("u")
v  = ssym("v")
p  = ssym("p")
```

```
# ODE right hand side
vdot = (1 - p*p)*v - p + u
pdot = v
```

- Syntax ≈ Matlab Symbolic Toolbox
- ODE can also be imported from **Modelica**

CasADi

Step 2: Create ODE function

- These expressions define the ODE rhs *function* $f : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$:

```
f = SXFunction( \
     daeIn(   x = vertcat([v,p]), p = u), \
     daeOut(ode = vertcat([vdot,pdot])))
```

- Creating a function means *topologically sorting* the expression graph
- Function can be evaluated:
  - In the CasADi interpretor: numerically *or* symbolically
  - By generating and compiling C-code
  - Through just-in-time compilation (using *LLVM* framework)
- Derivatives in CasADi are calculated by *automatic differentiation*

CasADi

Step 3: Formulate discrete time dynamics

- Assume a piecewise constant control with 20 intervals and let t_f be 10 s.

```
nk = 20    # Control discretization (uniform)
th = 10.0  # Length of the time horizon
```

- Get the discrete time dynamics by allocating an ODE integrator instance, e.g. using CasADi's interface to Sundials:

```
f_d = CVodesIntegrator(f)
f_d.setOption("tf",th/nk) # Interval length
f_d.init()
```

- Integrators in CasADi are differentiable functions in CasADi and can be differentiated an arbitrary number of times
- Derivatives calculated through *forward/adjoint sensitivity analysis*

CasADi

Step 4: Formulate NLP

The integrator allows us to form an expression for the state at the final time:

```
U = msym("U",nk)      # Controls for each interval
X0 = [0,1] # The initial state
# Build a graph of integrator calls
X  = X0
for k in range(nk):
  X,_,_,_ = I.call([X,U[k]])
```

this defines NLP objective functions and constraints:

```
# Objective function: ||U||^2
F = MXFunction([U],[mul(U.T,U)])

# Terminal constraints: x=[0,0]
G = MXFunction([U],[X])
```

CasADi

Step 5: Solve NLP

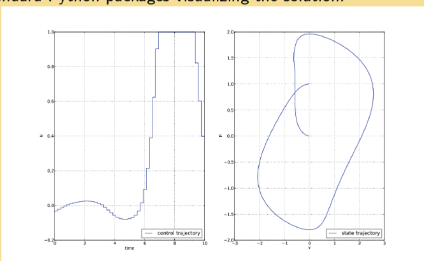
Solve NLP by using one of the interfaced NLP solvers, e.g. IPOPT:

```
import numpy       # Standard linear algebra routines

# Allocate an NLP solver
solver = IpoptSolver(F,G)
solver.init()

# Set bounds and initial guess
solver.setInput(-0.75*numpy.ones(nk), NLP_LBX)
solver.setInput(1.0*numpy.ones(nk), NLP_UBX)
solver.setInput(numpy.zeros(nk),NLP_X_INIT)
solver.setInput(numpy.zeros(2),NLP_LBG)
solver.setInput(numpy.zeros(2),NLP_UBG)

# Solve the problem
solver.solve()
```

CasADi

Step 6: Visualize solution

Use standard Python packages visualizing the solution:



CasADi Users

Other OCP methods successfully implemented using CasADi

- Direct collocation (J. Andersson, J. Åkesson & F. Magnusson, M. Zanon & S. Gross, J. Steinberg, J. Gillis . . .)
- Direct multiple-shooting (J. Andersson, K. Gevelen, J. Frasch)
- Distributed multiple-shooting (A. Kozma & C. Savorgnan)
- Pseudospectral optimization (C. Andersson)

Benchmarking CasADi vs AMPL Solver Library

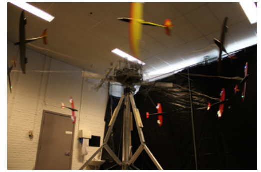
Problem	Dimensions		Time ASL [s]		Time CasADi [s]		Diff.
	#var	#con	Total	AD	Total	AD	
gpp	250	498	0.492	0.272	0.500	0.264	-3 %
reading1	10001	5000	0.712	0.408	0.306	0.104	-76 %
porous2	4900	4900	1.916	0.188	1.736	0.036	-81 %
orthrgds	10003	5000	0.949	0.568	0.512	0.164	-71 %
clnlbeam	1499	1000	0.776	0.184	0.784	0.184	0 %
svanberg	5000	5000	2.492	0.520	2.300	0.272	-48 %
orthregd	10003	5000	0.332	0.208	0.160	0.060	-71 %
trainh	20000	10002	3.932	1.984	2.804	0.896	-55 %
orthrgdm	10003	5000	0.328	0.208	0.156	0.068	-67 %
dtoc2	5994	3996	0.296	0.124	0.224	0.048	-61 %

Benchmarking

- CasADi VM outperformed ASL VM by a factor 2 on average
- Most of the time spent in linear solver anyway
- Note: ≈5x faster still with C-codegen or just-in-time

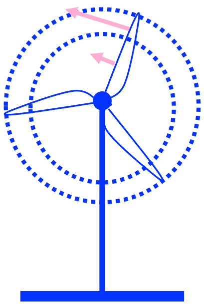
CasADi Usage in Leuven: Complex Plane Orbits

- Within ERC Project HIGHWIND, running from 2011-2016



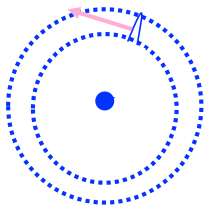
ERC HIGHWIND
SIMULATION, OPTIMIZATION & CONTROL OF HIGH-ALTITUDE WIND POWER GENERATORS

What is the Optimal Wind Turbine ?



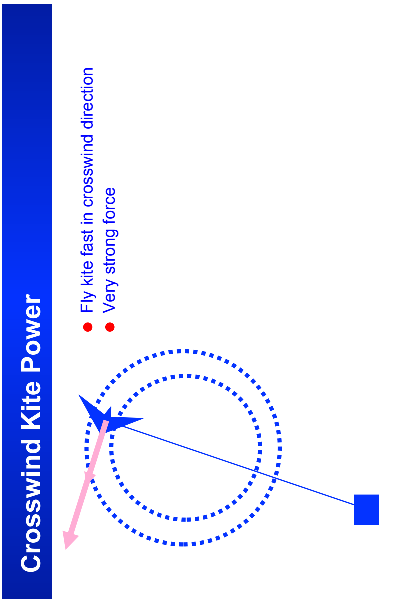
- Due to high speed, wing tips are ***most efficient*** part of wing
- Best winds are in high altitudes

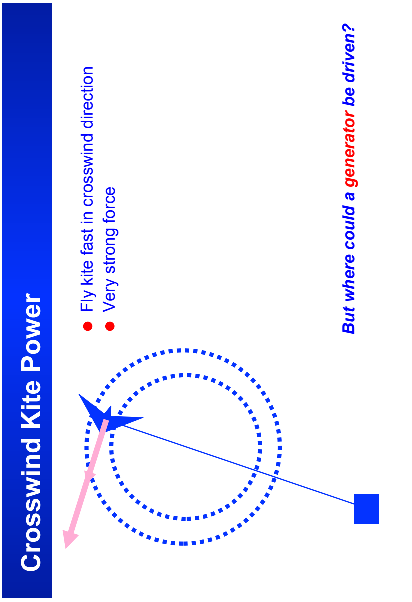
What is the Optimal Wind Turbine ?

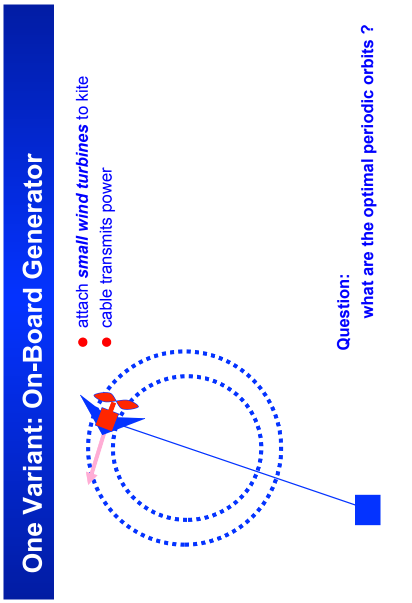


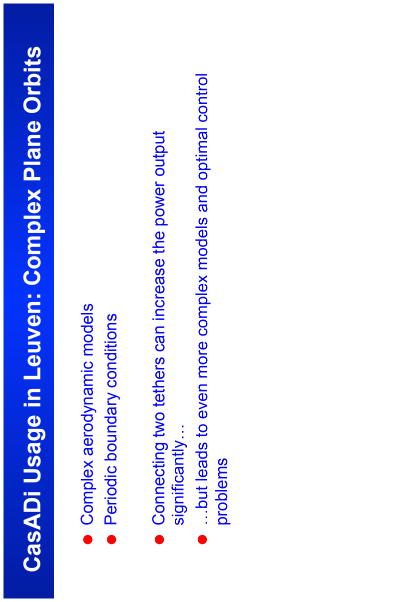
- Due to high speed, wing tips are ***most efficient*** part of wing
- Best winds are in high altitudes

Could we construct a wind turbine with only wing tips and generator?


Crosswind Kite Power
Fly kite fast in crosswind direction
Very strong force

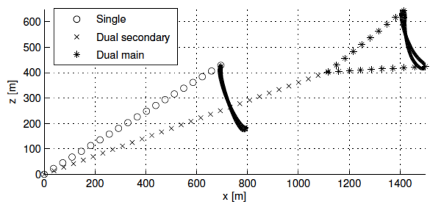

Crosswind Kite Power
Fly kite fast in crosswind direction
Very strong force
But where could a generator be driven?


One Variant: On-Board Generator
attach small wind turbines to kite
cable transmits power
Question:
what are the optimal periodic orbits ?


CasADi Usage in Leuven: Complex Plane Orbits
Complex aerodynamic models
Periodic boundary conditions
Connecting two tethers can increase the power output significantly…
…but leads to even more complex models and optimal control problems

Single vs. Dual Airfoils: Optimal Large System

Trajectory for $500\mathrm{m}^2$ of Airborne Surface



Complex OCPs solved with CasADi, Collocation, IPOPT, from [Zanon et al., submitted]

Visualization of Single vs. Dual Airfoils

Summary

- Optimal Control Tools now 100000x faster than 1997, and ACADO Code Generation is currently tested in a variety of fast real world applications (cranes, airplanes, vehicles, induction motors, ...)
- But non-standard problems need non-standard solvers: CasADi allows the user to easily write competitive state-of-the-art optimal control algorithms specifically designed for one problem class
- CasADi distributed under permissive LGPL license and used by a growing number of people in and outside Leuven (e.g. Jmodelica)

CasADi www.casadi.org

Appendix

CasADi Performance

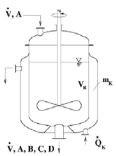
Benchmarking using CUTEr

- 10 NLPs from Bob Vanderbei's AMPL translation of CUTEr
- AMPL used to parse/pre-optimize AMPL models
- Solved using IPOPT 3.10 with MA27 as linear solver in two ways
  - Using AMPL Solver Library's (ASL) interface to IPOPT
  - Using CasADi's .nl import and interface to IPOPT
  - Only virtual machines (VM) for both tools, no codegen

Complete CasADi Code for OCP Solution

```
from casadi import *

nk = 50    # Control discretization
th = 10.0  # End time

# Declare variables
u  = ssym("u")
v  = ssym("v")
p  = ssym("p")
x  = vertcat([v,p])

# ODE right hand side
vdot = (1 - p*p)*v - p + u
pdot = v
xdot = vertcat([vdot,pdot])

# DAE residual function
f = SXFunction(daeIn(x=x,p=u),daeOut(ode=xdot))

# Create an integrator
I = CVodesIntegrator(f)
I.setOption("tf",th/nk) # final time
I.init()

# All controls (use matrix graph)
U = msym("U",nk) # nk-by-1 symbolic variable

# The initial state (x = [0,1])
X0 = [0,1]

# Build a graph of integrator calls
X  = X0
for k in range(nk):
  X,_,_,_ = I.call([X,U[k]])

# Objective function: ||U||^2
F = MXFunction([U],[mul(U.T,U)])

# Terminal constraints: x=[0,0]
G = MXFunction([U],[X])

# Allocate an NLP solver
solver = IpoptSolver(F,G)
solver.init()

# Set bounds and initial guess
solver.setInput(-0.75*ones(nk), NLP_LBX)
solver.setInput(1.0*ones(nk), NLP_UBX)
solver.setInput(zeros(nk),NLP_X_INIT)
solver.setInput(zeros(2),NLP_LBG)
solver.setInput(zeros(2),NLP_UBG)

# Solve the problem
solver.evaluate()
```

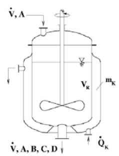
ACADO Code Generation for Benchmark CSTR



$$\dot{c}_A(t) = u_1(c_{A0} - c_A(t)) - k_1(\vartheta(t))c_A(t) - k_3(\vartheta(t))(c_A(t))^2$$
$$\dot{c}_B(t) = -u_1 c_B(t) + k_1(\vartheta(t))c_A(t) - k_2(\vartheta(t))c_B(t)$$
$$\dot{\vartheta}(t) = u_1(\vartheta_0 - \vartheta(t)) + \frac{k_w A_R}{\rho C_p V_R}(\vartheta_K(t) - \vartheta(t)) - \frac{1}{\rho C_p}[k_1(\vartheta(t))c_A(t)H_1 + k_2(\vartheta(t))c_B(t)H_2 + k_3(\vartheta(t))(c_A(t))^2 H_3]$$
$$\dot{\vartheta}_K(t) = \frac{1}{m_K C_{PK}}(u_2 + k_w A_R(\vartheta(t) - \vartheta_K(t))).$$

CSTR Benchmark by [Klatt, Engell, Kremling, Allgower 1995]

ACADO Code Generation for Benchmark CSTR



$$\dot{c}_A(t) = u_1(c_{A0} - c_A(t)) - k_1(\vartheta(t))c_A(t) - k_3(\vartheta(t))(c_A(t))^2$$
$$\dot{c}_B(t) = -u_1 c_B(t) + k_1(\vartheta(t))c_A(t) - k_2(\vartheta(t))c_B(t)$$
$$\dot{\vartheta}(t) = u_1(\vartheta_0 - \vartheta(t)) + \frac{k_w A_R}{\rho C_p V_R}(\vartheta_K(t) - \vartheta(t)) - \frac{1}{\rho C_p}[k_1(\vartheta(t))c_A(t)H_1 + k_2(\vartheta(t))c_B(t)H_2 + k_3(\vartheta(t))(c_A(t))^2 H_3]$$
$$\dot{\vartheta}_K(t) = \frac{1}{m_K C_{PK}}(u_2 + k_w A_R(\vartheta(t) - \vartheta_K(t))).$$

CSTR Benchmark by [Klatt, Engell, Kremling, Allgower 1995]

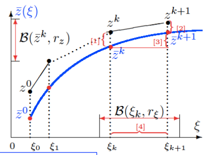
CPU Times for ACADO:

	CPU time (µs)	%
Integration & sensitivities	121	30
Condensing	98	24
QP solution (with qpOASES)[a]	180	44
Remaining operations	<5	<2
A complete real-time iteration	404	100

From [Houska, Ferreau, D., *Automatica*, 2011]

NMPC now 100 000x faster than 1997 (200x by CPU, 500x by algorithms)

(SCP Real-Time Iteration Contraction Estimate)



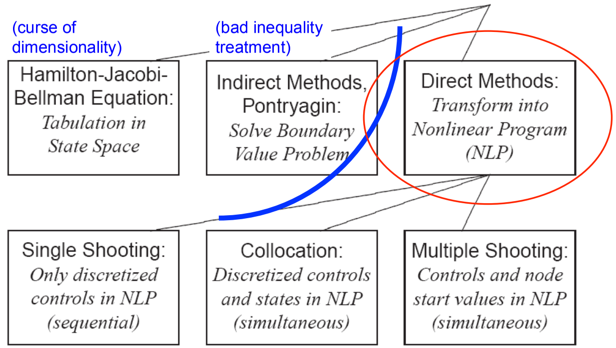
Contraction estimate for primal dual errors [1]:

$$\|z^{k+1} - \bar{z}^{k+1}\| \leq (\alpha + c_1 \|z^k - \bar{z}^k\|) \|z^k - \bar{z}^k\| + (c_2 + c_3 \|\xi_{k+1} - \xi_k\|) \|\xi_{k+1} - \xi_k\|$$

Depends only on nonlinearity of equalities, independent of active set changes!

[1] Tran Dinh, Savorgnan, Diehl: Adjoint-based predictor-corrector SCP for parametric nonlinear optimization. SIAM J. Opt. 2013 (in print)

Optimal Control Family Tree

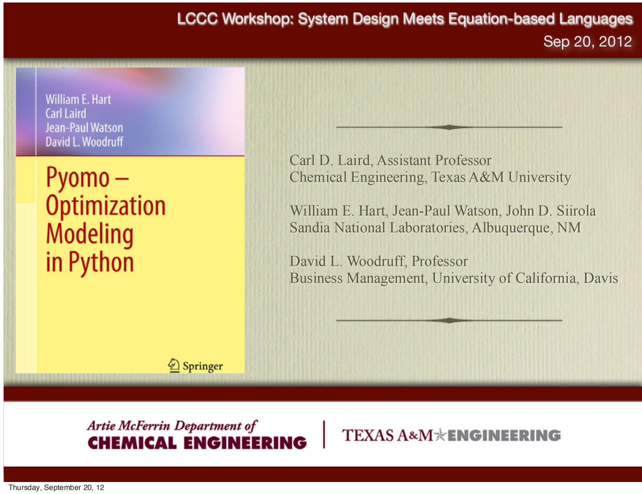


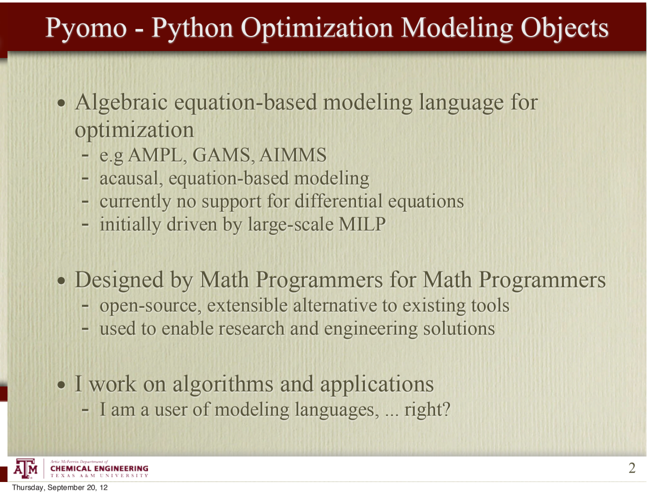
PYOMO: OPTIMIZATION MODELING IN PYTHON

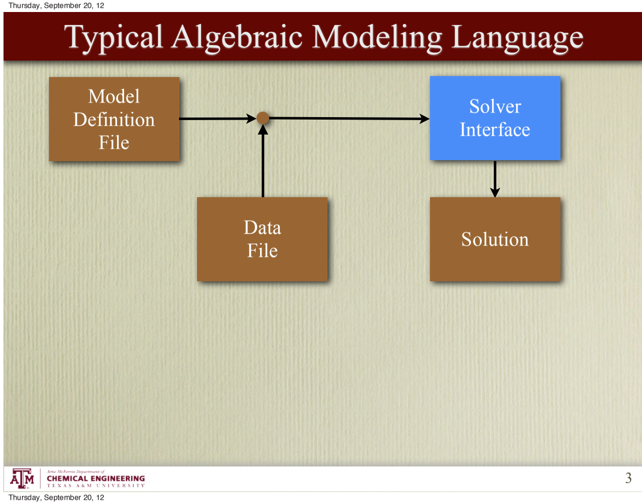
Carl Laird, Artie McFerrin Department of Chemical Engineering, Texas A&M University

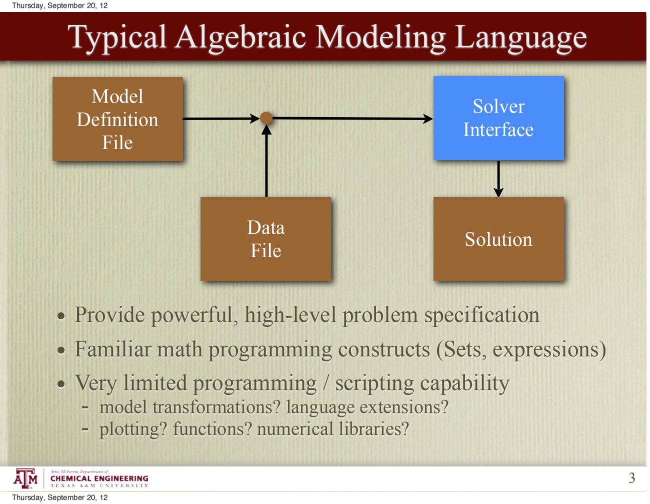
Mathematical programming has proven to be an efficient tool for design, optimization, and online operation of complex engineered systems. Algebraic modeling languages provide a convenient mechanism for the user to formulate mathematical models and optimization formulations in a language that is similar to the mathematical description of the problem, including constructs for defining sets, expressions, constraints, and objectives. In addition these tools must provide reasonable interface functionality for solvers, for example, first (and possibly second) order derivative information.

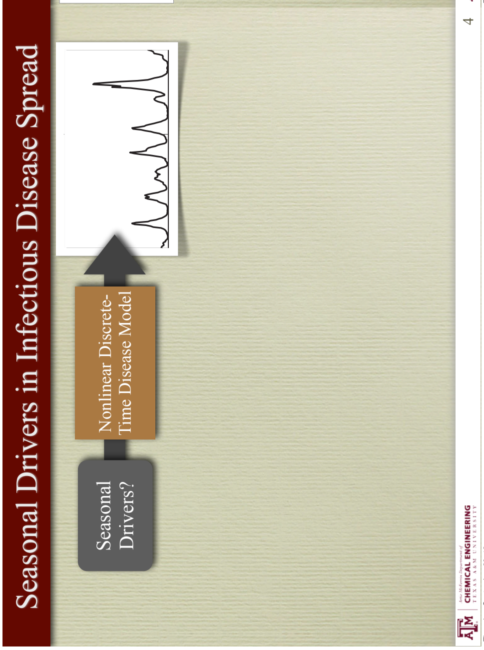
Pyomo (Python Optimization Modeling Objects) is a new open-source algebraic optimization language. Pyomo is implemented in Python, and allows the user to make use of extensive scripting capabilities within a familiar, exhaustive, and well-documented programming environment. Pyomo provides general functionality to formulate and solve optimization problems with little or no programming knowledge, but also provides the flexibility to implement high-level language constructs. In this presentation, I will discuss the design and implementation of the Pyomo framework, and give examples of several language extensions including PySP, an extension that supports parallel programming for solution of difficult stochastic programming problems.

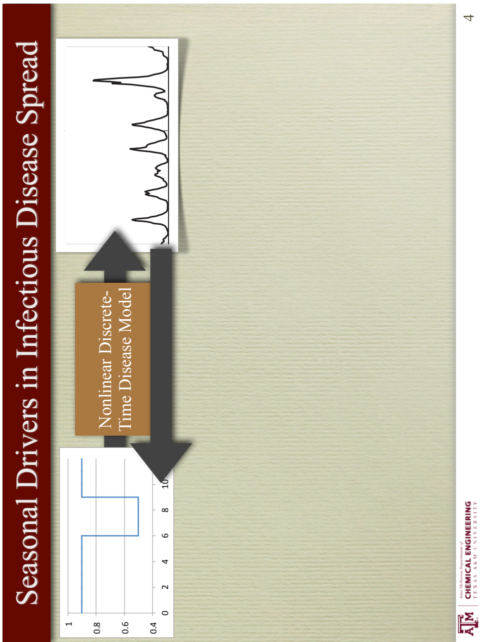

LCCC Workshop: System Design Meets Equation-based Languages
Sep 20, 2012
William E. Hart
Carl Laird
Jean-Paul Watson
David L. Woodruff
Pyomo – Optimization Modeling in Python
Springer
Carl D. Laird, Assistant Professor
Chemical Engineering, Texas A&M University
William E. Hart, Jean-Paul Watson, John D. Siirola
Sandia National Laboratories, Albuquerque, NM
David L. Woodruff, Professor
Business Management, University of California, Davis
Artie McFerrin Department of
CHEMICAL ENGINEERING
TEXAS A&M ENGINEERING
Thursday, September 20, 12

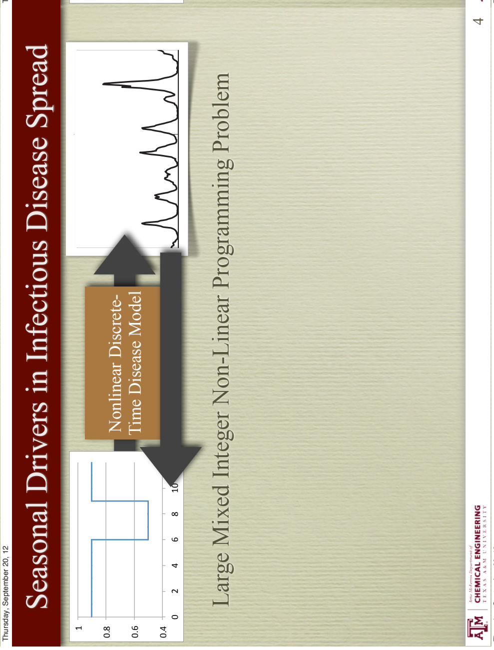

Pyomo - Python Optimization Modeling Objects
• Algebraic equation-based modeling language for optimization
- e.g AMPL, GAMS, AIMMS
- acausal, equation-based modeling
- currently no support for differential equations
- initially driven by large-scale MILP
• Designed by Math Programmers for Math Programmers
- open-source, extensible alternative to existing tools
- used to enable research and engineering solutions
• I work on algorithms and applications
- I am a user of modeling languages, ... right?
CHEMICAL ENGINEERING
2
Thursday, September 20, 12

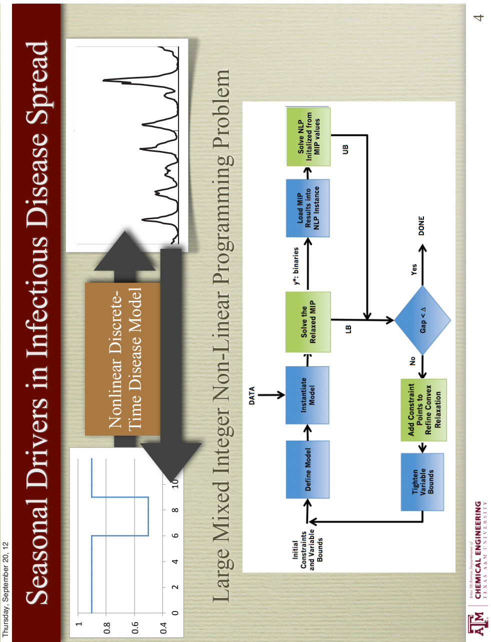

Typical Algebraic Modeling Language
Model Definition File
Data File
Solver Interface
Solution
CHEMICAL ENGINEERING
3
Thursday, September 20, 12

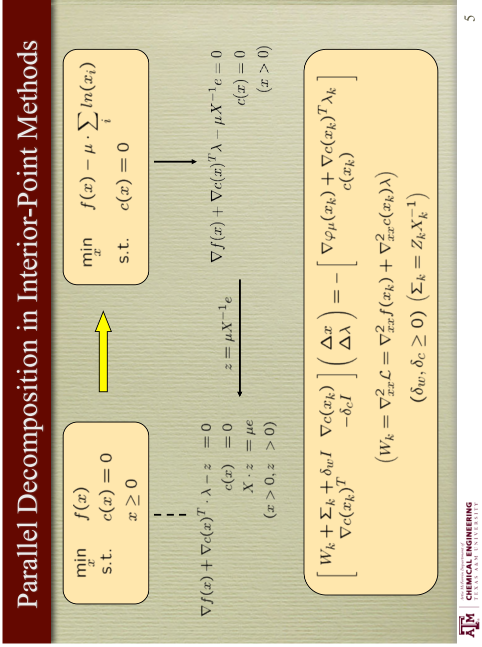

Typical Algebraic Modeling Language
Model Definition File
Data File
Solver Interface
Solution
• Provide powerful, high-level problem specification
• Familiar math programming constructs (Sets, expressions)
• Very limited programming / scripting capability
- model transformations? language extensions?
- plotting? functions? numerical libraries?
CHEMICAL ENGINEERING
3
Thursday, September 20, 12

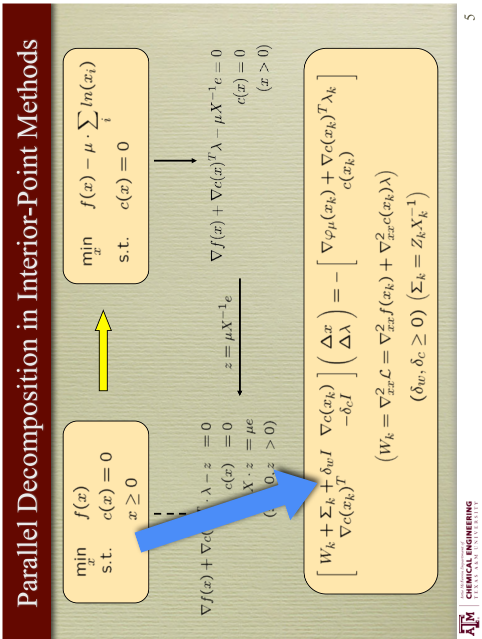

Seasonal Drivers in Infectious Disease Spread
Seasonal Drivers?
Nonlinear Discrete-Time Disease Model
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
Thursday, September 20, 12
4

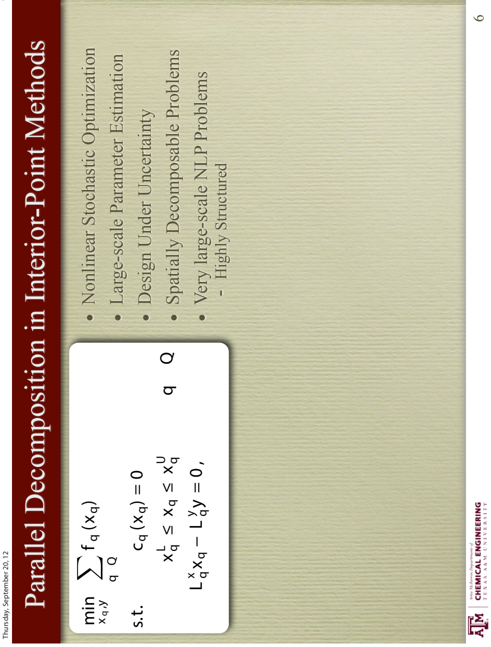

Seasonal Drivers in Infectious Disease Spread
Nonlinear Discrete-Time Disease Model
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
Thursday, September 20, 12
4

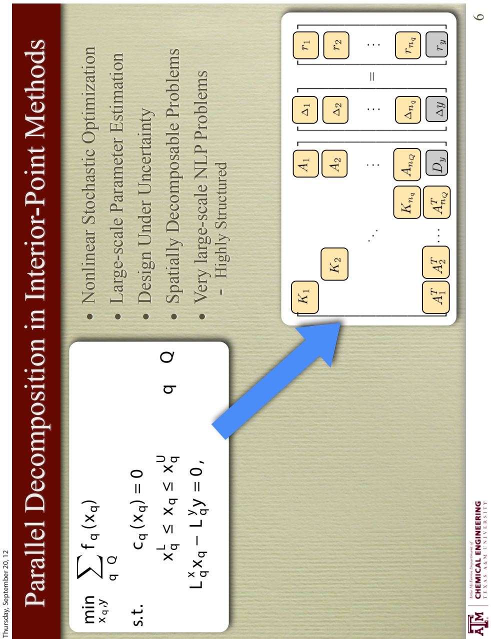

Seasonal Drivers in Infectious Disease Spread
Nonlinear Discrete-Time Disease Model
Large Mixed Integer Non-Linear Programming Problem
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
Thursday, September 20, 12
4

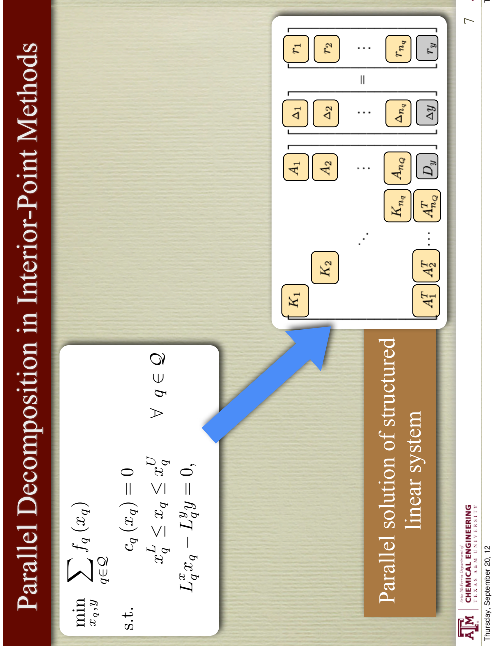

Seasonal Drivers in Infectious Disease Spread
Nonlinear Discrete-Time Disease Model
Large Mixed Integer Non-Linear Programming Problem
Initial Constraints and Variable Bounds
Define Model
DATA
Instantiate Model
Solve the Relaxed MIP
y*: binaries
Load MIP Results into NLP Instance
Solve NLP Initialized from MIP values
UB
LB
Gap < Δ
Yes
DONE
No
Add Constraint Points to Refine Convex Relaxation
Tighten Variable Bounds
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
Thursday, September 20, 12
4

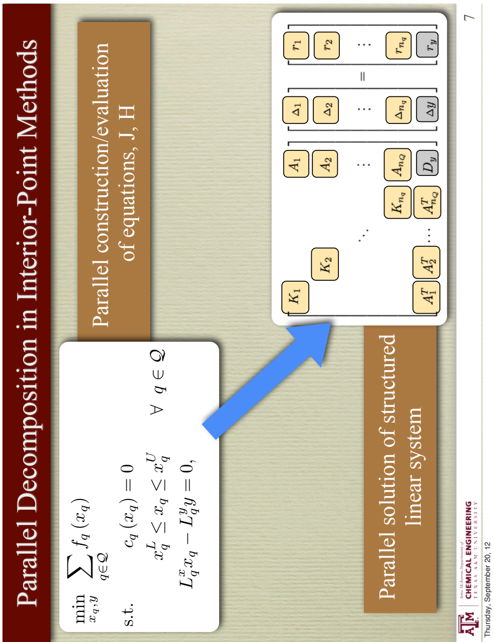

Parallel Decomposition in Interior-Point Methods
min_x f(x) s.t. c(x) = 0, x ≥ 0
min_x f(x) − μ · Σ_i ln(x_i) s.t. c(x) = 0
∇f(x) + ∇c(x)^T · λ − z = 0
c(x) = 0
X · z = μe
(x > 0, z > 0)
z = μX^{-1}e
∇f(x) + ∇c(x)^T λ − μX^{-1}e = 0
c(x) = 0
(x > 0)
[W_k + Σ_k + δ_w I, ∇c(x_k); ∇c(x_k)^T, −δ_c I] (Δx; Δλ) = − [∇φ_μ(x_k) + ∇c(x_k)^T λ_k; c(x_k)]
(W_k = ∇²_{xx}L = ∇²_{xx}f(x_k) + ∇²_{xx}c(x_k)λ)
(δ_w, δ_c ≥ 0) (Σ_k = Z_k X_k^{-1})
5
Thursday, September 20, 12
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY

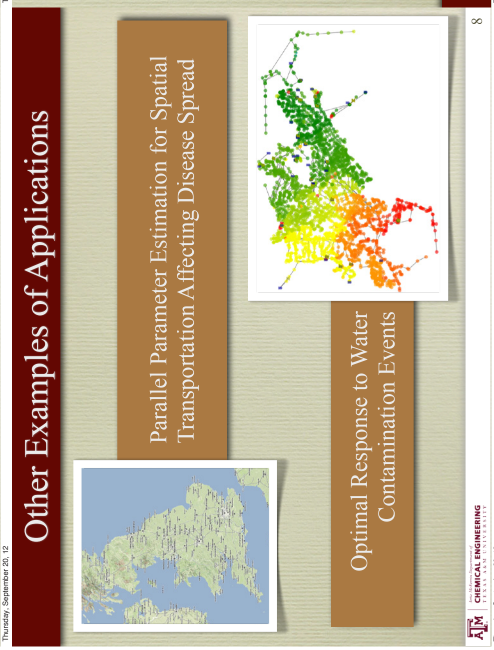

Parallel Decomposition in Interior-Point Methods
min_x f(x) s.t. c(x) = 0, x ≥ 0
min_x f(x) − μ · Σ_i ln(x_i) s.t. c(x) = 0
∇f(x) + ∇c(x)^T λ − μX^{-1}e = 0
c(x) = 0
(x > 0)
z = μX^{-1}e
[W_k + Σ_k + δ_w I, ∇c(x_k); ∇c(x_k)^T, −δ_c I] (Δx; Δλ) = − [∇φ_μ(x_k) + ∇c(x_k)^T λ_k; c(x_k)]
(W_k = ∇²_{xx}L = ∇²_{xx}f(x_k) + ∇²_{xx}c(x_k)λ)
(δ_w, δ_c ≥ 0) (Σ_k = Z_k X_k^{-1})
5
Thursday, September 20, 12
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY

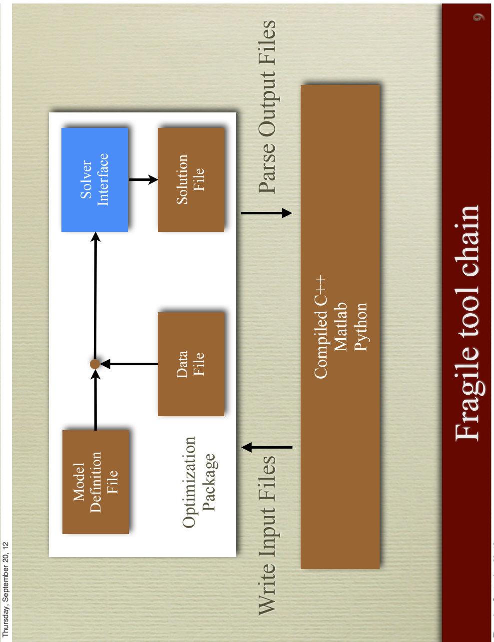

Parallel Decomposition in Interior-Point Methods
min_{x_q,y} Σ_{q Q} f_q(x_q)
s.t. c_q(x_q) = 0
x_q^L ≤ x_q ≤ x_q^U
L_q^x x_q − L_q^y y = 0, q Q
• Nonlinear Stochastic Optimization
• Large-scale Parameter Estimation
• Design Under Uncertainty
• Spatially Decomposable Problems
• Very large-scale NLP Problems
- Highly Structured
6
Thursday, September 20, 12
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY


Parallel Decomposition in Interior-Point Methods
min_{x_q,y} Σ_{q Q} f_q(x_q)
s.t. c_q(x_q) = 0
x_q^L ≤ x_q ≤ x_q^U
L_q^x x_q − L_q^y y = 0, q Q
• Nonlinear Stochastic Optimization
• Large-scale Parameter Estimation
• Design Under Uncertainty
• Spatially Decomposable Problems
• Very large-scale NLP Problems
- Highly Structured
K_1, A_1; K_2, A_2; K_{n_q}, A_{n_q}; A_1^T, A_2^T, A_{n_Q}^T, D_y
Δ_1, Δ_2, Δ_{n_q}, Δy
r_1, r_2, r_{n_q}, r_y
6
Thursday, September 20, 12
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY


Parallel Decomposition in Interior-Point Methods
$\min_{x_q,y} \sum_{q \in Q} f_q(x_q)$
s.t. $c_q(x_q) = 0$
$x_q^L \le x_q \le x_q^U$ $\forall q \in Q$
$L_q^x x_q - L_q^y y = 0,$
Parallel solution of structured linear system
Thursday, September 20, 12
7


Parallel Decomposition in Interior-Point Methods
Parallel construction/evaluation of equations, J, H
$\min_{x_q,y} \sum_{q \in Q} f_q(x_q)$
s.t. $c_q(x_q) = 0$
$x_q^L \le x_q \le x_q^U$ $\forall q \in Q$
$L_q^x x_q - L_q^y y = 0,$
Parallel solution of structured linear system
Thursday, September 20, 12
7


Other Examples of Applications
Parallel Parameter Estimation for Spatial Transportation Affecting Disease Spread
Optimal Response to Water Contamination Events
Thursday, September 20, 12
8


Model Definition File
Data File
Solver Interface
Solution File
Optimization Package
Write Input Files
Parse Output Files
Compiled C++
Matlab
Python
Fragile tool chain
Thursday, September 20, 12
9

Two Choices

1. Design new language
   - modeling, scripting syntax
   - compiler tools

2. Use programming language
   - develop components in another language
   - import types/functionality

Two Choices

1. Design new language
   - modeling, scripting syntax
   - compiler tools

2. Use programming language
   - develop components in another language
   - import types/functionality

- Selected to develop in Python (Choice 2)
  - tired of writing parsers
  - not language experts
  - existing tools are not actively updated
  - not responsible for full language functionality and packages
  - want full-featured language and user-extensibility (for "free")

Requirements

- Powerful
  - full support for standard math programming constructs (LP, MILP, NLP, MINLP, ...)
  - full-featured programming environment (model interrogation, scripting, functions, classes, standard & numerical libraries)
  - extensive solver integration - "out-of-the-box"
- Open
  - licensed under BSD (i.e. really open-source)
  - reduce barriers to adoption, ease of collaboration
  - transparency
- Flexible
  - extensible by users, contributors, not only by us
  - portable (Windows, Linux, OS X)
- Easy
  - language constructs familiar to math programmers - Abstract Models
  - scripting / programming capability well-defined
  - substantial documentation

Why Python?

- License
  - open-source
- Language Features
  - familiar, lean syntax, rich set of existing data types, object-oriented, exceptions, dynamic loading, ...
- Support and stability
  - highly stable, well-supported
- Documentation
  - extensive online documentation, several books
- Libraries
  - significant external libraries, numerical & scientific packages
- Portability
  - widely available on many platforms

Simple Modeling Example: Knapsack

$\mathcal{S}$: set of items (set)
v_i : value of item i (param)
w_i : weight of item i (param)
W_{max} : maximum weight (param)
x_i : binary indicator (var)

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$
$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_{max}$$
$$x_i \in \{0,1\} \quad \forall i \in \mathcal{S}$$

CHEMICAL ENGINEERING TEXAS A&M UNIVERSITY 13

Thursday, September 20, 12

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$
$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$
$$x_i \in \{0,1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
      <= model.W_max
model.weight = Constraint( )
```

14

Thursday, September 20, 12

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$
$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$
$$x_i \in \{0,1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
      <= model.W_max
model.weight = Constraint( )
```

15

Thursday, September 20, 12

Knapsack Problem: Abstract Model

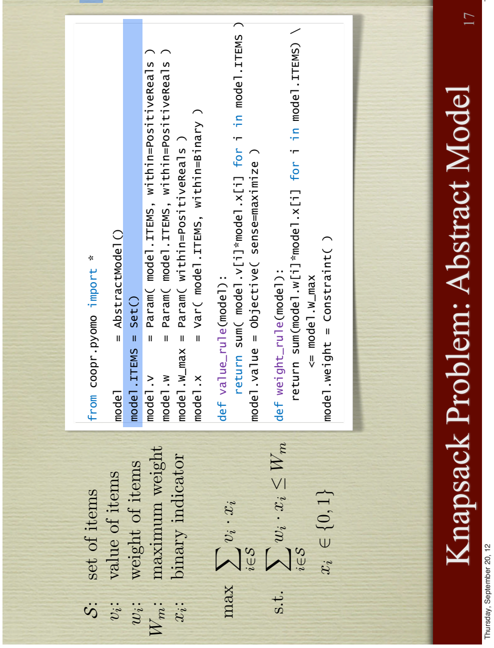
$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$
$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$
$$x_i \in \{0,1\}$$

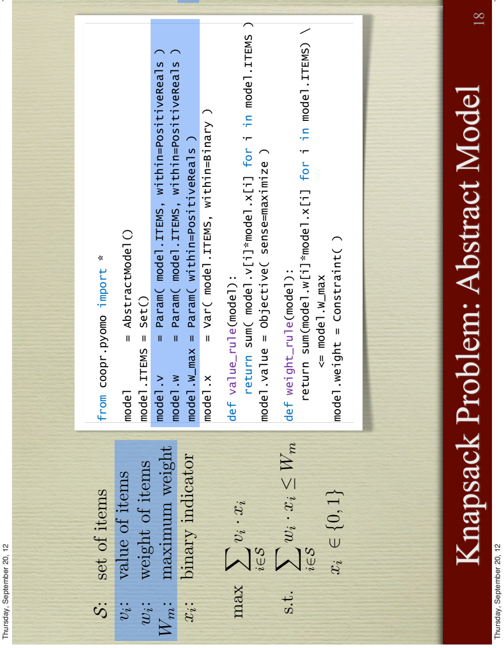
```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
      <= model.W_max
model.weight = Constraint( )
```

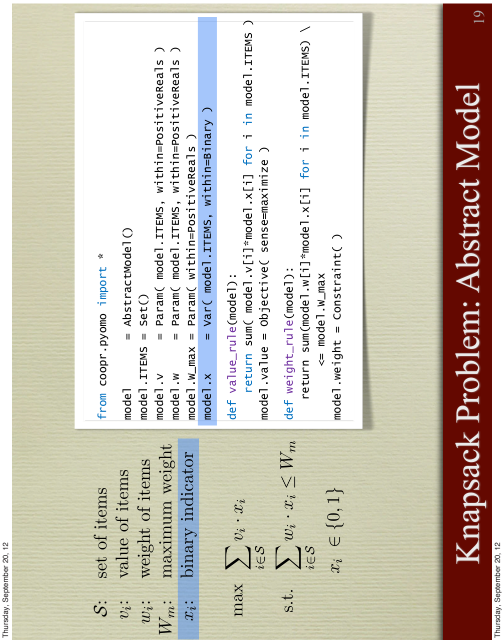
16

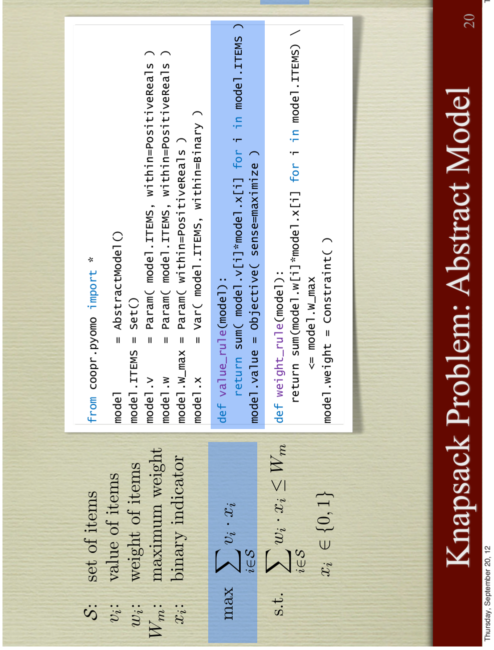
Thursday, September 20, 12

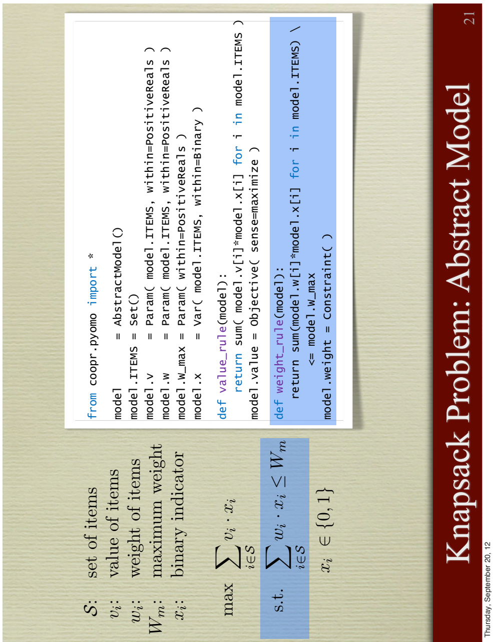

S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max ∑_{i∈S} v_i · x_i
s.t. ∑_{i∈S} w_i · x_i ≤ W_m
x_i ∈ {0,1}
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
Knapsack Problem: Abstract Model
Thursday, September 20, 12
17


S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max ∑_{i∈S} v_i · x_i
s.t. ∑_{i∈S} w_i · x_i ≤ W_m
x_i ∈ {0,1}
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
Knapsack Problem: Abstract Model
Thursday, September 20, 12
17

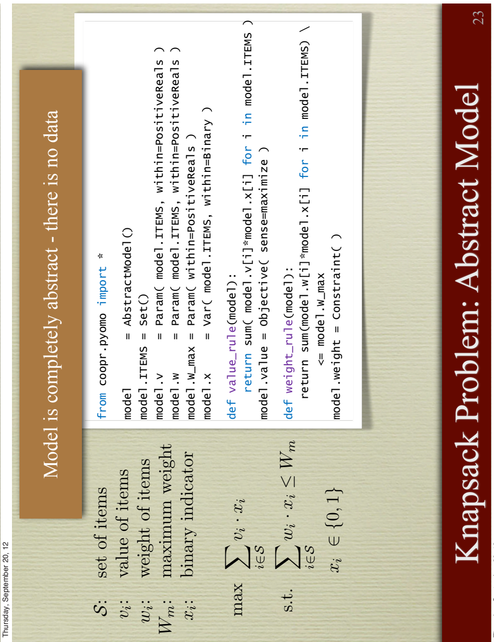

S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max ∑_{i∈S} v_i · x_i
s.t. ∑_{i∈S} w_i · x_i ≤ W_m
x_i ∈ {0,1}
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
Knapsack Problem: Abstract Model
Thursday, September 20, 12
18

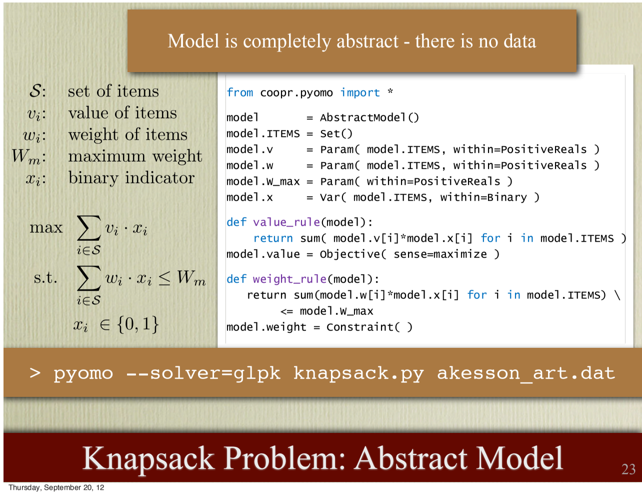

S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max ∑_{i∈S} v_i · x_i
s.t. ∑_{i∈S} w_i · x_i ≤ W_m
x_i ∈ {0,1}
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
Knapsack Problem: Abstract Model
Thursday, September 20, 12
19

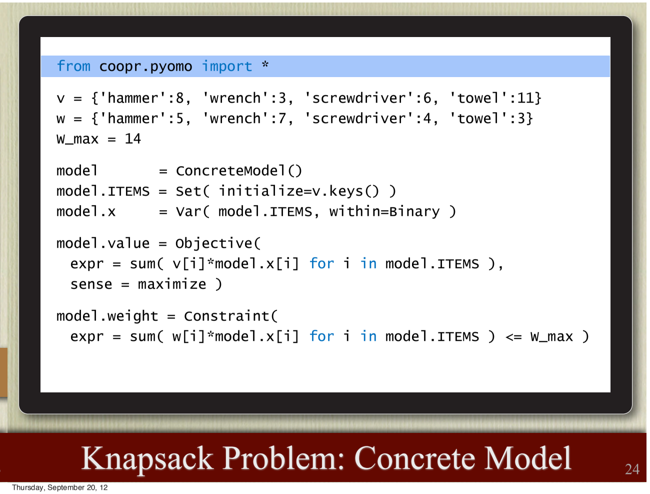

Knapsack Problem: Abstract Model
S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max $\sum_{i \in S} v_i \cdot x_i$
s.t. $\sum_{i \in S} w_i \cdot x_i \leq W_m$
$x_i \in \{0,1\}$
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
20
Thursday, September 20, 12

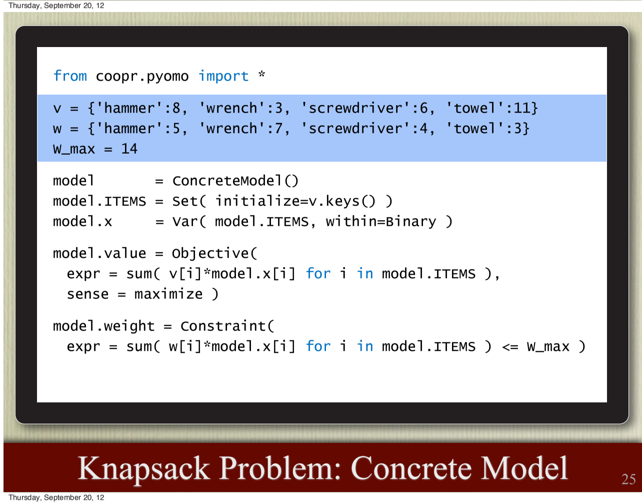

Knapsack Problem: Abstract Model
S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max $\sum_{i \in S} v_i \cdot x_i$
s.t. $\sum_{i \in S} w_i \cdot x_i \leq W_m$
$x_i \in \{0,1\}$
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
21
Thursday, September 20, 12

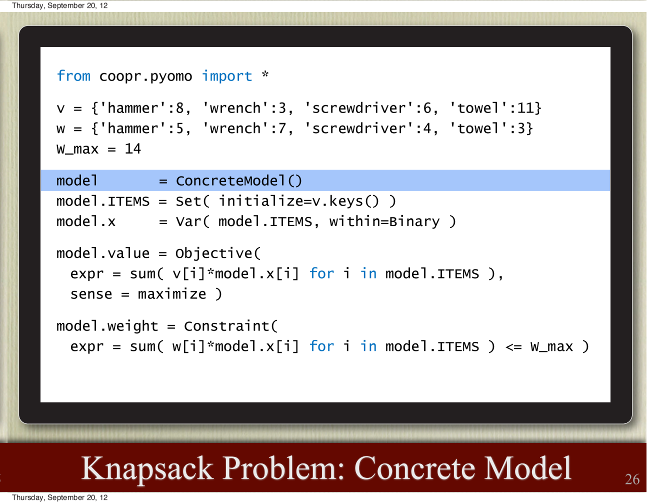

Knapsack Problem: Abstract Model
S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max $\sum_{i \in S} v_i \cdot x_i$
s.t. $\sum_{i \in S} w_i \cdot x_i \leq W_m$
$x_i \in \{0,1\}$
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
22
Thursday, September 20, 12

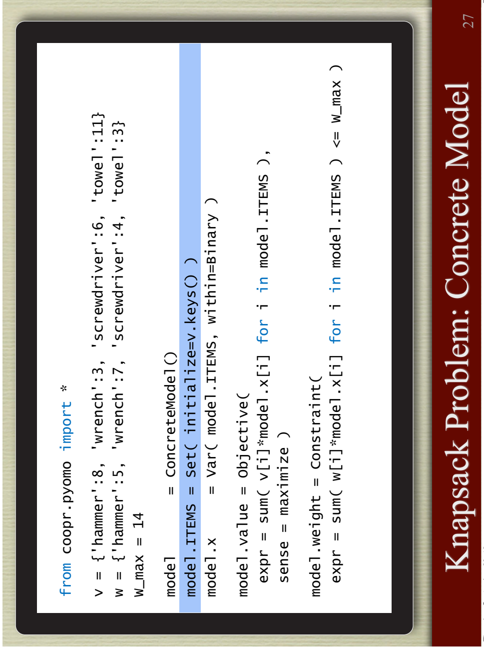

Knapsack Problem: Abstract Model
Model is completely abstract - there is no data
S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max $\sum_{i \in S} v_i \cdot x_i$
s.t. $\sum_{i \in S} w_i \cdot x_i \leq W_m$
$x_i \in \{0,1\}$
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
23
Thursday, September 20, 12

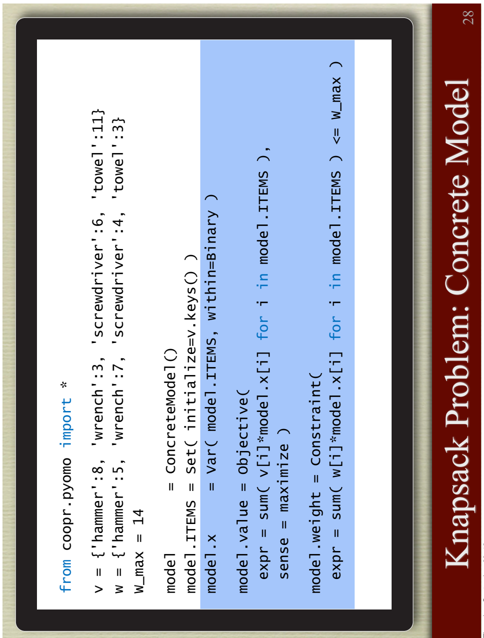

Model is completely abstract - there is no data
S: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator
max ∑_{i∈S} v_i · x_i
s.t. ∑_{i∈S} w_i · x_i ≤ W_m
x_i ∈ {0,1}
from coopr.pyomo import *
model = AbstractModel()
model.ITEMS = Set()
model.v = Param(model.ITEMS, within=PositiveReals)
model.w = Param(model.ITEMS, within=PositiveReals)
model.W_max = Param(within=PositiveReals)
model.x = Var(model.ITEMS, within=Binary)
def value_rule(model):
return sum(model.v[i]*model.x[i] for i in model.ITEMS)
model.value = Objective(sense=maximize)
def weight_rule(model):
return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
<= model.W_max
model.weight = Constraint( )
> pyomo --solver=glpk knapsack.py akesson_art.dat
Knapsack Problem: Abstract Model
23
Thursday, September 20, 12

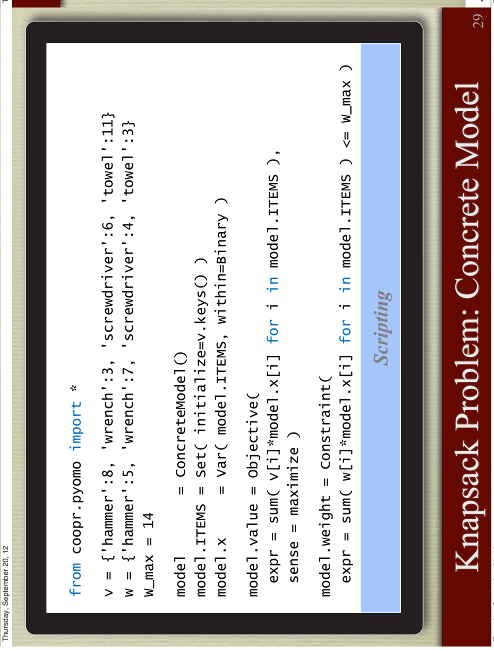

from coopr.pyomo import *
v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14
model = ConcreteModel()
model.ITEMS = Set(initialize=v.keys())
model.x = Var(model.ITEMS, within=Binary)
model.value = Objective(
expr = sum(v[i]*model.x[i] for i in model.ITEMS),
sense = maximize)
model.weight = Constraint(
expr = sum(w[i]*model.x[i] for i in model.ITEMS) <= W_max)
Knapsack Problem: Concrete Model
24
Thursday, September 20, 12

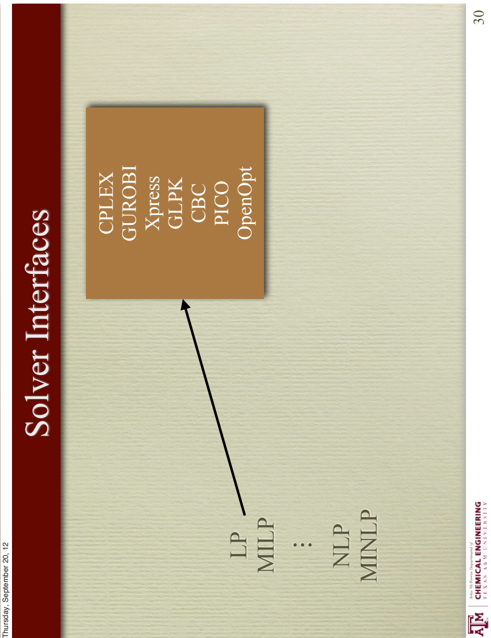

from coopr.pyomo import *
v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14
model = ConcreteModel()
model.ITEMS = Set(initialize=v.keys())
model.x = Var(model.ITEMS, within=Binary)
model.value = Objective(
expr = sum(v[i]*model.x[i] for i in model.ITEMS),
sense = maximize)
model.weight = Constraint(
expr = sum(w[i]*model.x[i] for i in model.ITEMS) <= W_max)
Knapsack Problem: Concrete Model
25
Thursday, September 20, 12

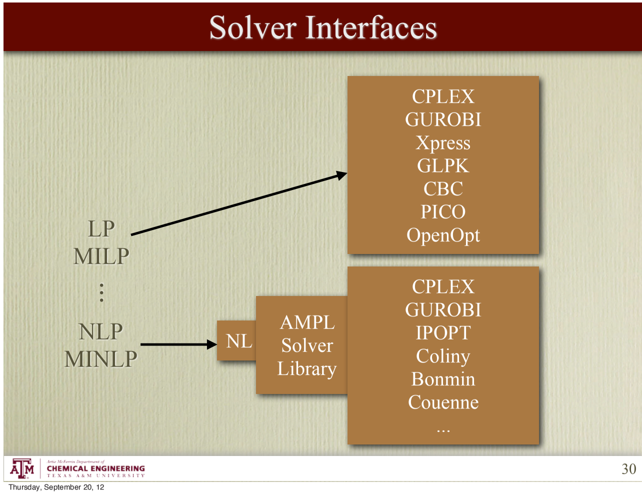

from coopr.pyomo import *
v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14
model = ConcreteModel()
model.ITEMS = Set(initialize=v.keys())
model.x = Var(model.ITEMS, within=Binary)
model.value = Objective(
expr = sum(v[i]*model.x[i] for i in model.ITEMS),
sense = maximize)
model.weight = Constraint(
expr = sum(w[i]*model.x[i] for i in model.ITEMS) <= W_max)
Knapsack Problem: Concrete Model
26
Thursday, September 20, 12

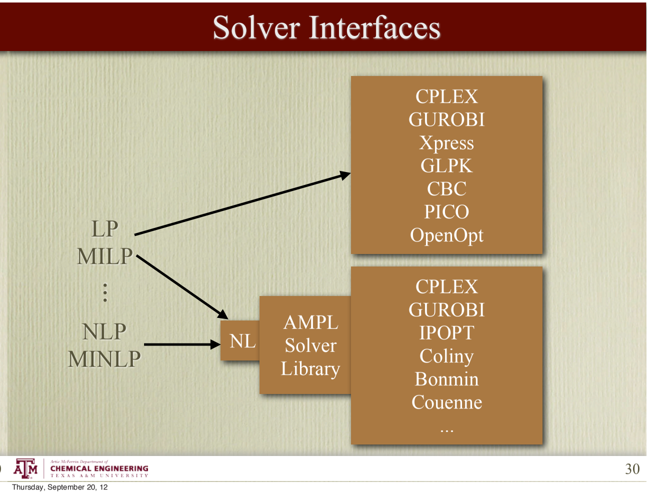

from coopr.pyomo import *
v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14
model = ConcreteModel()
model.ITEMS = Set(initialize=v.keys())
model.x = Var(model.ITEMS, within=Binary)
model.value = Objective(
expr = sum(v[i]*model.x[i] for i in model.ITEMS),
sense = maximize)
model.weight = Constraint(
expr = sum(w[i]*model.x[i] for i in model.ITEMS) <= W_max)
Knapsack Problem: Concrete Model
27
Thursday, September 20, 12

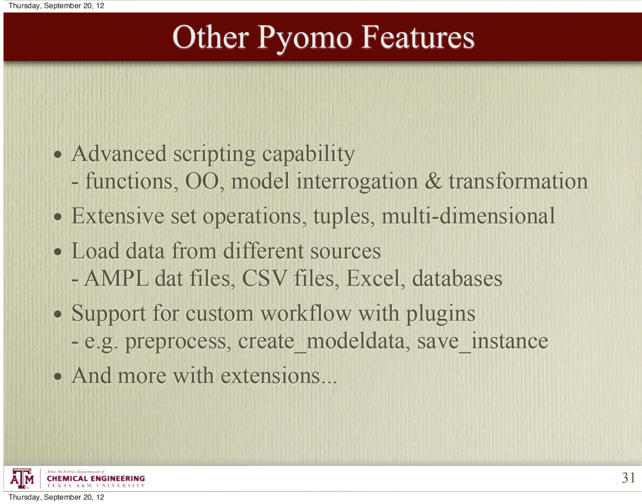

from coopr.pyomo import *
v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14
model = ConcreteModel()
model.ITEMS = Set(initialize=v.keys())
model.x = Var(model.ITEMS, within=Binary)
model.value = Objective(
expr = sum(v[i]*model.x[i] for i in model.ITEMS),
sense = maximize)
model.weight = Constraint(
expr = sum(w[i]*model.x[i] for i in model.ITEMS) <= W_max)
Knapsack Problem: Concrete Model
28
Thursday, September 20, 12

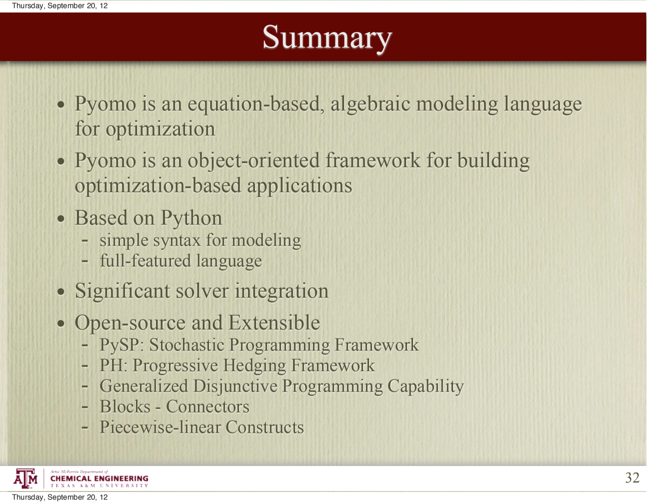

from coopr.pyomo import *
v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14
model = ConcreteModel()
model.ITEMS = Set(initialize=v.keys())
model.x = Var(model.ITEMS, within=Binary)
model.value = Objective(
expr = sum(v[i]*model.x[i] for i in model.ITEMS),
sense = maximize)
model.weight = Constraint(
expr = sum(w[i]*model.x[i] for i in model.ITEMS) <= W_max)
Scripting
Knapsack Problem: Concrete Model
29
Thursday, September 20, 12

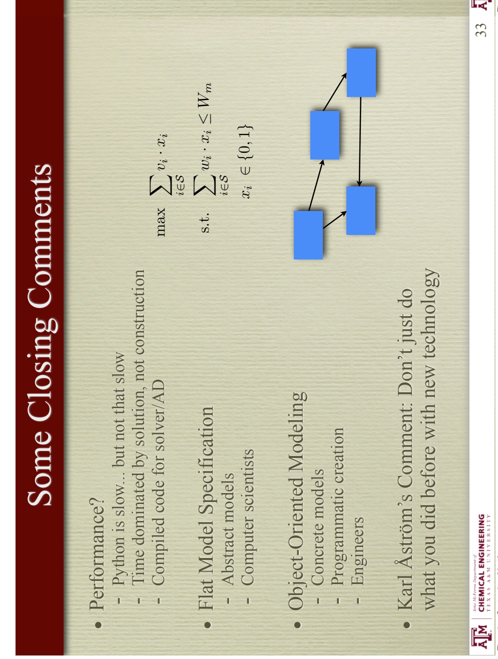

Solver Interfaces
LP
MILP
:
NLP
MINLP
CPLEX
GUROBI
Xpress
GLPK
CBC
PICO
OpenOpt
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
30
Thursday, September 20, 12


Solver Interfaces
LP
MILP
⋮
NLP
MINLP
NL
AMPL Solver Library
CPLEX
GUROBI
Xpress
GLPK
CBC
PICO
OpenOpt
CPLEX
GUROBI
IPOPT
Coliny
Bonmin
Couenne
...
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
30
Thursday, September 20, 12

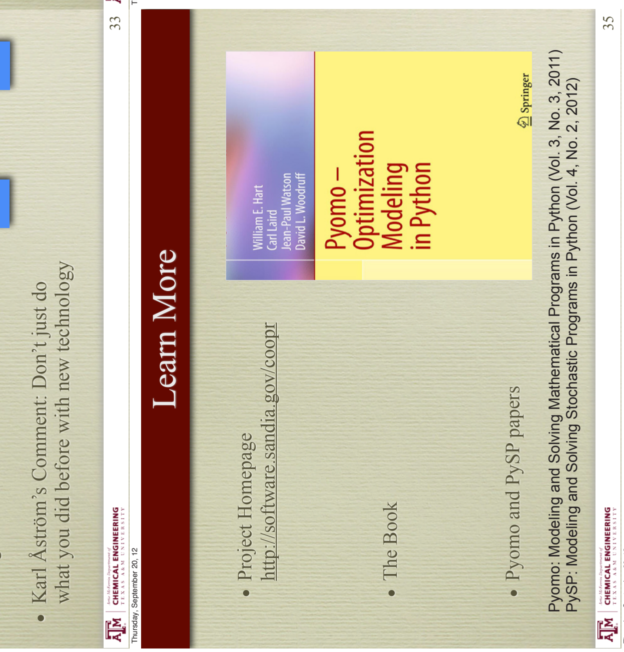

Solver Interfaces
LP
MILP
⋮
NLP
MINLP
NL
AMPL Solver Library
CPLEX
GUROBI
Xpress
GLPK
CBC
PICO
OpenOpt
CPLEX
GUROBI
IPOPT
Coliny
Bonmin
Couenne
...
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
30
Thursday, September 20, 12


Other Pyomo Features
• Advanced scripting capability
- functions, OO, model interrogation & transformation
• Extensive set operations, tuples, multi-dimensional
• Load data from different sources
- AMPL dat files, CSV files, Excel, databases
• Support for custom workflow with plugins
- e.g. preprocess, create_modeldata, save_instance
• And more with extensions...
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
31
Thursday, September 20, 12


Summary
• Pyomo is an equation-based, algebraic modeling language for optimization
• Pyomo is an object-oriented framework for building optimization-based applications
• Based on Python
- simple syntax for modeling
- full-featured language
• Significant solver integration
• Open-source and Extensible
- PySP: Stochastic Programming Framework
- PH: Progressive Hedging Framework
- Generalized Disjunctive Programming Capability
- Blocks - Connectors
- Piecewise-linear Constructs
CHEMICAL ENGINEERING
TEXAS A&M UNIVERSITY
32
Thursday, September 20, 12


Some Closing Comments
• Performance?
- Python is slow... but not that slow
- Time dominated by solution, not construction
- Compiled code for solver/AD
• Flat Model Specification
- Abstract models
- Computer scientists
• Object-Oriented Modeling
- Concrete models
- Programmatic creation
- Engineers
• Karl Åström's Comment: Don't just do what you did before with new technology
max $\sum_{i \in S} v_i \cdot x_i$
s.t. $\sum_{i \in S} w_i \cdot x_i \leq W_m$
$x_i \in \{0,1\}$
CHEMICAL ENGINEERING
Thursday, September 20, 12
33


Aknowledgments (Development Community)
• Sandia National Laboratories
- Bill Hart
- Jean-Paul Watson
- John Siirola
- David Hart
- Tom Brounstein
• University of California, Davis
- Prof. David L. Woodruff
- Prof. Roger Wets
• Texas A&M University
- Prof. Carl D. Laird
- Daniel Word
- James Young
- Gabe Hackebeil
• Texas Tech University
- Zev Friedman
• Rose Hulman Institute
- Tim Ekl
- William & Mary
- Patrick Steele
• North Carolina State
- Kevin Hunter
Plus our many users, including:
- University of California, Davis
- Texas A&M University
- University of Texas
- Rose-Hulman Institute of Technology
- University of Southern California
- George Mason University
- Iowa State University
- N.C. State University
- University of Washington
- Naval Postgraduate School
- Universidad de Santiago de Chile
- University of Pisa
- Lawrence Livermore National Lab
- Los Alamos National Lab
CHEMICAL ENGINEERING
Thursday, September 20, 12
34


Learn More
• Project Homepage
http://software.sandia.gov/coopr
• The Book
William E. Hart
Carl Laird
Jean-Paul Watson
David L. Woodruff
Pyomo – Optimization Modeling in Python
Springer
• Pyomo and PySP papers
Pyomo: Modeling and Solving Mathematical Programs in Python (Vol. 3, No. 3, 2011)
PySP: Modeling and Solving Stochastic Programs in Python (Vol. 4, No. 2, 2012)
CHEMICAL ENGINEERING
Thursday, September 20, 12
35

EFFICIENT SYMBOLICAL AND NUMERICAL ALGORITHMS FOR NONLINEAR MODEL PREDICTIVE CONTROL WITH OPENMODELICA

Bernhard Bachmann, Fachhochschule Bielefeld University of Applied Sciences

During the last decade nonlinear model predictive control (NMPC) has become increasingly important for today's control engineers. In order to apply NMPC a nonlinear optimal control problem (NOCP) must be solved, which needs a very high computational effort. Nowadays, corresponding modeling of the system dynamics and formulation of the optimization problem can be done in Modelica and Optimica, respectively.

State-of-the-art NOCP solution algorithms are based on multiple shooting and/or collocation algorithms. Only parallelizing these time-consuming algorithms can give reasonable performance appropriate for online-applications. In addition, efficient symbolical and numerical treatment of the underlying model formulation (e.g. matching, sorting, and tearing) are necessary, when solving NOCP involving complex system. Furthermore, for performance and stability reasons NOCP corresponding symbolically derived Jacobian and Hessian matrices and their efficient computation (e.g. identify and utilize the sparsity pattern of Jacobian matrices) are needed. This talk will discuss these mathematical aspects of NMPC as well as the current and future implementation of efficient, partly parallelized symbolical and numerical algorithms available in and with OpenModelica.

Efficient Symbolical and Numerical Algorithms for nonlinear model predictive control with OpenModelica

Bernhard Bachmann
Department of Engineering and Mathematics
University of Applied Sciences Bielefeld

FH Bielefeld
University of
Applied Sciences

Outline

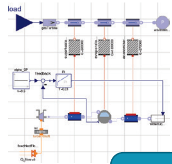
1. Excerpt of OpenModelica's symbolic machinery

2. Symbolically derived Jacobians
   i. Directional derivatives
   ii. Sparsity pattern
   iii. Coloring of the Jacobian

3. Nonlinear Optimal Control Problem
   i. General Discretization Scheme
   ii. Multiple Shooting/Collocation
   iii. Total Collocation
   iv. Applications

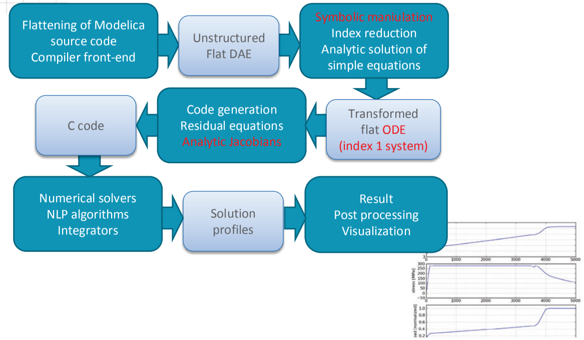
4. Lessons learned & Outlook

FH Bielefeld
University of
Applied Sciences



A Modelica-based Tool Chain

(Johan Åkesson)



Symbolic Machinery of OpenModelica

General representation of DAEs (continuous signals):

$$\underline{0} = \underline{f}\left(t,\ \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$

t	time
$\underline{\dot{x}}(t)$	vector of differentiated state variables
$\underline{x}(t)$	vector of state variables
$\underline{y}(t)$	vector of algebraic variables
$\underline{u}(t)$	vector of input variables
$\underline{p}$	vector of parameters and/or constants

Basic Transformation Steps

Transformation to explicit state-space representation:

$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$

$$\downarrow$$

$$\underline{0} = \underline{f}\left(t, \underline{z}(t), \underline{x}(t), \underline{u}(t), \underline{p}\right), \quad \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\nearrow$$

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

$$\downarrow$$

$$\underline{\dot{x}}(t) = \underline{h}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

$$\underline{y}(t) = \underline{k}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

Implicit function theorem:

Necessary condition for the existence of the transformation is that the following matrix is regular at the point of interest:

$$\det\left(\frac{\partial}{\partial \underline{z}} \underline{f}\left(t, \underline{z}(t), \underline{x}(t), \underline{u}(t), \underline{p}\right)\right) \neq 0$$

Symbolic Transformation Algorithmic Steps

- DAEs and bipartite graph representation
  - Structural representation of the equation system
- The matching problem
  - Assign to each variable exact one equation
  - Same number of equations and unknowns
- Construct a directed graph
  - Find sinks, sources and strong components
  - Sorting the equation system
- Adjacence Matrix and structural regularity
  - Block-lower triangular form (BLT-Transformation)

$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$

$$\downarrow$$

$$\underline{0} = \underline{f}\left(t, \underline{z}(t), \underline{x}(t), \underline{u}(t), \underline{p}\right), \quad \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\downarrow$$

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

$$\downarrow$$

$$\underline{\dot{x}}(t) = \underline{h}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

$$\underline{y}(t) = \underline{k}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

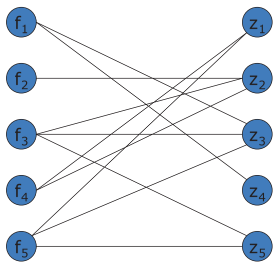
DAEs and Bipartite Graph Representation

Example of a regular DAE:

$$\underline{0} = \underline{f}\left(t, \underline{z}(t), \underline{x}(t), \underline{u}(t), \underline{p}\right), \quad \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$f_1(z_3,z_4) = 0$
$f_2(z_2) = 0$
$f_3(z_2,z_3,z_5) = 0$
$f_4(z_1,z_2) = 0$
$f_5(z_1,z_3,z_5) = 0$

Bipartite graph



Adjacence matrix

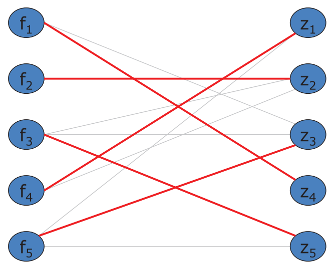
	z_1	z_2	z_3	z_4	z_5
f_1	0	0	1	1	0
f_2	0	1	0	0	0
f_3	0	1	1	0	1
f_4	1	1	0	0	0
f_5	1	0	1	0	1

Solve the Matching Problem

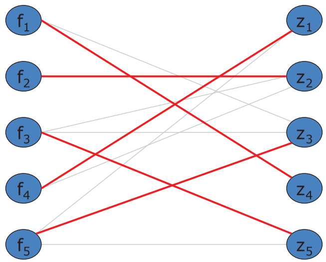
Example of a regular DAE:

$f_1(z_3,z_4) = 0$
$f_2(z_2) = 0$
$f_3(z_2,z_3,z_5) = 0$
$f_4(z_1,z_2) = 0$
$f_5(z_1,z_3,z_5) = 0$

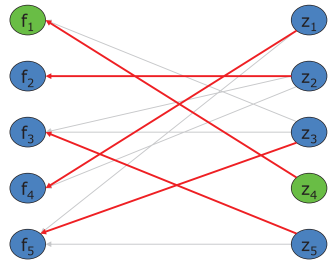
Bipartite graph



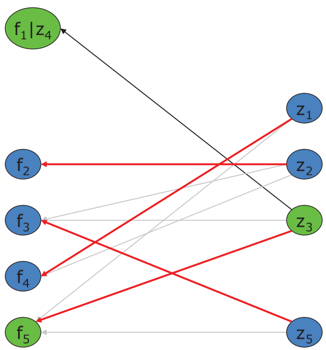
Construct a Directed Graph



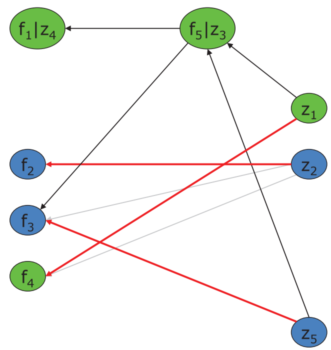
Construct a Directed Graph



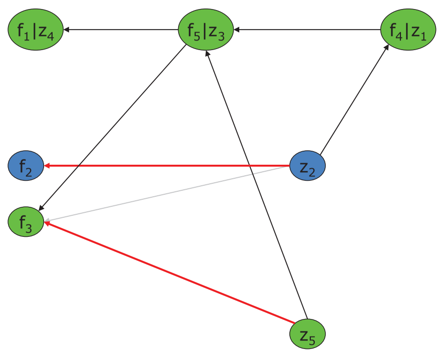
Construct a Directed Graph



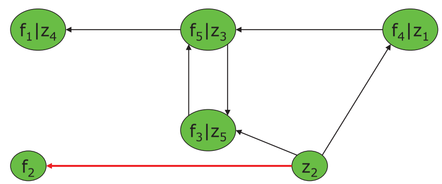
Construct a Directed Graph



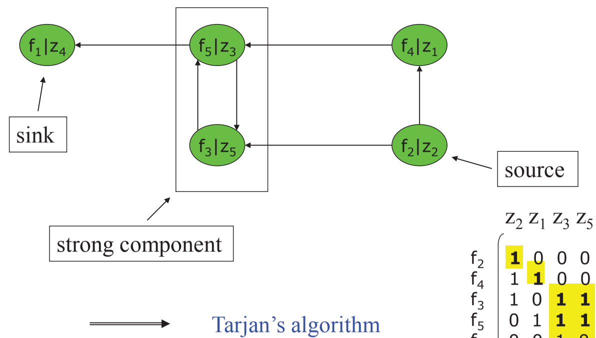
Construct a Directed Graph



Construct a Directed Graph



Construct a Directed Graph



	z_2	z_1	z_3	z_5	z_4
f_2	**1**	0	0	0	0
f_4	1	**1**	0	0	0
f_3	1	0	**1**	**1**	0
f_5	0	1	**1**	**1**	0
f_1	0	0	1	0	**1**

Further Efficiency Issues - Dummy-Derivative Method

- Matching algorithm fails
  - System is structurally singular
  - Find minimal subset of equations
    - more equations than unknown variables
  - Singularity is due to equations, constraining states
- Differentiate subset of equations
  - Static state selection during compile time
    - choose one state and corresponding derivative as purely algebraic variable
      - so-called dummy state and dummy derivative
    - by differentiation introduced variables are algebraic
    - continue matching algorithm
    - check initial conditions
  - Dynamic state selection during simulation time
    - store information on constrained states
    - make selection dynamically based on stability criteria
    - new state selection triggers an event (re-initialize states)

Further Efficiency Issues – Algebraic Loops

- Solution of linear equation systems
  - Advanced solver packages (e.g. LAPACK) are used
  - Calculate LU-Decomposition for constant matrices
    - Small systems are inverted symbolically
- Solution of nonlinear systems
  - Advanced solver packages are used
    - Performance is depending on good starting values
  - Analytical Jacobian is provided symbolically
- Tearing systems of equations
  - Reducing the iteration variables dramatically
- Analytical Jacobians of the overall system
  - Minimize simulation/integration time needed

Outline

1. Excerpt of OpenModelica's symbolic machinery
2. Symbolically derived Jacobians
   i. Directional derivatives
   ii. Sparsity pattern
   iii. Coloring of the Jacobian
3. Nonlinear Optimal Control Problem
   i. General Discretization Scheme
   ii. Multiple Shooting/Collocation
   iii. Total Collocation
   iv. Applications
4. Lessons learned & Outlook

FH Bielefeld
University of
Applied Sciences

Fast Simulation of Fluid Models with Colored Jacobians

Willi Braun, Bernhard Bachmann
Department of Engineering and Mathematics
University of Applied Sciences Bielefeld

Stephanie Gallardo Yances, Kilian Link
Siemens AG, Energy Section
Erlangen

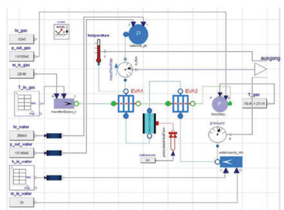
MODELICA

(see 9th International Modelica Conference)

FH Bielefeld
University of
Applied Sciences

Symbolically Generation of Jacobians

How is simulation time effected by Jacobians?



Fluid Test Model

States	231
Equations	942
Simulation time	10.8
J evaluations	111
J evaluation time	9.7

The evaluation of Jacobians effects the simulation time a lot!

Symbolically Generation of Jacobians

State-Space Equations

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Simulation

- Many integration algorithms need "the Jacobian" : $A(t) = \frac{\partial h}{\partial \underline{x}}$
- Integrator DASSL

Jacobian matrices

- $A(t) = \frac{\partial h}{\partial \underline{x}}$
- $B(t) = \frac{\partial h}{\partial \underline{u}}$
- $C(t) = \frac{\partial k}{\partial \underline{x}}$
- $D(t) = \frac{\partial k}{\partial \underline{u}}$

Symbolically Generation of Jacobians

Jacobian

$$J_A = \frac{\partial h}{\partial \underline{x}} = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n} \end{pmatrix}$$

Full Symbolic Jacobian

Generation of the full symbolic jacobian requires n-times differentiation of every equation.

Symbolically Generation of Jacobians

Jacobian

$$J_A = \frac{\partial h}{\partial \underline{x}} = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n} \end{pmatrix}$$

Generic Directional Derivative

$$J_A = \frac{\partial h}{\partial z}(\underline{e}_k)$$

$\underline{e}_k \in R^n := k -$ th coordinate vector

Symbolically Generation of Jacobians

Example

```
model twoflattankmodel
  Real h1, h2;
  Real F1, F2;
  input Real F;
  parameter Real A1=2, A2=0.5;
  parameter Real R1=2, R2=1;
equation
  der(h1) = (F/A1) - (F1/A1);
  der(h2) = (F1/A2) - (F2/A2);
  F1 = R1 * sqrt(h1 - h2);
  F2 = R2 * sqrt(h2);
end twoflattankmodel;
```

Jacobian

$$J_A = \frac{\partial h}{\partial z}\left(\frac{\partial x}{\partial z}\right) = \begin{pmatrix} \frac{\partial der(h1)}{\partial z}\left(\frac{\partial h1}{\partial z}, \frac{\partial h2}{\partial z}\right) \\ \frac{\partial der(h2)}{\partial z}\left(\frac{\partial h1}{\partial z}, \frac{\partial h2}{\partial z}\right) \end{pmatrix}$$

$$\frac{\partial der(h2)}{\partial z} = \frac{\frac{\partial F1}{\partial z} * A2 - F1 * \frac{\partial A2}{\partial z}}{A2^2} - \frac{\frac{\partial F2}{\partial z} * A2 - F2 * \frac{\partial A2}{\partial z}}{A2^2}$$

$$\frac{\partial F1}{\partial z} = R1 * \frac{\left(\frac{\partial h1}{\partial z} - \frac{\partial h2}{\partial z}\right)}{2 * \sqrt{h1 - h2}}$$

$$\frac{\partial F2}{\partial z} = R2 * \frac{\frac{\partial h2}{\partial z}}{2 * \sqrt{h2}}$$

$$\frac{\partial der(h1)}{\partial z} = \frac{\frac{\partial F}{\partial z} * A1 - F * \frac{\partial A1}{\partial z}}{A1^2} - \frac{\frac{\partial F1}{\partial z} * A1 - F1 * \frac{\partial A1}{\partial z}}{A1^2}$$

Symbolically Generation of Jacobians

Numerical

$$\frac{\partial h}{\partial \underline{x}} = \frac{(\underline{h}(\underline{x} + \delta \underline{e}_k) - \underline{h}(\underline{x}))}{\delta}$$

$\underline{e}_k \in R^n := k -$ th coordinate vector

Calculate the Jacobian numerical needs $n+1$ call of the ODE-Block $\underline{h}$.

Symbolical

$$J_A = \frac{\partial \underline{h}}{\partial \underline{z}}(\underline{e}_k)$$

$\underline{e}_k \in R^n := k -$ th coordinate vector

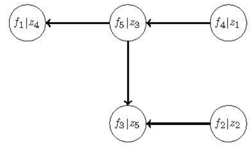
Evaluate the Jacobian symbolical needs n calls of Directional Derivative.

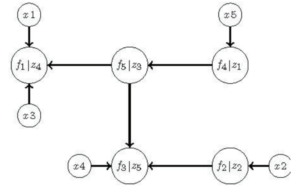
The amount of calls could be reduced by exploiting the **sparsity pattern** and partitioning the columns by colors.

Compute sparsity pattern of the Jacobians

Example system

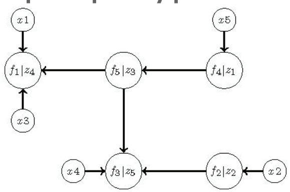
$$\underline{z}(t) = \underline{f}(\underline{x}(t), t)$$





$$J = \begin{pmatrix} 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \\ * & 0 & * & 0 & * \\ 0 & * & 0 & 0 & 0 \\ 0 & * & 0 & * & * \end{pmatrix}$$

Compute sparsity pattern of the Jacobians



$$J = \begin{pmatrix} 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \\ * & 0 & * & 0 & * \\ 0 & * & 0 & 0 & 0 \\ 0 & * & 0 & * & * \end{pmatrix}$$

	z1	z3	z4	z2	z5	x1	x2	x3	x4	x5
f4	1	0	0	0	0	0	0	0	0	1
f5	1	1	0	0	0	0	0	0	0	0
f1	0	1	1	0	0	1	0	1	0	0
f2	0	0	0	1	0	0	1	0	0	0
f3	0	1	0	1	1	0	0	0	1	0

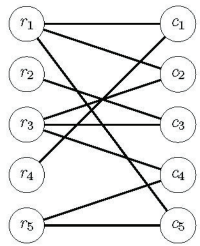
Accumulation Lists

f4: <5>
f5: <5>
f1: <1,3,5>
f2: <2>
f3: <5,2>

Utilize sparsity pattern of the Jacobians

Jacobian

$$J = \begin{pmatrix} j_{11} & j_{12} & 0 & 0 & j_{15} \\ 0 & 0 & j_{23} & 0 & 0 \\ 0 & j_{32} & j_{33} & j_{34} & 0 \\ j_{41} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & j_{54} & j_{55} \end{pmatrix}$$

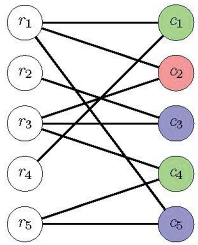


Utilize sparsity pattern of the Jacobians

Jacobian

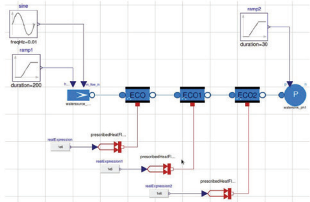
$$J = \begin{pmatrix} j_{11} & j_{12} & 0 & 0 & j_{15} \\ 0 & 0 & j_{23} & 0 & 0 \\ 0 & j_{32} & j_{33} & j_{34} & 0 \\ j_{41} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & j_{54} & j_{55} \end{pmatrix}$$

$$J_R = \begin{pmatrix} j_{11} & j_{12} & j_{15} \\ 0 & 0 & j_{23} \\ j_{34} & j_{32} & j_{33} \\ j_{41} & 0 & 0 \\ j_{54} & 0 & j_{55} \end{pmatrix}$$



MODELICA

Performance gain of implementation



Model details

States	231
Equations	1 006
JacElements	53 361
NonZero	3 032
Colors	79

Simulation statistics

method	steps	F-Eval	J-Eval	time
num	922	27184	111	10.8
numC	922	8929	94	4.5
sym	937	1539	103	8.5
symC	937	1539	103	4.3
Dymola	783	8772	90	1.6

MODELICA

Outline

1. Excerpt of OpenModelica's symbolic machinery

2. Symbolically derived Jacobians
   - i. Directional derivatives
   - ii. Sparsity pattern
   - iii. Coloring of the Jacobian

3. Nonlinear Optimal Control Problem
   - i. General Discretization Scheme
   - ii. Multiple Shooting/Collocation
   - iii. Total Collocation
   - iv. Applications

4. Lessons learned & Outlook

FH Bielefeld
University of
Applied Sciences

MODELICA

Parallel Multiple-Shooting and Collocation Optimization with OpenModelica

MODELICA

Bernhard Bachmann, Lennart Ochel, Vitalij Ruge
Mathematics and Engineering
University of Applied Sciences Bielefeld

Mahder Gebremedhin, Peter Fritzson,
PELAB – Programming Environment Lab

Vaheed Nezhadali, Lars Eriksson, Martin Sivertsson
Vehicular Systems
Linköping University

(see 9th International Modelica Conference)

FH Bielefeld
University of
Applied Sciences

Linköping University

Nonlinear Optimal Control Problem (NOCP)

Mathematical problem formulation

– objective function

$$\min_{u(t)} J(x(t),u(t),t) = E\left(x(t_f)\right) + \int_{t_0}^{t_f} L(x(t),u(t),t)\,dt$$

– subject to

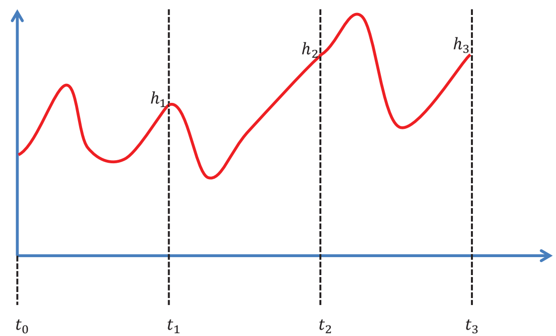
$x(t_0)$	$=$	h_0	initial conditions
$\dot{x}(t)$	$=$	$f(x(t),u(t),t)$	DAEs, Modelica
$g(x(t),y(t),u(t),t)$	$\geq$	0	path constraints
$r\left(x(t_f),y(t_f)\right)$	$=$	0	terminal constraints

Theoretical Background

General discretization scheme

$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t),u(t),t)\,dt$$

$$x_i(t_i) = h_i$$

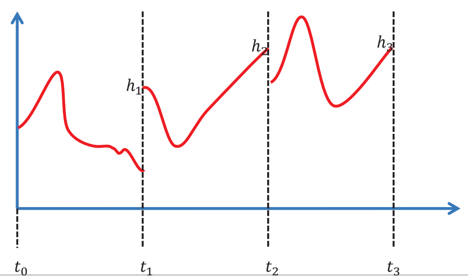


Theoretical Background

Multiple Shooting/Collocation

- Solve sub-problem in each sub-interval

$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t),u(t),t)\,dt \approx F(t_i,t_{i+1},h_i,u_i), \qquad x_i(t_i) = h_i$$



Theoretical Background

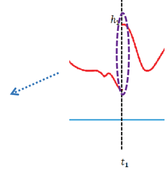
Multiple Shooting / Collocation Optimization

- Discretized Nonlinear Optimal Control Problem
  – objective function (integral approximation by trapezoidal rule)

$$\min_{u(t)} J(x(t),u(t),t) = E(h_n) + \frac{\Delta t}{2}\sum_{i=0}^{n-1} L(h_i,u_i,t_i) + L(h_{i+1},u_i,t_{i+1})$$

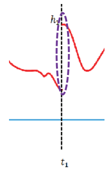
  – subject to

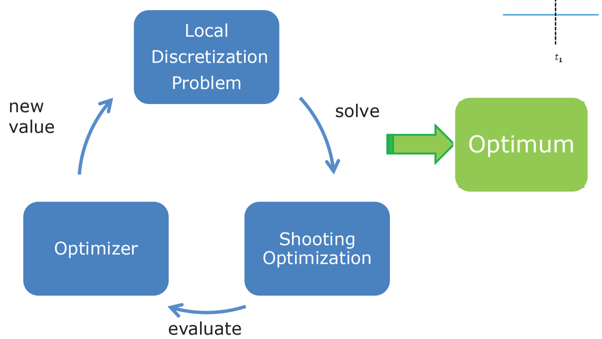
$x(t_0)$	$=$	h_0
$F(t_i,t_{i+1},h_i,u_i)$	$=$	h_{i+1}
$g(h_i,u_i,t_i)$	$\geq$	0
$g(h_{i+1},u_i,t_{i+1})$	$\geq$	0



Theoretical Background

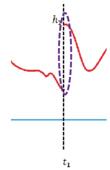
Multiple Shooting / Collocation Optimization

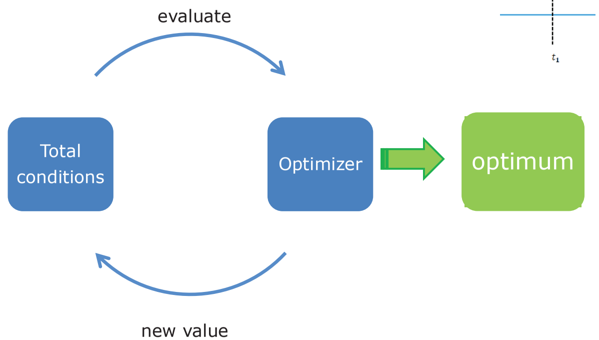




Theoretical Background

Total Collocation Optimization





Theoretical Background

Total Collocation Optimization

- Discretized Nonlinear Optimal Control Problem
  - objective function (integral approximation by Gauß quadrature)

$$\min_{u(t)} J(x(t),u(t),t) = E(h_n) + \Delta t \sum_{j=0}^{m} w_j \cdot \sum_{i=0}^{n-1} L\left(h_i^{(j)}, u_i, t_i + s_j\right)$$

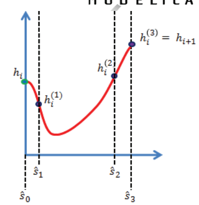
  - subject to

$$\begin{aligned} x(t_0) &= h_0 \\ g(h_i, u_i, t_i) &\geq 0 \\ g(h_{i+1}, u_i, t_{i+1}) &\geq 0 \end{aligned}$$

additional collocation conditions

Theoretical Background

Collocation Condition – Approximation of States



- Assumption:
  States are locally polynomial

$$x_i(t_i + \hat{s} \cdot \Delta t) = p_0(\hat{s}) \cdot h_{i-1}^{(m)} + \sum_{j=1}^{m} p_j(\hat{s}) \cdot h_i^{(j)}$$

where $x_i(t_i + \hat{s}_k \cdot \Delta t) = \delta_{k,0} \cdot h_{i-1}^{(m)} + \sum_{j=1}^{m} \delta_{k,j} \cdot h_i^{(j)} = h_i^{(k)}$

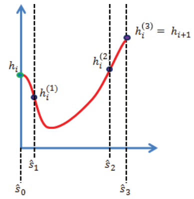
$\hat{s}_k$ are the Radau points

$p_j(\hat{s})$ are the Lagrange Basis polynomial to the nodes $\hat{s}_k$

- Collocation conditions

$$\Delta t \cdot f\left(h_i^{(j)}, u_i, t_i + \hat{s}_k \cdot \Delta t\right) = p_0'(\hat{s}_k) \cdot h_{i-1}^{(m)} + \sum_{j=1}^{m} p_j'(\hat{s}_k) \cdot h_i^{(j)}$$

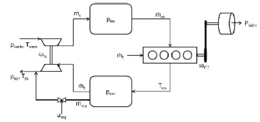
Theoretical Background
Collocation Condition – Approximation of State Derivatives



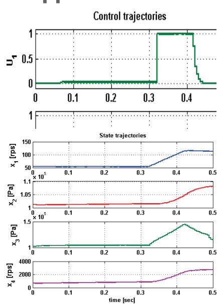
- Assumption:
  State derivatives are locally polynomial
  $$\Delta t \cdot f(x_i(t_i + \hat{s} \cdot \Delta t), u_i, t_i + \hat{s} \cdot \Delta t) = \sum_{j=0}^{m} p_j(\hat{s}) \cdot f_i^{(j)}$$
  where $\Delta t \cdot f\left(h_i^{(k)}, u_i, t_i + \hat{s}_k \cdot \Delta t\right) = \sum_{j=0}^{m} \delta_{k,j} \cdot f_i^{(j)} = f_i^{(k)}$
  $\hat{s}_j$ are the Lobatto points
  $p_j(\hat{s})$ are the Lagrange Basis polynomial to the nodes $\hat{s}_j$
- Collocation conditions
  $$h_i^{(k)} = \sum_{j=0}^{m} \int p_j(\hat{s}_k) \cdot f_i^{(j)} + h_{i-1}^{(m)}$$

Applications – Diesel Electric Powertrain

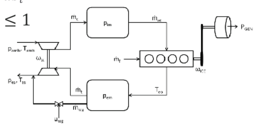
- Find fuel optimal control and state trajectories from idling condition to a certain power level
- Nonlinear mean value engine model
- Only diesel operating condition
- Mathematical problem formulation:
  - 2 inputs (u_f, u_{wg})
  - 4 states $(\omega_{ice}, p_{im}, p_{em}, \omega_{tc})$
  - 32 algebraic equations



Applications – Diesel Electric Powertrain



- Mathematical problem formulation
  - Object function
  $$\min_{u(t)} \sum_{i=1}^{4} \left(x_i(t_f) - x_i^{ref}\right)^2 + \int_0^T \dot{m}_f \, dt$$
  - subject to
  $$\dot{x}_1 = f_1(x_2, x_3, u_1)$$
  $$\dot{x}_2 = f_2(x_1, x_2, x_4)$$
  $$\dot{x}_3 = f_3(x_1, x_2, x_3, u_1, u_2)$$
  $$\dot{x}_4 = f_4(x_2, x_3, x_4, u_2)$$
  $$x_{lb_i} \le x_i \le x_{ub_i}, \; i = 1, \dots, 4$$
  $$0 \le u_1, u_2 \le 1$$



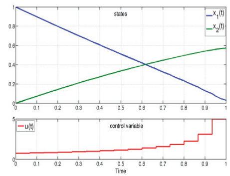
Engine is accelerated only near the end of the time interval to meet the end constraints while minimizing the fuel consumption

Implementation Details – Current Status

- Realization with OpenModelica Environment
- Optimica prototype implementation is available
- Using Ipopt for solution process
- Necessary derivatives are numerically calculated
  - Gradients, Jacobians, Hessians, …
- **But:** Complete tool chain not yet implemented

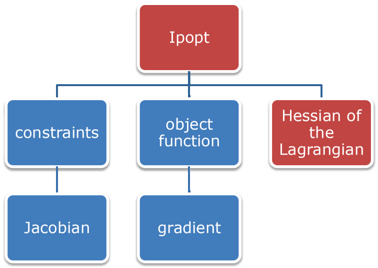
Test Environment

- Processor:
- 2xIntel Xeon CPU E5-2650
  16 cores @ 2.00GHz
- OpenMP



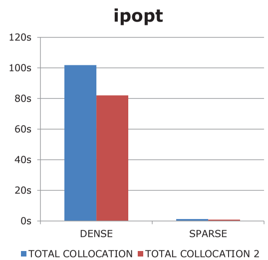
Implementation Details – Ipopt & Parallelization

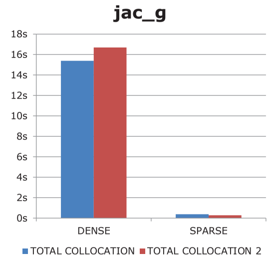
- Schematic view of the required components of Ipopt



Implementation Details - Numerical Optimization

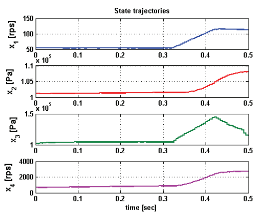
- Enormous speed-up when utilizing sparse Jacobian matrix
- Speed-up for the over-all optimization
- Sparse-structure model independent





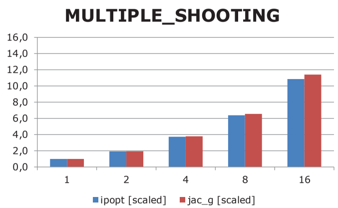
Results - Diesel Electric Powertrain

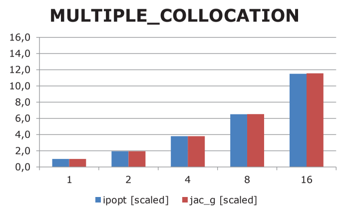
	Diesel
MULTIPLE SHOOTING	921,6s
MULTIPLE COLLOCATION	29519,8s
TOTAL COLLOCATION	9,5s
TOTAL COLLOCATION 2	15,6s



Results - Diesel Electric Powertrain

- Ipopt runs in serial mode
- Most execution time is elapsed in Jacobian calculation and solution process of the local discretization problem
- Reasonable speed-up
- Factors are non-optimal due to memory handling
  - Further investigations will be performed

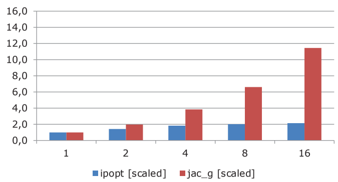




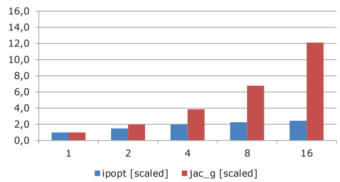
Results - Diesel Electric Powertrain

- Ipopt runs in serial mode
- Less execution time is elapsed in Jacobian calculation
- Reasonable speed-up for Jacobian calculation
- Factors are non-optimal due to memory handling
- Overall Speed-up increases with model complexity
  - Parallelizing of Ipopt necessary

TOTAL COLLOCATION



TOTAL COLLOCATION 2



Lessons learned

- Symbolic calculation of derivatives improve performance
  - Jacobian, Gradient, …
- Utilizing sparsity pattern is crucial
- In serial mode total collocation methods are superior to multiple shooting/collocation methods
- Parallelizing the algorithms performs better on multiple shooting/collocation methods
- Symbolic transformation to ODE form is a key issue for the realization of an automatic tool chain

Example 1 – From the dark side (Francesco Casella)

```
optimization Example1A(
  objective = a1/b1*(x1 - x10)^2 +
              a2/b2*(x2 - x20)^2 +
              a3/b3*(x3 - x30)^2,
...
end Example1A;
```

```
optimization Example1B(
  objective = f1 + f2 + f3,
...
equation
  f1 = a1/b1*(x1 - x10)^2;
  f2 = a2/b2*(x2 - x20)^2;
  f3 = a3/b3*(x3 - x30)^2;
...
end Example1B;
```


Efficient Symbolical and Numerical Algorithms for nonlinear model predictive control with OpenModelica
Bernhard Bachmann, et. al.
MODELICA

Example 2 – From the dark side (Francesco Casella)

```
optimization Example2A(
  parameter Real PR = 10;
...
equation
  p_in/p_out = PR "Turbine pressure ratio";
...
end Example1A;
```

```
optimization Example2A(
  parameter Real PR = 10;
...
equation
  p_in = PR*p_out "Turbine pressure ratio";
...
end Example1A;
```

Future work

- Implement complete tool chain in OpenModelica
- Automatic generation of simulation code based on Optimica
- Utilizing symbolically derived derivative information
  - Gradient, Jacobian, Hessian, …
- Further improvements with appropriate scaling
- Exploiting parallel evaluation of the optimization method
- Advanced use of OMC symbolic machinery
  - Efficient handling of model dependent algebraic loops
- Generalization of NOCP problem formulation
  - e.g. time minimal optimization, parameter estimation

- Further testing on industrial-relevant problems

Thank you

Questions?

MODELING SEEN AS PROGRAMMING

Klaus Havelund, Jet Propulsion Laboratory, California Institute of Technology

Code generation often requires the model to be concrete, at which point the distinction between model and code disappears from a philosophical point of view, and we are back in the situation where there is only code. If the purpose of the model is to be an alternative statement of the solution, which the code can be checked against, this approach fails to deliver that. Verification of code against model is challenging and suffers from computational complexity. Models can, however, be used for monitoring program execution. In this approach, often referred to as runtime verification, code is instrumented to emit events when executed. The generated execution trace (a sequence of events) is then monitored against the model, and if a discrepancy is detected according to the model, an error can be reported. Runtime verification can be performed during testing, either as the system executes, or post-mortem, by analyzing generated logs; or it can be performed during the actual operation of the software. We shall demonstrate an RV system called TraceContract, which in essence is an API in the high-level Scala programming language. The API offers a range of methods for writing models that are suited for trace analysis. This includes data parameterized state machines, temporal logic, and rule-based programming. Common for these techniques is the reliance on rewriting as the basis for the implementation. We argue that for certain forms of trace analysis, and modeling in general, the best weapon is a high level programming language augmented with constructs for temporal reasoning.

Modeling Seen as Programming

Klaus Havelund
NASA JPL, California Inst. of Technology, USA

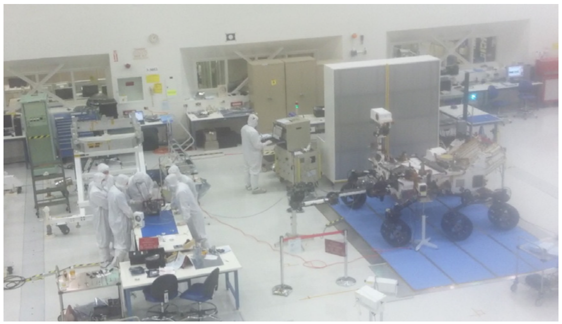
System Design meets Equation-based Languages

September 21, 2012

Acknowledgements

Part of the work described in this publication was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

MSL



Landing

3.5 million lines of C code

Terminology

Terminology

- model engineering

Terminology

- model engineering = engineering models

Terminology

- model engineering = engineering models
- model-based engineering

Terminology

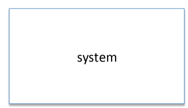
- model engineering = engineering models
- model-based engineering
- mode-based programming

Terminology

- model engineering = engineering models
- model-based engineering
- mode-based programming
- models, specifications used in software engineering (formal methods)

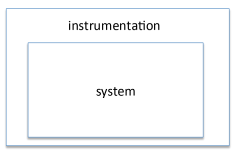
Runtime verification

- Start with a system to monitor.



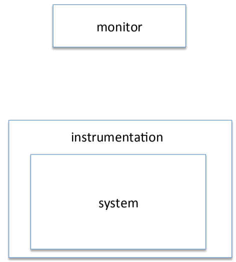
Runtime verification

- *Instrument* the system to record relevant events.



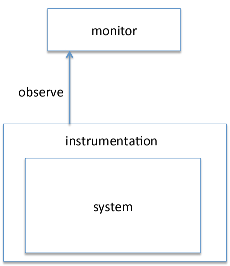
Runtime verification

- *Provide* a monitor.



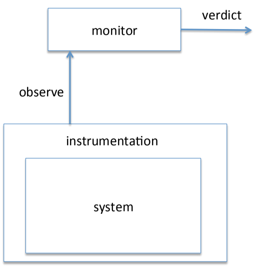
Runtime verification

- *Dispatch* each received event to the monitor.



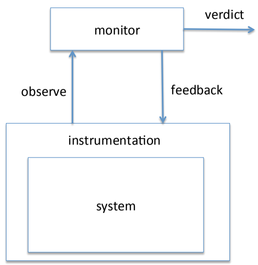
Runtime verification

- Compute a *verdict* for the trace received so far.



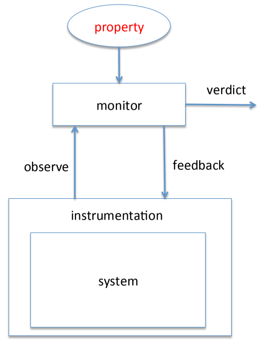
Runtime verification

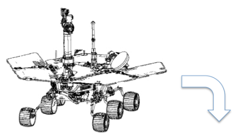
- Possibly generate *feedback* to the system.



Runtime verification

- We might possibly have synthesized monitor from a *property*.





```
COMMAND("STOP_CAMERA",1,22:50.00)
COMMAND("ORIENT_ANTENNA_TOWARDS_GROUND",2,22:50.10)
SUCCESS("ORIENT_ANTENNA_TOWARDS_GROUND",3,22:52.02)
COMMAND("STOP_CAMERA",4,22:55.01)
SUCCESS("ORIENT_ANTENNA_TOWARDS_GROUND",5,22:56.19)
COMMAND("STOP_ALL",6,23:01.10)
FAIL("ORIENT_ANTENNA_TOWARDS_GROUND",7,23:02.02)
```

requirements relating events across time

External versus internal DSL

External versus internal DSL

- External DSL

External versus internal DSL

- External DSL
  - small language typically with very focused functionality

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - "easy" to learn for person not familiar with programming language

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - "easy" to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - "easy" to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - “easy” to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL
  - an extension of an existing programming language

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - “easy” to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL
  - an extension of an existing programming language
  - typically an API - using base language’s features only

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - “easy” to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL
  - an extension of an existing programming language
  - typically an API - using base language’s features only
  - pros:

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - “easy” to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL
  - an extension of an existing programming language
  - typically an API - using base language’s features only
  - pros:
    - easier to develop and later adapt

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - "easy" to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL
  - an extension of an existing programming language
  - typically an API - using base language's features only
  - pros:
    - easier to develop and later adapt
    - expressive, the programming language is never far away

External versus internal DSL

- External DSL
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - "easy" to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL
  - an extension of an existing programming language
  - typically an API - using base language's features only
  - pros:
    - easier to develop and later adapt
    - expressive, the programming language is never far away
    - allows use of existing tools such as type checkers, IDEs, etc.

External versus internal DSL

- External DSL: LogScope
  - small language typically with very focused functionality
  - specialized parser
  - pros:
    - can be optimally succinct
    - "easy" to learn for person not familiar with programming language
    - analyzable: a spec can be analyzed easily, visualized, etc.
- Internal DSL: TraceContract
  - an extension of an existing programming language
  - typically an API - using base language's features only
  - pros:
    - easier to develop and later adapt
    - expressive, the programming language is never far away
    - allows use of existing tools such as type checkers, IDEs, etc.

LogScope V1 syntax

LogScope V2 syntax

```
rule_schema ::=
    modifier+ "{" transition+ "}"
  | modifier* ident ["(" ident,* ")"] ["{" transition+ "}"]

modifier ::=
    "init" | "always" | "step" | "next" | "hot"

transition ::= pattern,* "=>" pattern,*

pattern ::= ["!"] ident ["(" constraint,* ")"]

constraint ::=
   ident ":" range
 | range
```

Quote

Hemmingway & Hotchner, 1920ies:

If you are lucky enough to have lived in Paris as a young man, then wherever you go for the rest of your life, it stays with you, for Paris is a moveable feast.

Quote

Havelund, 2012:

If you are lucky enough to have explored VDM as a young man, then wherever you go for the rest of your life, it stays with you, for VDM is a moveable feast.

What is VDM?

- Combination of imperative and functional programming (data types, pattern matching, curried functions, lambda abstractions, side effects, loops, exceptions,)
- Design-by-contract: pre/post conditions + invariants
- Predicate subtypes
- Non-deterministic expressions (let x be such that P(x))
- First order predicate logic as Boolean expressions: universal and existential quantification
- Sets, lists and maps as built-in data types
- VDM^{++} added object orientation (Nico Plat et. al)

Chemical plant model in VDM versus Scala

```
class Plant

instance variables

alarms   : set of Alarm;
schedule : map Period to set of Expert;
inv PlantInv(alarms,schedule);

operations

PlantInv: set of Alarm * map Period to set of Expert ==>
          bool
PlantInv(as,sch) ==
  return
  (forall p in set dom sch & sch(p) <> {}) and
  (forall a in set as &
     forall p in set dom sch &
       exists expert in set sch(p) &
         a.GetReqQuali() in set expert.GetQuali());

types

public Period = token;

operations

public ExpertToPage: Alarm * Period ==> Expert
ExpertToPage(a, p) ==
  let expert in set schedule(p) be st
      a.GetReqQuali() in set expert.GetQuali()
  in
    return expert
pre a in set alarms and
    p in set dom schedule
post let expert = RESULT
      in
        expert in set schedule(p) and
        a.GetReqQuali() in set expert.GetQuali();
```

```
class Plant(alarms: Set[Alarm],
    schedule: Map[Period, Set[Expert]]) {
  assert(PlantInv(alarms, schedule))

  def PlantInv(alarms: Set[Alarm], schedule: Map[Period,
    Set[Expert]]): Boolean =
    (schedule.keySet forall { schedule(_) != Set() }) &&
      (alarms forall { a =>
        schedule.keySet forall { p =>
          schedule(p) exists { expert =>
            a.reqQuali ? expert.quali
          }
        }
      })

  def ExpertToPage(a: Alarm, p: Period): Expert = {
    require(a ? alarms && p ? schedule.keySet)
    schedule(p) suchthat {expert =>
      a.reqQuali ? expert.quali}
  } ensuring { expert =>
    a.reqQuali ? expert.quali &&
      expert ? schedule(p)
  }
```

Scala is a high-level unifying language

- Object-oriented + functional programming features
- Strongly typed with type inference
- Script-like, semicolon inference
- Sets, list, maps, iterators, comprehensions
- Lots of libraries
- Compiles to JVM
- Lively growing community

Commands must succeed

- We are analyzing log files containing information about commands being issued, and their success and failure respectively.

Requirement CommandMustSucceed

An issued command must succeed, without a failure to occur before then.

Property in LogScope

- For comparison we first show spec in the external DSL: LogScope.
- a **hot** state must be exited before end of log (non-final state).

automaton CommandMustSucceed {
  always {
    Command(n,x) => RequireSuccess(n,x)
  }

  hot RequireSuccess(name,number) {
    Fail (name,number) => **error**
    Success(name,number) => **ok**
  }
}

Property in LogScope

- Using LogScope's temporal logic layer.

```
pattern CommandMustSucceed:
  Command(n,x) =>
    [
      ! Fail(n,x),
        Success(n,x),
    ]
```

Events in TraceContract

- First we need to define the events we observe:
  - commands being issued, each having a name and a number
  - successes of commands
  - failures of commands
- Each event type sub-classes a type: Event
- **case**-classes allow for pattern matching over objects of the class

```
abstract class Event

case class Command(name: String, nr: Int) extends Event
case class Success(name: String, nr: Int) extends Event
case class Fail(name: String, nr: Int) extends Event
```

Property in TraceContract - looks very similar

- Uses partial functions: {**case** ... =>...} defined with pattern matching as arguments to DSL functions (*require* and *hot*) defined in *Monitor* class. *RequireSuccess* is a user-defined function representing a state.
- A quoted name, such as 'name' represents the *value of* that name.

```
class CommandMustSucceed extends Monitor[Event] {
  always {
    case Command(n, x) => RequireSuccess(n, x)
  }

  def RequireSuccess(name: String, number: Int) =
    hot {
      case Fail('name', 'number') => error
      case Success('name', 'number') => ok
    }
}
```

Property in TraceContract - looks very similar

- Uses partial functions: {**case** ... =>...} defined with pattern matching as arguments to DSL functions (*require* and *hot*) defined in *Monitor* class. *RequireSuccess* is a user-defined function representing a state.
- A quoted name, such as 'name' represents the *value of* that name.

```
class CommandMustSucceed extends Monitor[Event] {
  require {
    case Command(n, x) => RequireSuccess(n, x)
  }

  def RequireSuccess(name: String, number: Int) =
    hot {
      case Fail('name', 'number') => error
      case Success('name', 'number') => ok
    }
}
```

Inlining the call of *RequireSuccess(n,x)*

- Since *RequireSuccess(n, x)* is a function, the call of it can be inlined.
- After all, this is "just" a program and standard program transformation works.
- The result is an interesting temporal logic like specification with an un-named hot state.

```
class CommandMustSucceed extends Monitor[Event] {
  require {
    case Command(n, x) =>
      hot {
        case Fail (`n`, `x`) => error
        case Success(`n`, `x`) => ok
      }
  }
}
```

Same property in LTL

- TRACECONTRACT also offers future time linear temporal logic (LTL).
- allowing to write events as formulas, negations, propositional formulas, and temporal.
- ϕ until ψ means: ψ must eventually hold, and until then ϕ must hold.

```
class CommandMustSucceed extends Monitor[Event] {
  require {
    case Command(n, x) =>
      not( Fail (n, x)) until (Success(n, x))
  }
}
```

Same property in LTL

- TRACECONTRACT also offers future time linear temporal logic (LTL).
- allowing to write events as formulas, negations, propositional formulas, and temporal.
- ϕ until ψ means: ψ must eventually hold, and until then ϕ must hold.

```
class CommandMustSucceed extends Monitor[Event] {
  require {
    case Command(n, x) =>
      not( Fail (n, x)) until (Success(n, x))
  }
}
```

- note mix of Scala's pattern matching (to catch arguments of command) and LTL.

Success of power commands

Requirement PowerCommandSuccess

Power commands must succeed within 10 seconds.

Property in LogScope

- Defining and using Python predicates in LogScope.

```
{:
def within(t1,t2,max):
  return (t2−t1) <= max
:}

pattern PowerCommands:
  Command(n, x, t1) where {: n.startswith("PWR") :} =>
    Success(n, x, t2) where {: within(t1,t2,10000) :}
```

Same property in TraceContract

- TraceContract allows direct integration of code and formulas.

```
class PowerCommands extends Monitor[Event] {
  def within(t1: Int,t2: Int, max: Int) = (t2−t1) <= max

  require {
    case Command(n, x, t1) if n.startsWith("PWR") =>
      hot {
        case Success('n', 'x', t2) if within(t1,t2,10000) => ok
      }
  }
}
```

10 first commands must succeed

Requirement First10CommandsMustSucceed

The first 10 issued commands must succeed, without a failure to occur before then.

Counting: first 10 commands must succeed

- Code (here counting and testing on counter) can be mixed with logic.
- That is: increase counter and return LTL formula.

```
class First10CommandsMustSucceed extends Monitor[Event] {
  var count = 0
  require {
    case Command(n, x) if count < 10 =>
      count = count + 1
      not(Fail(n, x)) until (Success(n, x))
  }
}
```

long sequence

Requirement CommandSequence

Whenever a flight software command is issued, there should follow a dispatch and then exactly one success.
No dispatch failure before the dispatch, and
no failure between dispatch and success.

Property in LogScope

- Using LogScope's sequence operator.

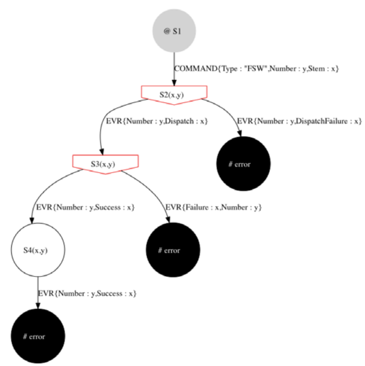
```
pattern CommandSequence:
  Command(n,x) =>
    [
      ! DispatchFailure(n,x),
        Dispatch(n,x),
      ! Fail(n,x),
        Success(n,x),
      ! Success(n,x)
    ]
```

Same property in TraceContract

- TraceContract allows mixing of states.

```
class CommandSequence extends Monitor[Event] {
  require {
    case Command(n, x) =>
      hot {
        case DispatchFailure('n', 'x') => error
        case Dispatch('n', 'x') =>
          hot {
            case Fail('n', 'x') => error
            case Success('n', 'x') =>
              state {
                case Success('n', 'x') => error
              }
          }
      }
  }
}
```

Visualization of LogScope statemachine



Much more difficult to do with internal DSL such as TraceContract.

Property that we cannot write in LogScope

- Antecedent (condition) containing multiple events.

```
pattern CommandSequenceAsCondition:
  [
      Command(n,x),
    ! DispatchFailure(n,x),
      Dispatch(n,x)
  ]
  =>
  [
    ! Fail(n,x),
      Success(n,x),
    ! Success(n,x)
  ]
```

However we can write it in TraceContract

- TraceContract by just changing one of the state modifiers.

```
class CommandSequence extends Monitor[Event] {
  require {
    case Command(n, x) =>
      state {
        case DispatchFailure('n', 'x') => error
        case Dispatch('n', 'x') =>
          hot {
            case Fail('n', 'x') => error
            case Success('n', 'x') =>
              state {
                case Success('n', 'x') => error
              }
          }
      }
  }
}
```

Some notes from a notebook - before TraceContract

```
First a spec in LogScope as it is:

monitor CommandsMustSucceed {
  always {
    COMMAND(name : x) => RequireSuccess(x)
  }

  hot RequireSuccess(cmdName) {
    FAIL(name : cmdName) => error
    SUCCESS(name : cmdName) => ok
  }
}

We can try to eliminate the  state RequireSuccess by simply inlining it:

monitor CommandsMustSucceed {
  always {
    COMMAND(name : x) => hot {
      FAIL(name : x) => error
      SUCCESS(name : x) => ok
    }
  }
}
```

TraceContract later offered this feature.

Alternation

Requirement AlternatingCommandSuccess

Commands and successes should alternate.

State machine solution

```
class AlternatingCommandSuccess extends Monitor[Event] {
  property(s1)

  def s1: Formula =
    state {
      case Command(n, x) => s2(n, x)
      case _ => error
    }

  def s2(name: String, number: Int) =
    state {
      case Success(`name`, `number`) => s1
      case _ => error
    }
}
```

State machine solution - with next-states

```
class AlternatingCommandSuccess extends Monitor[Event] {
  property(s1)

  def s1: Formula =
    next {
      case Command(n, x) => s2(n, x)
    }

  def s2(name: String, number: Int) =
    next {
      case Success(`name`, `number`) => s1
    }
}
```

A past time property

- Properties so far have been future time properties: from some event, the future behavior must satisfy some property.
- The following requirement refers to the past of some event (success).

Requirement SuccessHasAReason
A success must be caused by a previously issued command.

TraceContract offers limited rule-based programming

- State logic and LTL cannot express this property.

TraceContract offers limited rule-based programming

- State logic and LTL cannot express this property.
- TraceContract offers a limited form of rule-based programming, were a fact f (sub-classing class *Fact*) can be queried ($f?$), created ($f+$), and deleted ($f-$). The result in the latter two cases is True.

TraceContract offers limited rule-based programming

- State logic and LTL cannot express this property.
- TraceContract offers a limited form of rule-based programming, were a fact f (sub-classing class *Fact*) can be queried ($f?$), created ($f+$), and deleted ($f-$). The result in the latter two cases is True.

```
class SuccessHasAReason extends Monitor[Event] {
  case class Commanded(name: String, nr: Int) extends Fact

  require {
    case Command(n, x) => Commanded(n, x) +
    case Success(n, x) =>
      if (Commanded(n, x) ?)
        Commanded(n, x) -
      else
        error
  }
}
```

The ?- abbreviation

- We can we make this monitor simpler by using test-and-set: $f\ ?-$, for a given fact f, meaning: *return true iff. the fact f is recorded, delete the fact in any case.*

```
class SuccessHasAReason extends Monitor[Event] {
  case class Commanded(name: String, nr: Int) extends Fact

  require {
    case Command(n, x) => Commanded(n, x) +
    case Success(n, x) => Commanded(n, x) ?-
  }
}
```

Making monitors of monitors

- We can create a new monitor which includes other monitors as sub-monitors. Useful for organizing properties.
- The semantics is the obvious one of conjunction: all monitors will get checked individually.

```
class CommandRequirements extends Monitor[Event] {
  monitor(
    new CommandMustSucceed,
    new MaxOneSuccess,
    new SuccessHasAReason)
}
```

Analyzing a complete trace (log analysis)

- To verify a trace: first create it, then instantiate monitor, and call *verify* method on monitor with trace as argument.

```
object TraceAnalysis extends Application {
  val trace: List[Event] =
    List(
      Command("STOP_DRIVING", 1),
      Command("TAKE_PICTURE", 2),
      Fail("STOP_DRIVING", 1),
      Success("TAKE_PICTURE", 2),
      Success("SEND_TELEMETRY", 42))

  val monitor = new CommandRequirements
  monitor.verify(trace)
}
```

Alternatively: analyzing event by event (online monitoring)

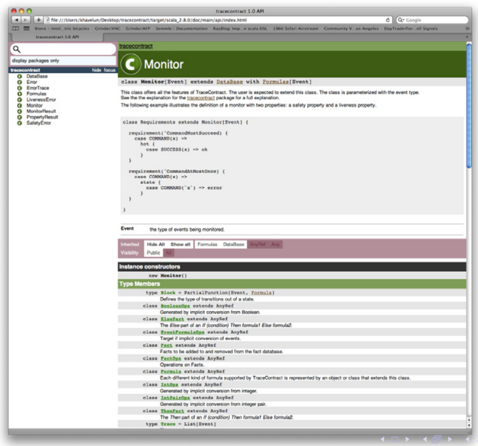
- To verify a sequence of events: instantiate monitor, and call *verify* method on monitor for each event, and call *end()* if event flow terminates.

```
object TraceAnalysis extends Application {
  val monitor = new CommandRequirements
  monitor.verify(Command("STOP_DRIVING", 1))
  monitor.verify(Command("TAKE_PICTURE", 2))
  monitor.verify(Fail("STOP_DRIVING", 1))
  monitor.verify(Success("TAKE_PICTURE", 2))
  monitor.verify(Success("SEND_TELEMETRY", 42))
  monitor.end()
}
```

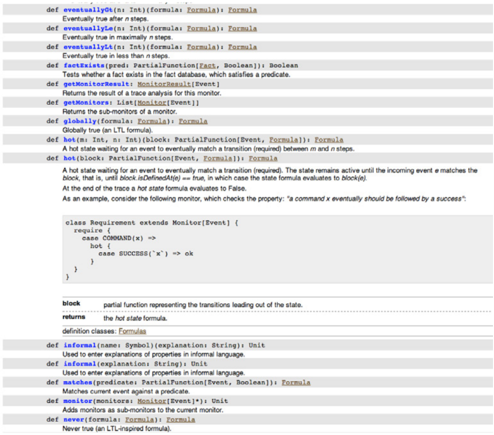
Result

```
CommandMustSucceed property violated
Violating event number 3: Fail(STOP_DRIVING,1)
Error trace:
  1=Command(STOP_DRIVING,1)
  3=Fail(STOP_DRIVING,1)

SuccessHasAReason property violated
Violating event number 5: Success(SEND_TELEMETRY,42)
Error trace:
  5=Success(SEND_TELEMETRY,42)
```

ScalaDoc documentation of API

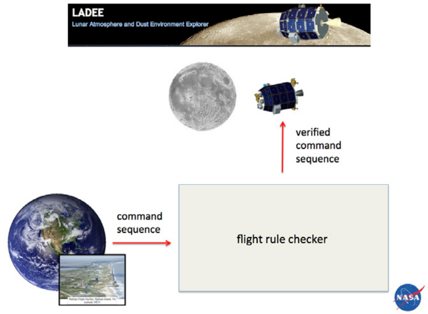


ScalaDoc documentation of API

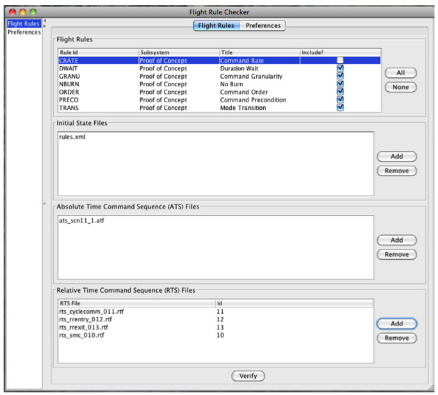


```
def eventuallyGt(n: Int)(formula: Formula): Formula
    Eventually true after n steps.
def eventuallyLe(n: Int)(formula: Formula): Formula
    Eventually true in maximally n steps.
def eventuallyLt(n: Int)(formula: Formula): Formula
    Eventually true in less than n steps.
def factExists(pred: PartialFunction[Fact, Boolean]): Boolean
    Tests whether a fact exists in the fact database, which satisfies a predicate.
def getMonitorResult: MonitorResult[Event]
    Returns the result of a trace analysis for this monitor.
def getMonitors: List[Monitor[Event]]
    Returns the sub-monitors of a monitor.
def globally(formula: Formula): Formula
    Globally true (an LTL formula).
def hot(m: Int, n: Int)(block: PartialFunction[Event, Formula]): Formula
    A hot state waiting for an event to eventually match a transition (required) between m and n steps.
def hot(block: PartialFunction[Event, Formula]): Formula
    A hot state waiting for an event to eventually match a transition (required). The state remains active until the incoming event e matches the
    block, that is, until block.isDefinedAt(e) == true, in which case the state formula evaluates to block(e).
    At the end of the trace a hot state formula evaluates to False.
    As an example, consider the following monitor, which checks the property: "a command x eventually should be followed by a success":

    class Requirement extends Monitor[Event] {
      require {
        case COMMAND(x) =>
          hot {
            case SUCCESS(`x`) => ok
          }
      }
    }

    block    partial function representing the transitions leading out of the state.
    returns  the hot state formula.
    definition classes: Formulas
def informal(name: Symbol)(explanation: String): Unit
    Used to enter explanations of properties in informal language.
def informal(explanation: String): Unit
    Used to enter explanations of properties in informal language.
def matches(predicate: PartialFunction[Event, Boolean]): Formula
    Matches current event against a predicate.
def monitor(monitors: Monitor[Event]*): Unit
    Adds monitors as sub-monitors to the current monitor.
def never(formula: Formula): Formula
    Never true (an LTL-inspired formula).
```

LADEE mission



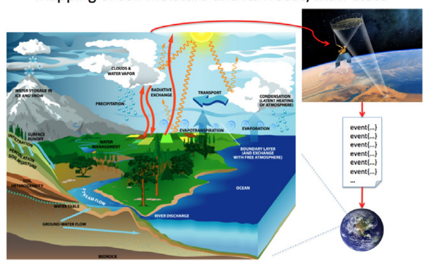
GUI interface to TraceContract (LADEE mission)



SMAP mission

SMAP

mapping of soil moisture and its freeze/thaw state



Definition of parameterized monitors

```
class CommandSuccess(cmd: String, success: Boolean = true)
extends Monitor[Event] {
    require {
      case Command(`cmd`,number) =>
        hot {
          case Success(`cmd`,`number`) => success
          case Fail (`cmd`,`number`) => !success
        }
    }
}

monitor(new CommandSuccess("STOP"))
```

Summary

- TraceContract is an API.
- Very expressive and convenient for programmers to use.
- For this reason mainly it has been adopted by practitioners.
- Has very simple implementation, which is easy to modify.
- Change requests are easy to process.
- It is, however, difficult to analyze a TraceContract specification since it fundamentally is a `Scala` program - requires some form of reflection or interaction with compiler.
- It will not be suitable for non-`Scala` programmers.

VERIFICATION OF STIFF HYBRID SYSTEMS BY MODELING THE APPROXIMATIONS OF COMPUTATIONAL SEMANTICS

Pieter J. Mosterman, MathWorks

With the seemingly unbounded proliferation of computing power into most any engineered artifact, ever more 'smart' systems are being created. This increase of available smarts in engineered systems has given rise to a new field of innovation where unique value is derived from having intelligent systems interact in novel and unforeseen manners. With the physical world an intrinsic part of the interaction and the smarts being implemented in a networked information modality, also called cyber space, these innovative systems are referred to as Cyber-Physical Systems. Modeling cyber aspects, physics, and their nexus then plays a crucial role in the design of Cyber-Physical Systems. A pick-and-place machine is presented as a paradigmatic example of such Cyber-Physical Systems to illustrates the intricate interplay between cyber space and physics, which serves to motivate the importance of integrated heterogeneous modeling paradigms that support modeling, simulation, and analysis of combined physics, geometry, signal processing, and control aspects. At a macroscopic level, physics models often comprise differential and algebraic equations and these equations typically require computational approaches to derive solutions. Approximations introduced by the solvers that derive these solutions to a large extent determine the meaning of the models, in particular when continuous-time behavior interacts with discontinuities such as in so-called hybrid dynamic systems. In reasoning about models that are solved computationally it is therefore imperative to also model the solvers. This presentation outlines an approach to modeling numerical solver approximations to help reason about approximations and to enable verification of stiff hybrid dynamic systems.

Modeling Approximation of Computational Semantics for Cyber-Physical System Design

Pieter J. Mosterman

Senior Research Scientist
Design Automation Department
MathWorks

Adjunct Professor
School of Computer Science
McGill

In your opinion, what lasting legacy has YACC brought to language development?

YACC made it possible for many people who were not language experts to make little languages (also called domain-specific languages) to improve their productivity. Also, the design style of YACC - base the program on solid theory, implement the theory well, and leave lots of escape hatches for the things you want to do that don't fit the theory - was something many Unix utilities embodied. It was part of the atmosphere in those days, and this design style has persisted in most of my work since then.

Interview with Stephen C. Johnson in "The A-Z of programming languages: YACC," *Computerworld*, 09.07.2008
http://news.idg.no/cw/art.cfm?id=094E3B6E-17A4-0F78-311509693E8E95C1

2

Modeling Approximation of Computational Semantics for Cyber-Physical System Design

Pieter J. Mosterman

Senior Research Scientist
Design Automation Department
MathWorks

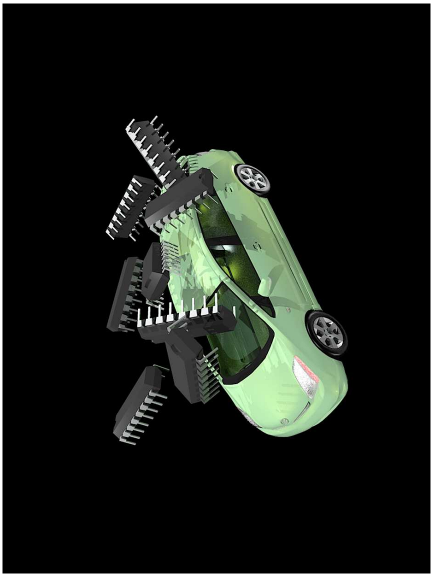
Adjunct Professor
School of Computer Science
McGill

© 2012 The MathWorks, Inc.

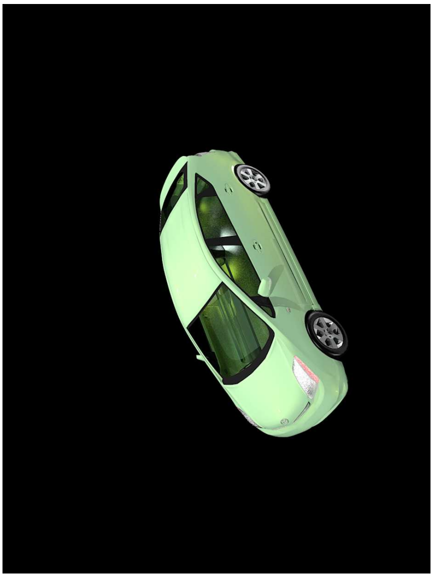
Agenda

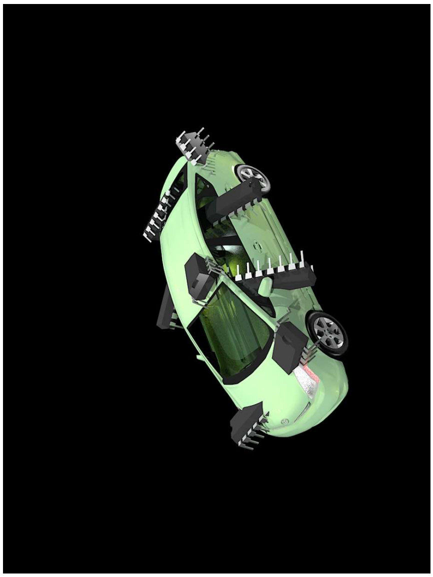
- Cyber-physical systems
- Modeling cyber-physical systems
- Modeling approximations
- A solver model for control synthesis
- Conclusions

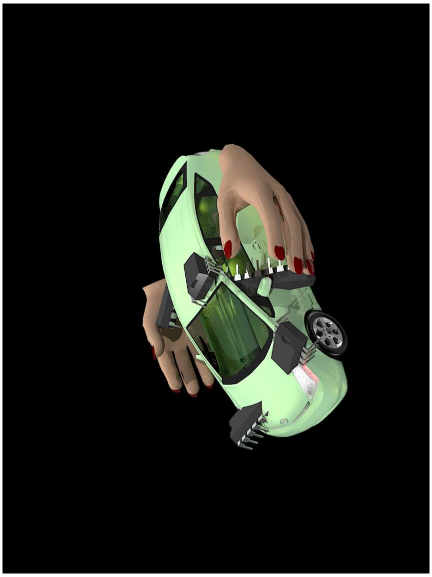
4

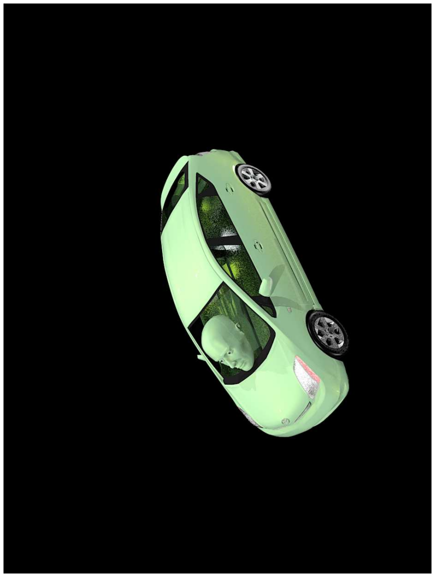


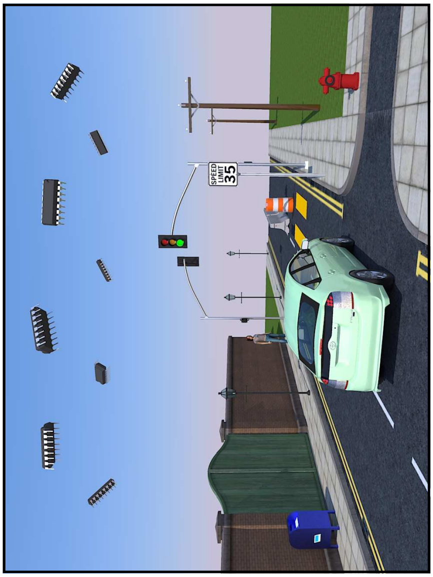

System Integration
Timing
Concurrency
Interfaces
Shared resources
...

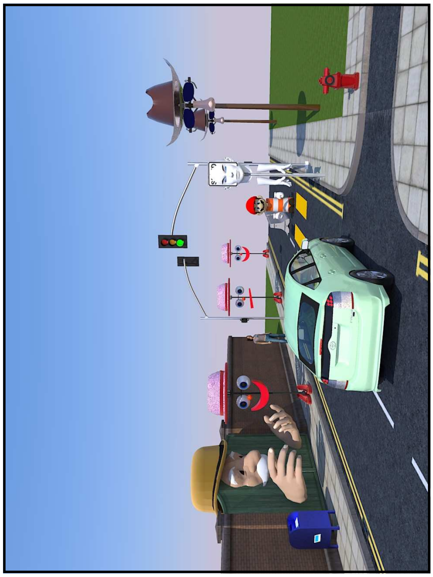


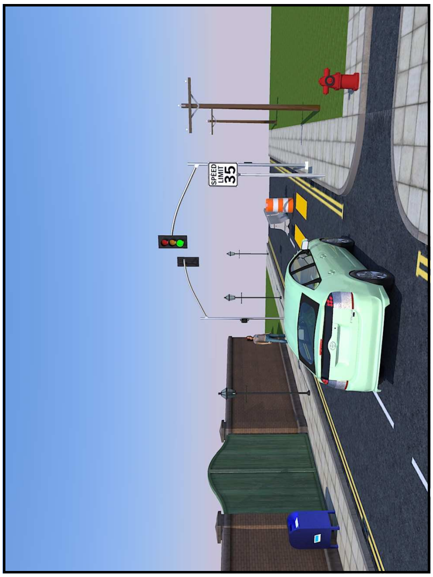


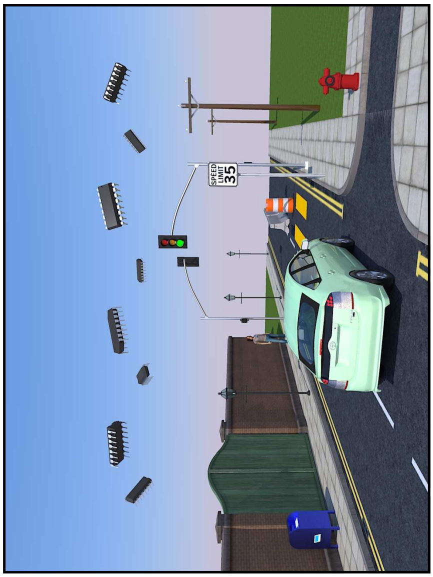


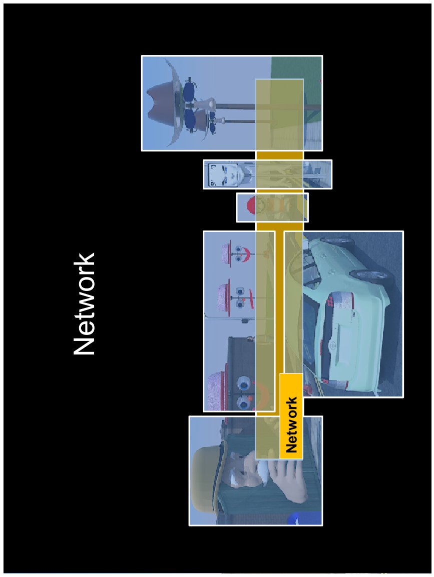


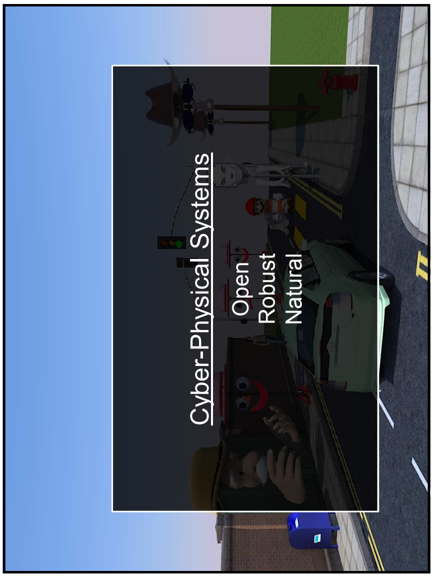

SPEED
LIMIT
35

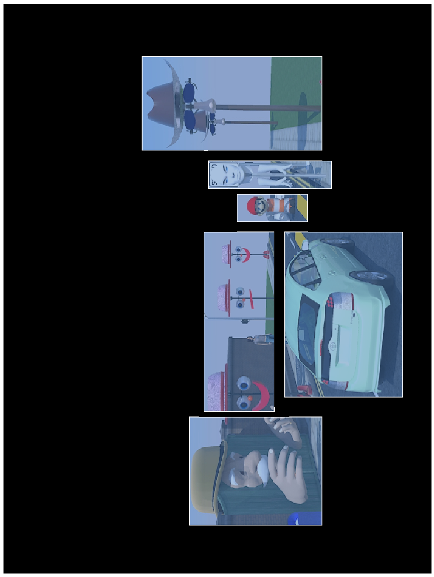


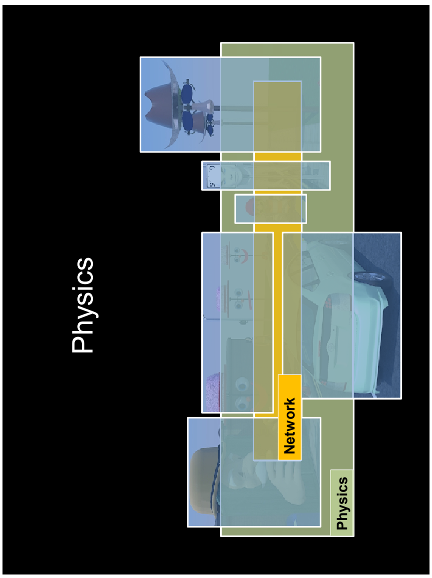

SPEED
LIMIT
35

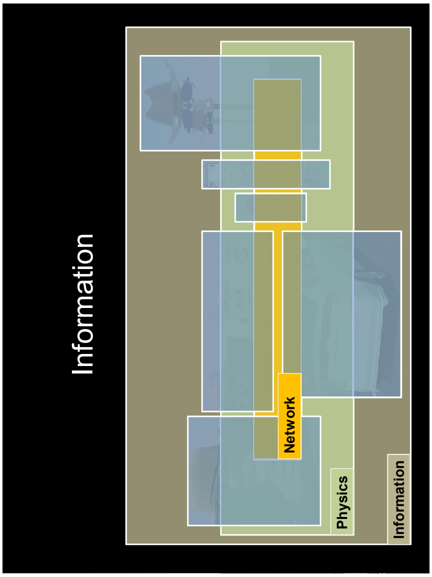

SPEED
LIMIT
35

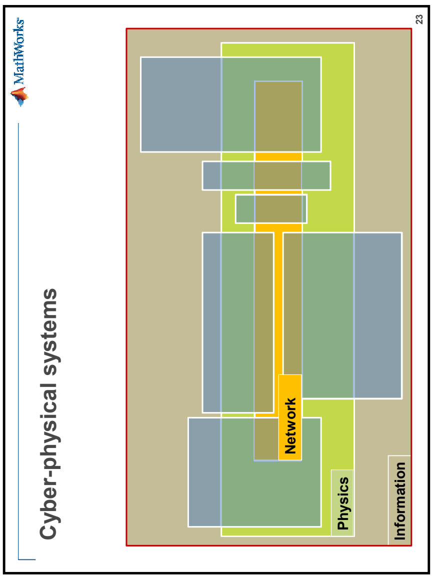

Network
Network

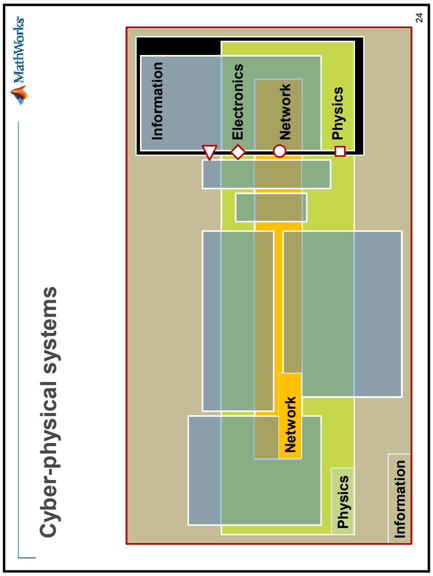

Cyber-Physical Systems
Open
Robust
Natural

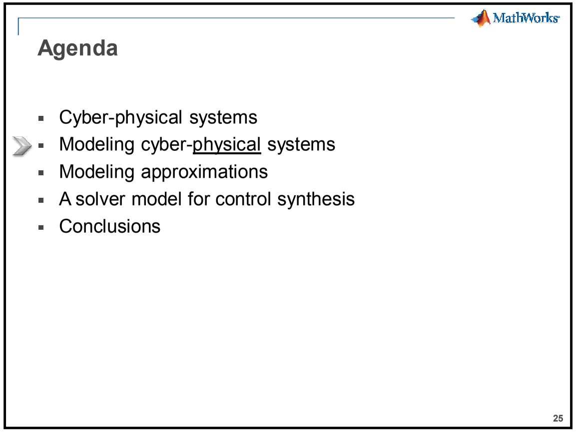


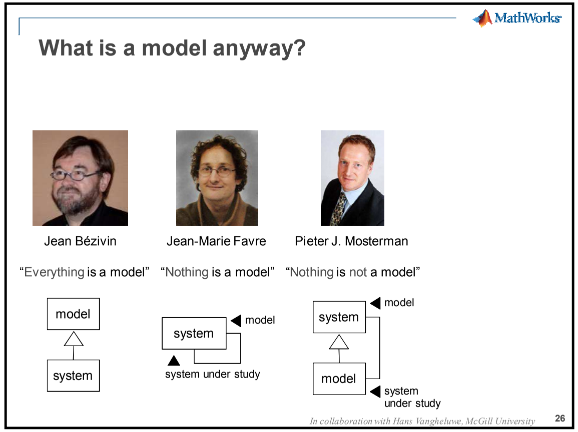

Physics
Physics
Network

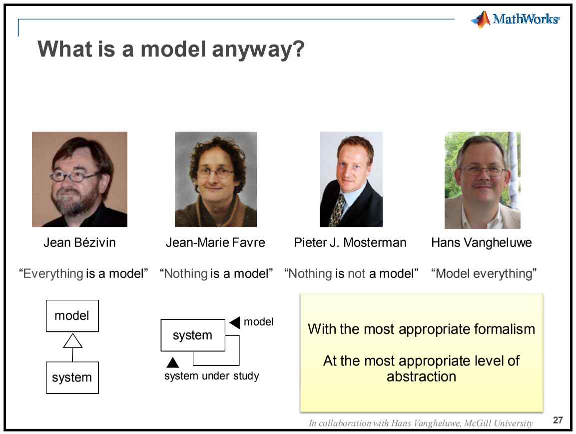

Information
Information
Physics
Network

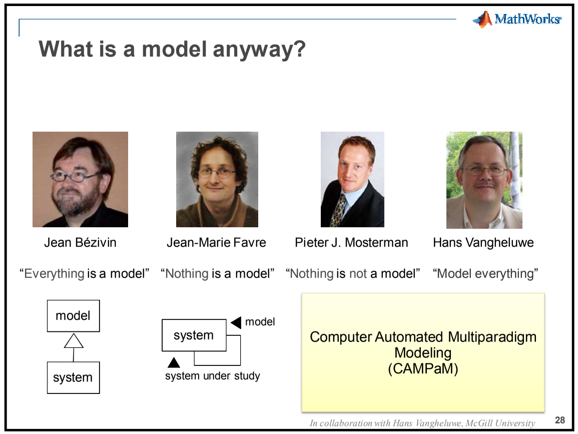

Cyber-physical systems
MathWorks
Information
Physics
Network
23

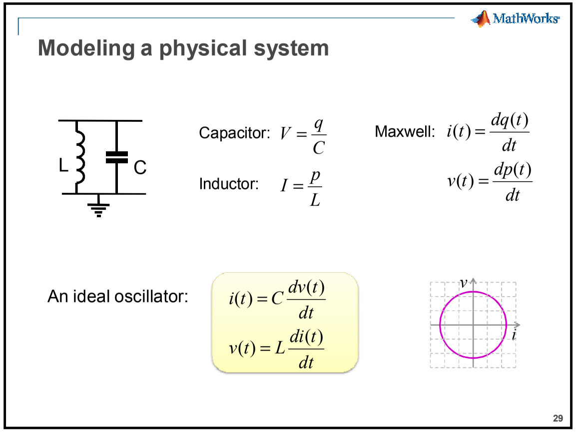

Cyber-physical systems
MathWorks
Information
Physics
Network
Information
Electronics
Network
Physics
24

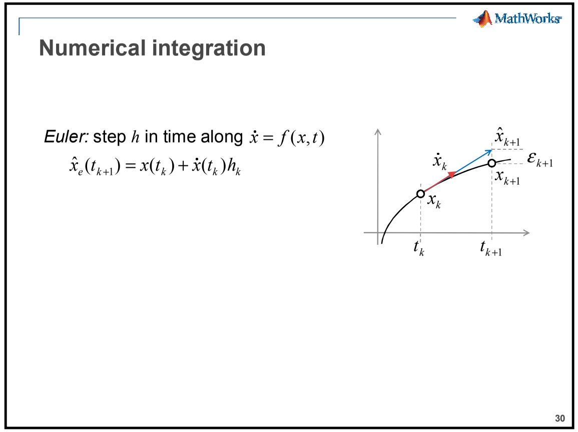

MathWorks
Agenda
Cyber-physical systems
Modeling cyber-physical systems
Modeling approximations
A solver model for control synthesis
Conclusions
25

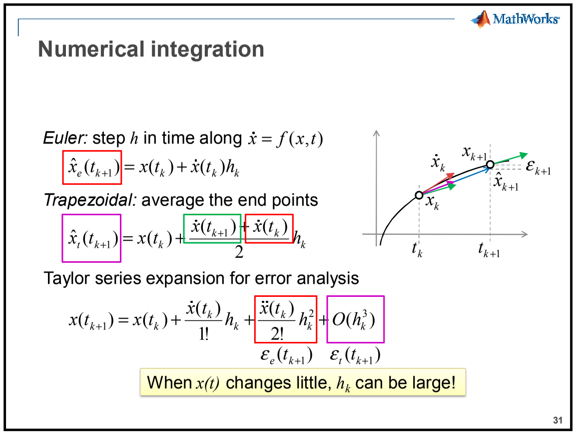

MathWorks
What is a model anyway?
Jean Bézivin
Jean-Marie Favre
Pieter J. Mosterman
"Everything is a model"
"Nothing is a model"
"Nothing is not a model"
model
system
system
model
system under study
system
model
model
system under study
In collaboration with Hans Vangheluwe, McGill University
26

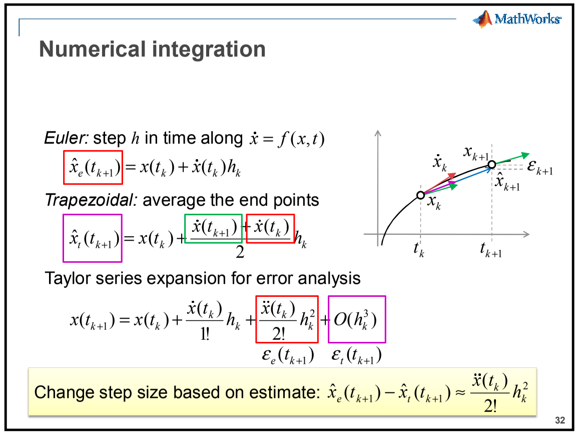

MathWorks
What is a model anyway?
Jean Bézivin
Jean-Marie Favre
Pieter J. Mosterman
Hans Vangheluwe
"Everything is a model"
"Nothing is a model"
"Nothing is not a model"
"Model everything"
model
system
system
model
system under study
With the most appropriate formalism
At the most appropriate level of abstraction
In collaboration with Hans Vangheluwe, McGill University
27

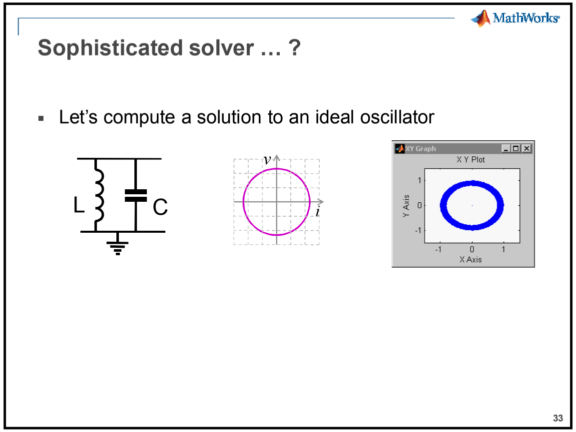

MathWorks
What is a model anyway?
Jean Bézivin
Jean-Marie Favre
Pieter J. Mosterman
Hans Vangheluwe
"Everything is a model"
"Nothing is a model"
"Nothing is not a model"
"Model everything"
model
system
system
model
system under study
Computer Automated Multiparadigm Modeling (CAMPaM)
In collaboration with Hans Vangheluwe, McGill University
28

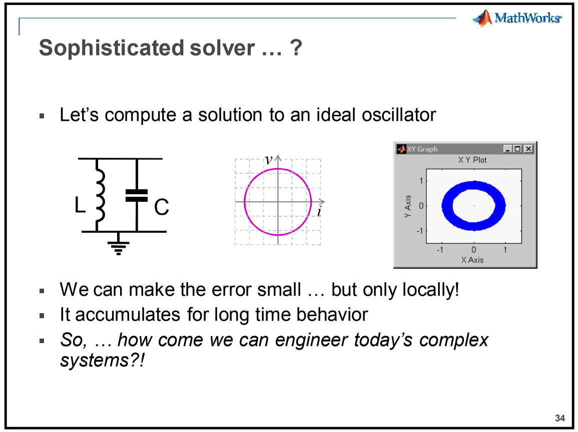

MathWorks
Modeling a physical system
L
C
Capacitor: $V = \frac{q}{C}$
Inductor: $I = \frac{p}{L}$
Maxwell: $i(t) = \frac{dq(t)}{dt}$
$v(t) = \frac{dp(t)}{dt}$
An ideal oscillator:
$i(t) = C\frac{dv(t)}{dt}$
$v(t) = L\frac{di(t)}{dt}$
v
i
29

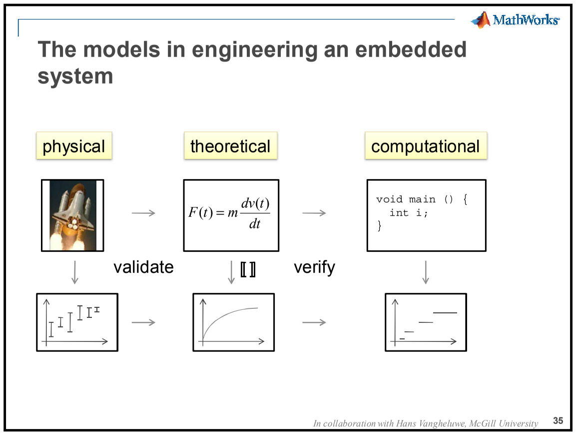

MathWorks
Numerical integration
Euler: step h in time along $\dot{x} = f(x,t)$
$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k)h_k$
$\hat{x}_{k+1}$
$\dot{x}_k$
ε_{k+1}
x_{k+1}
x_k
t_k
t_{k+1}
30

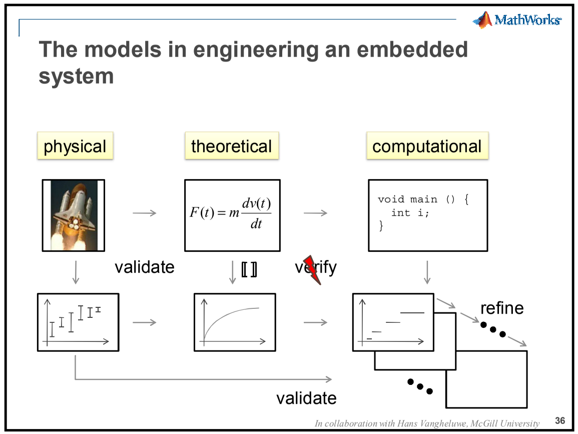

MathWorks
Numerical integration
Euler: step h in time along $\dot{x} = f(x,t)$
$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k)h_k$
Trapezoidal: average the end points
$\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2}h_k$
Taylor series expansion for error analysis
$x(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_k)}{1!}h_k + \frac{\ddot{x}(t_k)}{2!}h_k^2 + O(h_k^3)$
$\varepsilon_e(t_{k+1})$ $\varepsilon_t(t_{k+1})$
When $x(t)$ changes little, h_k can be large!
$\dot{x}_k$
x_{k+1}
ε_{k+1}
$\hat{x}_{k+1}$
x_k
t_k
t_{k+1}
31

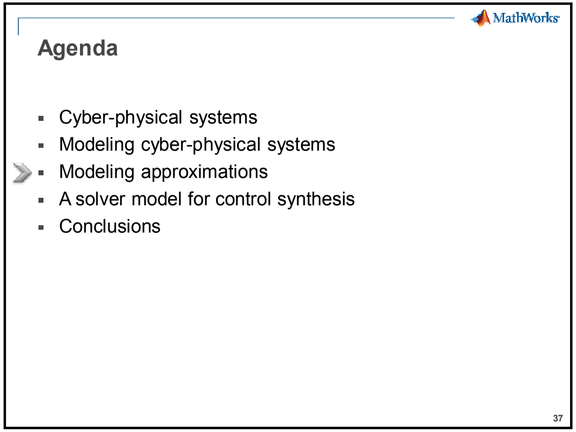

MathWorks
Numerical integration
Euler: step h in time along $\dot{x} = f(x,t)$
$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k)h_k$
Trapezoidal: average the end points
$\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2}h_k$
Taylor series expansion for error analysis
$x(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_k)}{1!}h_k + \frac{\ddot{x}(t_k)}{2!}h_k^2 + O(h_k^3)$
$\varepsilon_e(t_{k+1})$ $\varepsilon_t(t_{k+1})$
Change step size based on estimate: $\hat{x}_e(t_{k+1}) - \hat{x}_t(t_{k+1}) \approx \frac{\ddot{x}(t_k)}{2!}h_k^2$
$\dot{x}_k$
x_{k+1}
ε_{k+1}
$\hat{x}_{k+1}$
x_k
t_k
t_{k+1}
32

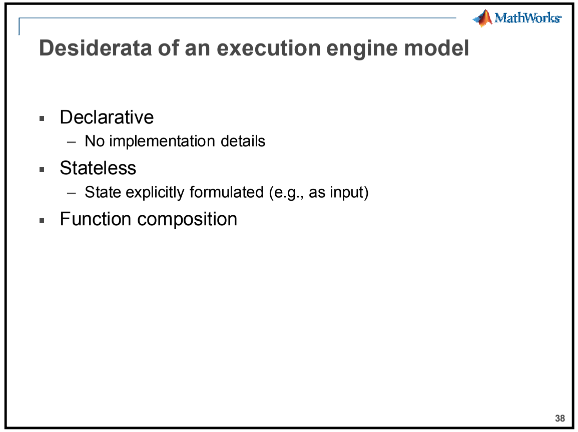

MathWorks
Sophisticated solver … ?
Let's compute a solution to an ideal oscillator
L
C
v
i
XY Graph
X Y Plot
Y Axis
X Axis
33

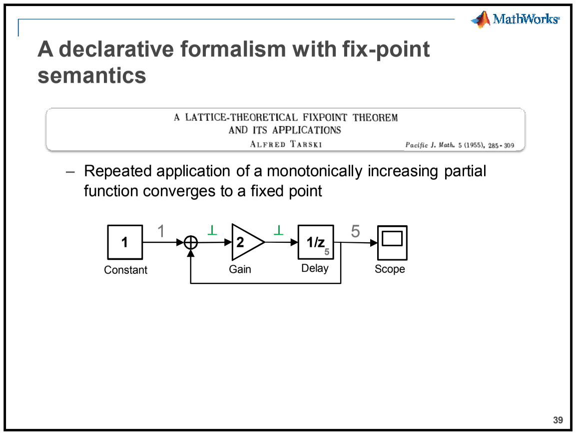

MathWorks
Sophisticated solver … ?
Let's compute a solution to an ideal oscillator
L
C
v
i
XY Graph
X Y Plot
Y Axis
X Axis
We can make the error small … but only locally!
It accumulates for long time behavior
So, … how come we can engineer today's complex systems?!
34

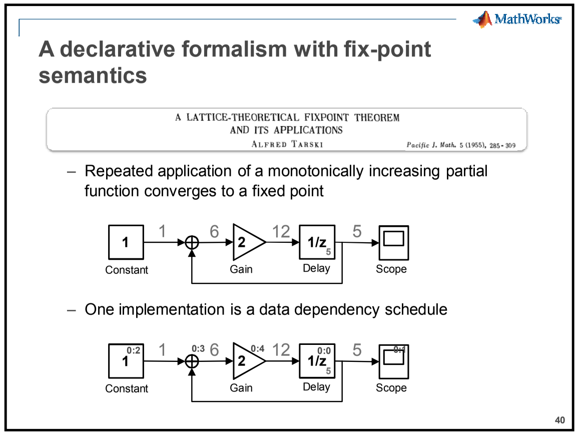

MathWorks
The models in engineering an embedded system
physical
theoretical
computational
F(t) = m dv(t)/dt
void main () {
int i;
}
validate
verify
In collaboration with Hans Vangheluwe, McGill University
35

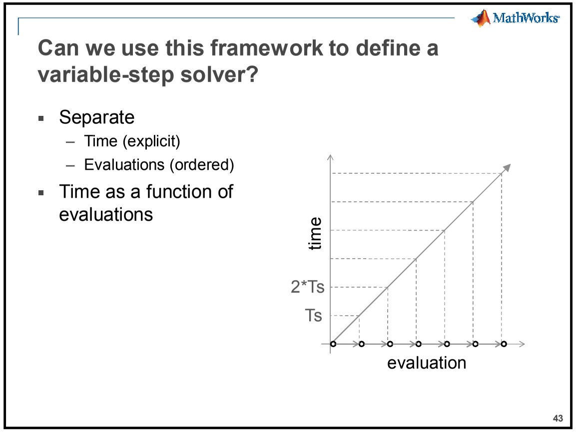

MathWorks
The models in engineering an embedded system
physical
theoretical
computational
F(t) = m dv(t)/dt
void main () {
int i;
}
validate
verify
refine
validate
In collaboration with Hans Vangheluwe, McGill University
36

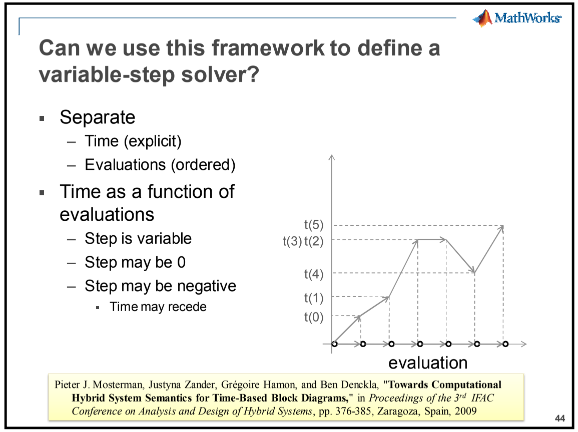

MathWorks
Agenda
Cyber-physical systems
Modeling cyber-physical systems
Modeling approximations
A solver model for control synthesis
Conclusions
37

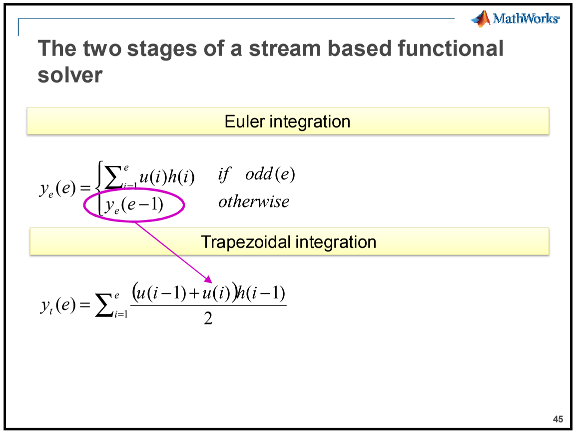

MathWorks
Desiderata of an execution engine model
Declarative
– No implementation details
Stateless
– State explicitly formulated (e.g., as input)
Function composition
38

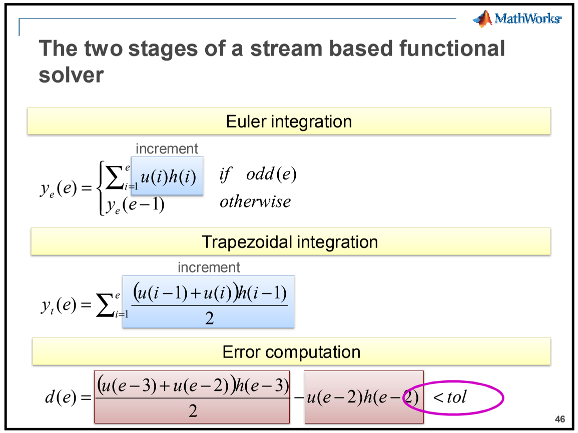

MathWorks
A declarative formalism with fix-point semantics
A LATTICE-THEORETICAL FIXPOINT THEOREM AND ITS APPLICATIONS
ALFRED TARSKI
Pacific J. Math. 5 (1955), 285-309
– Repeated application of a monotonically increasing partial function converges to a fixed point
1
⊥
⊥
5
Constant
Gain
Delay
Scope
39

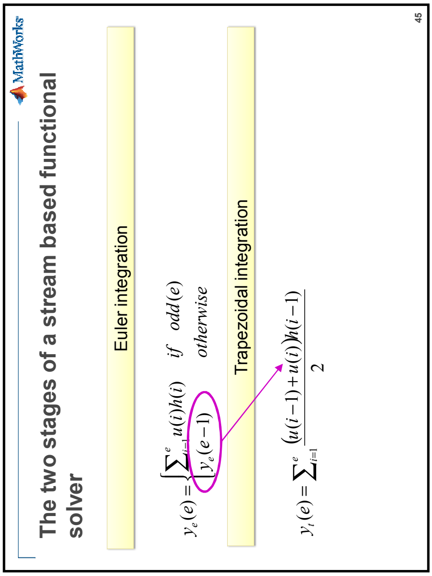

MathWorks
A declarative formalism with fix-point semantics
A LATTICE-THEORETICAL FIXPOINT THEOREM AND ITS APPLICATIONS
ALFRED TARSKI
Pacific J. Math. 5 (1955), 285-309
– Repeated application of a monotonically increasing partial function converges to a fixed point
1
6
12
5
Constant
Gain
Delay
Scope
– One implementation is a data dependency schedule
0:2
0:3
0:4
0:0
Constant
Gain
Delay
Scope
40

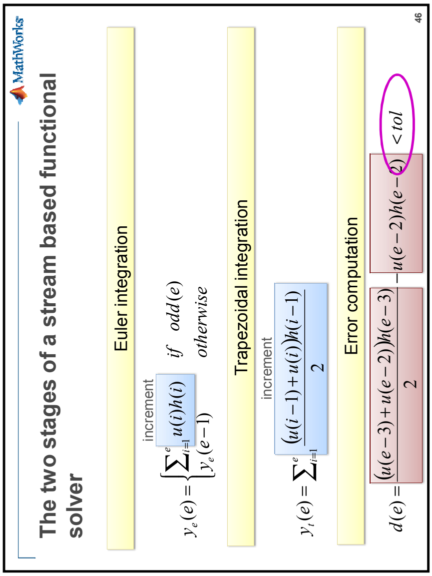

MathWorks
Can we use this framework to define a variable-step solver?
Separate
– Time (explicit)
– Evaluations (ordered)
Time as a function of evaluations
time
2*Ts
Ts
evaluation
43

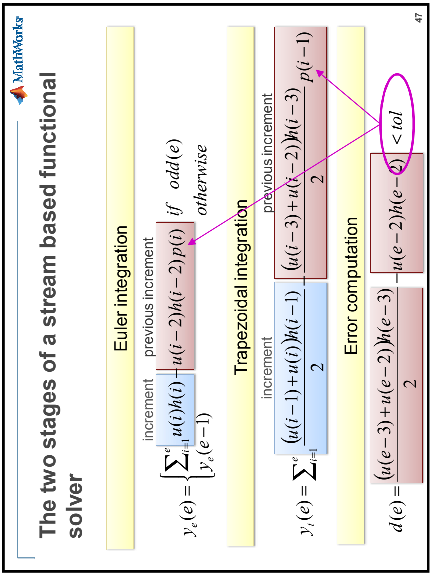

MathWorks
Can we use this framework to define a variable-step solver?
Separate
– Time (explicit)
– Evaluations (ordered)
Time as a function of evaluations
– Step is variable
– Step may be 0
– Step may be negative
Time may recede
t(5)
t(3) t(2)
t(4)
t(1)
t(0)
evaluation
Pieter J. Mosterman, Justyna Zander, Grégoire Hamon, and Ben Denckla, "Towards Computational Hybrid System Semantics for Time-Based Block Diagrams," in Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems, pp. 376-385, Zaragoza, Spain, 2009
44

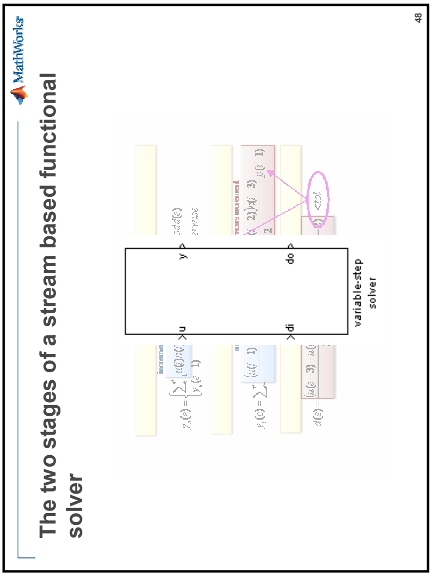

MathWorks
The two stages of a stream based functional solver
Euler integration
$y_e(e) = \begin{cases} \sum_{i=1}^{e} u(i)h(i) & if \quad odd(e) \\ y_e(e-1) & otherwise \end{cases}$
Trapezoidal integration
$y_t(e) = \sum_{i=1}^{e} \frac{(u(i-1)+u(i))h(i-1)}{2}$
45

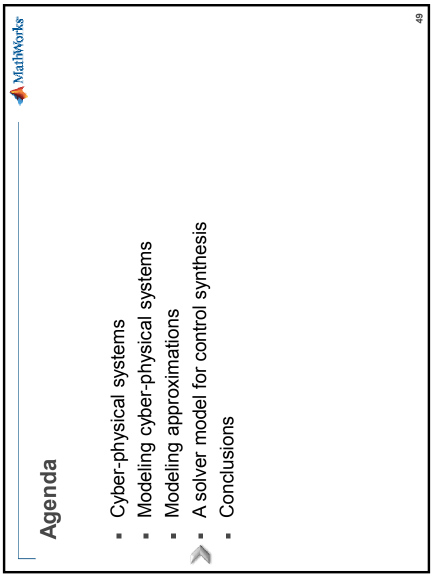

MathWorks
The two stages of a stream based functional solver
Euler integration
increment
$y_e(e) = \begin{cases} \sum_{i=1}^{e} u(i)h(i) & if \quad odd(e) \\ y_e(e-1) & otherwise \end{cases}$
Trapezoidal integration
increment
$y_t(e) = \sum_{i=1}^{e} \frac{(u(i-1)+u(i))h(i-1)}{2}$
Error computation
$d(e) = \frac{(u(e-3)+u(e-2))h(e-3)}{2} - u(e-2)h(e-2) < tol$
46

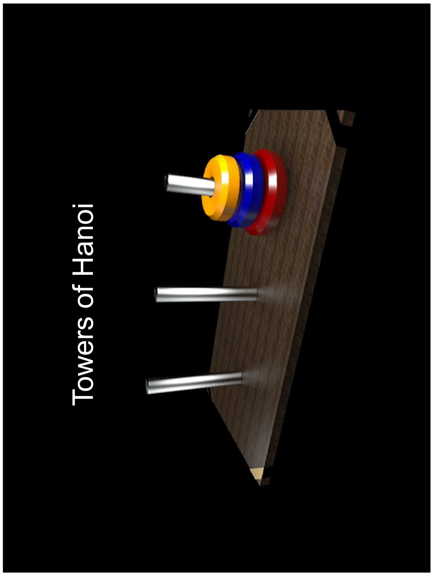

The two stages of a stream based functional solver
MathWorks
Euler integration
y_e(e) = { ∑_{i=1}^{e} u(i)h(i) if odd(e) ; y_e(e−1) otherwise
Trapezoidal integration
y_t(e) = ∑_{i=1}^{e} (u(i−1)+u(i))h(i−1) / 2
45


The two stages of a stream based functional solver
MathWorks
Euler integration
increment
y_e(e) = { ∑_{i=1}^{e} u(i)h(i) if odd(e) ; y_e(e−1) otherwise
Trapezoidal integration
increment
y_t(e) = ∑_{i=1}^{e} (u(i−1)+u(i))h(i−1) / 2
Error computation
d(e) = (u(e−3)+u(e−2))h(e−3) / 2 − u(e−2)h(e−2) < tol
46

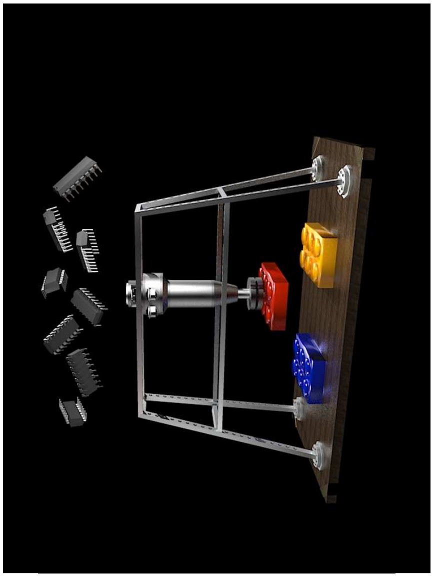

The two stages of a stream based functional solver
MathWorks
Euler integration
increment
previous increment
y_e(e) = { ∑_{i=1}^{e} u(i)h(i) − u(i−2)h(i−2)p(i) if odd(e) ; y_e(e−1) otherwise
Trapezoidal integration
increment
previous increment
y_t(e) = ∑_{i=1}^{e} (u(i−1)+u(i))h(i−1) / 2 − (u(i−3)+u(i−2))h(i−3) / 2 p(i−1)
Error computation
d(e) = (u(e−3)+u(e−2))h(e−3) / 2 − u(e−2)h(e−2) < tol
47


The two stages of a stream based functional solver
MathWorks
u
y
di
do
variable-step solver
48

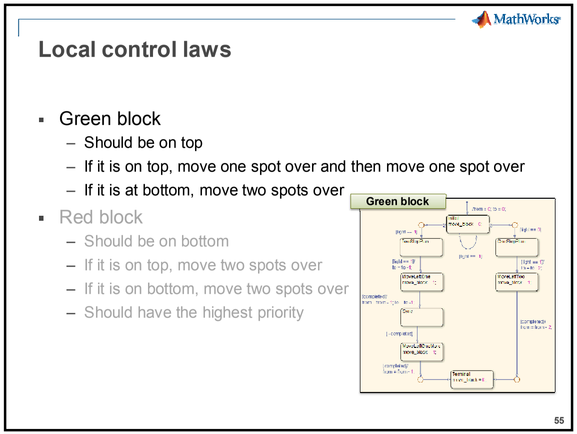

Agenda
Cyber-physical systems
Modeling cyber-physical systems
Modeling approximations
A solver model for control synthesis
Conclusions
MathWorks
49

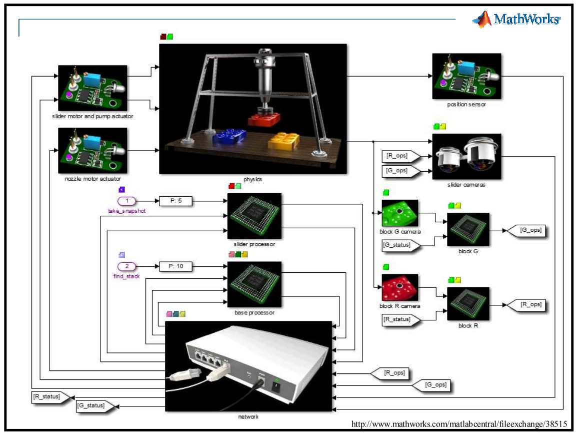

Towers of Hanoi

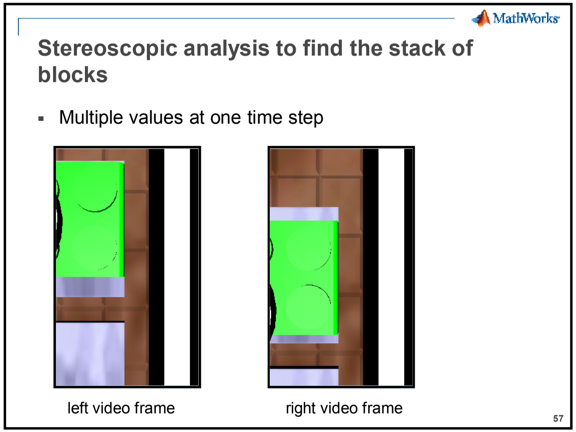

A Cyber-Physical System?

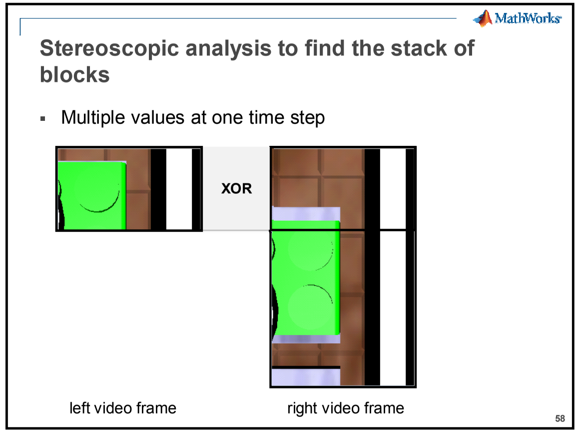


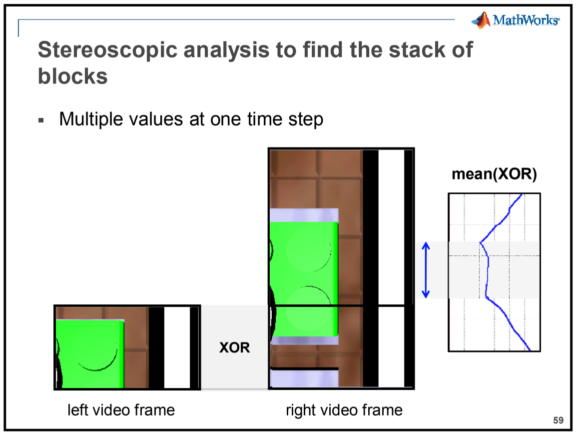

A Cyber-Physical System!
Justyna Zander and Pieter J. Mosterman, "Technical Engine for Democratizing Modeling, Simulation, and Prediction," in Winter Simulation Conference, December, 2012

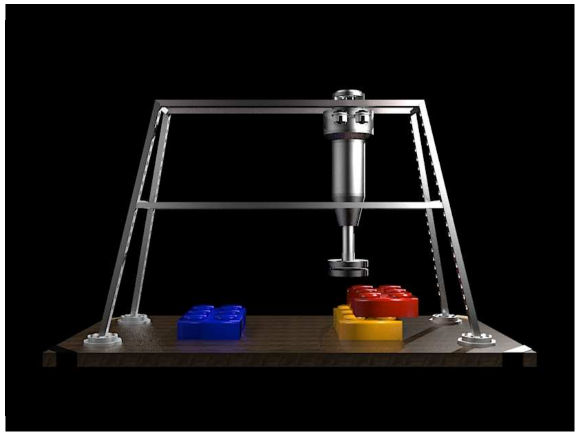

MathWorks
Local control laws
Green block
Should be on top
If it is on top, move one spot over and then move one spot over
If it is at bottom, move two spots over
Red block
Should be on bottom
If it is on top, move two spots over
If it is on bottom, move two spots over
Should have the highest priority
Green block
55

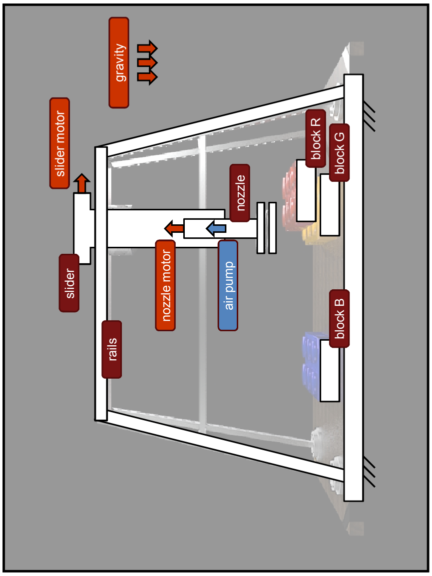

MathWorks
slider motor and pump actuator
nozzle motor actuator
physics
position sensor
slider cameras
take_snapshot
P: 5
slider processor
find_stack
P: 10
base processor
block G camera
block G
block R camera
block R
[R_ops]
[G_ops]
[R_status]
[G_status]
network
http://www.mathworks.com/matlabcentral/fileexchange/38515

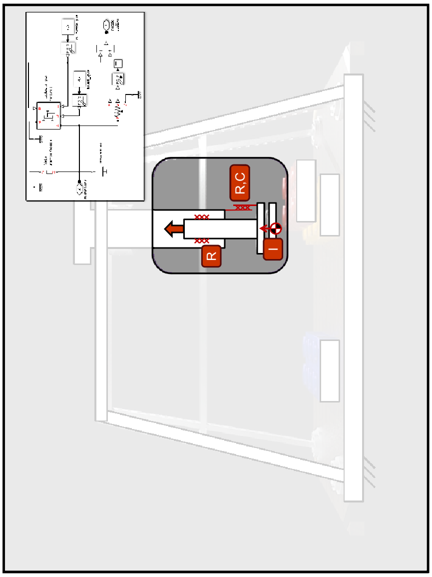

MathWorks
Stereoscopic analysis to find the stack of blocks
▪ Multiple values at one time step
left video frame
right video frame
57

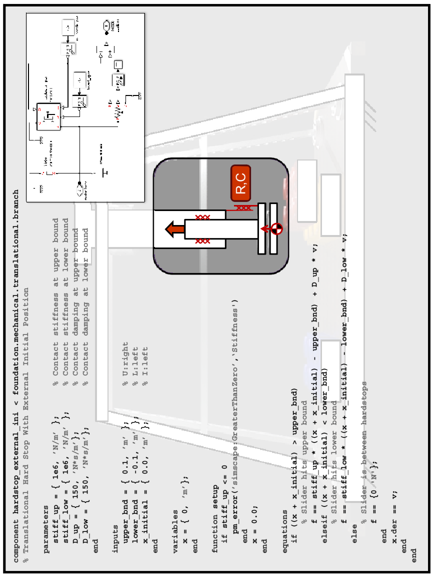

MathWorks
Stereoscopic analysis to find the stack of blocks
▪ Multiple values at one time step
XOR
left video frame
right video frame
58

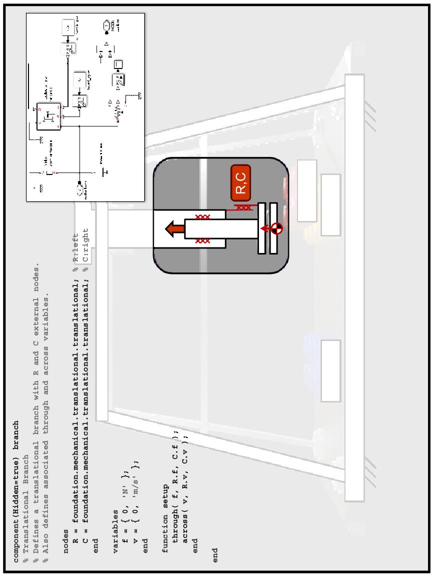

MathWorks
Stereoscopic analysis to find the stack of blocks
▪ Multiple values at one time step
mean(XOR)
XOR
left video frame
right video frame
59

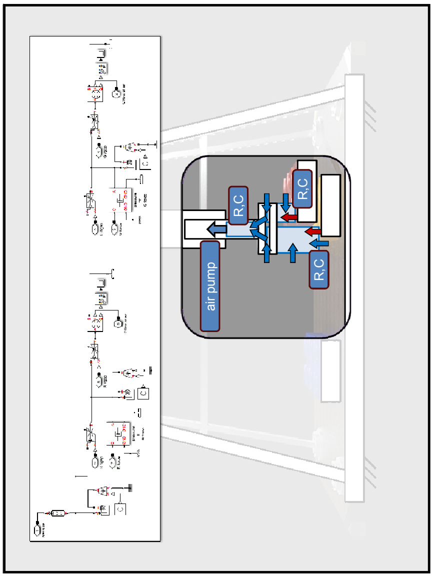


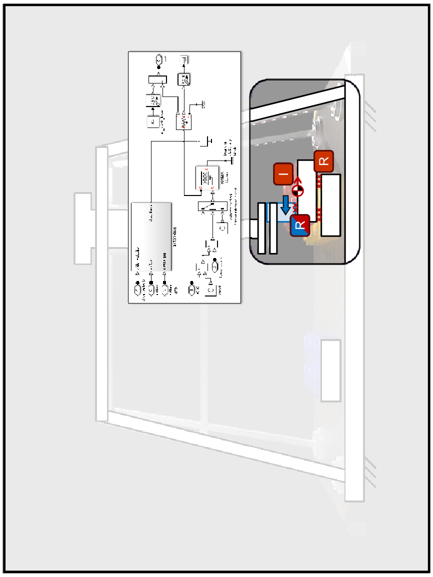

gravity
slider motor
slider
rails
nozzle motor
air pump
nozzle
block R
block G
block B

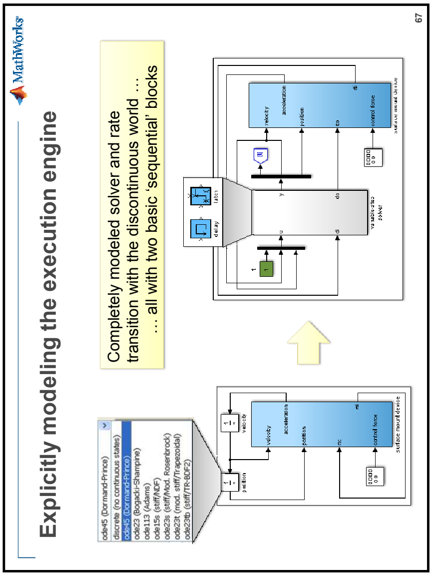

R,C
R
I

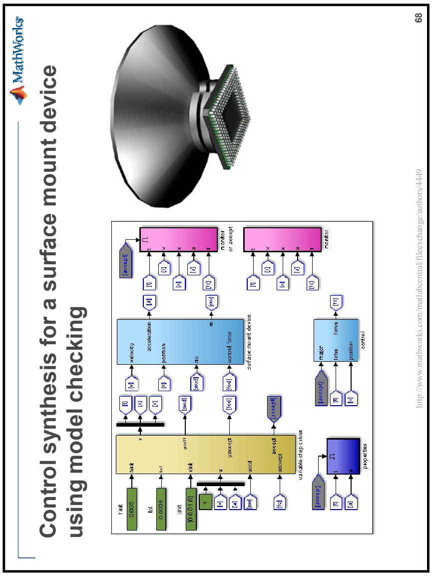

component hardstop_external_ini < foundation.mechanical.translational.branch
% Translational Hard Stop With External Initial Position
parameters
stiff_up = { 1e6, 'N/m' }; % Contact stiffness at upper bound
stiff_low = { 1e6, 'N/m' }; % Contact stiffness at lower bound
D_up = { 150, 'N*s/m'}; % Contact damping at upper bound
D_low = { 150, 'N*s/m'}; % Contact damping at lower bound
end
inputs
upper_bnd = { 0.1, 'm' }; % U:right
lower_bnd = { -0.1, 'm' }; % L:left
x_initial = { 0.0, 'm' }; % I:left
end
variables
x = { 0, 'm'};
end
function setup
if stiff_up <= 0
pm_error('simscape:GreaterThanZero','Stiffness')
end
x = 0.0;
end
equations
if ((x + x_initial) > upper_bnd)
% Slider hits upper bound
f == stiff_up * ((x + x_initial) - upper_bnd) + D_up * v;
elseif ((x + x_initial) < lower_bnd)
% Slider hits lower bound
f == stiff_low * ((x + x_initial) - lower_bnd) + D_low * v;
else
% Slider is between hardstops
f == {0 'N'};
end
x.der == v;
end
end
R,C


component(Hidden=true) branch
% Translational Branch
% Defines a translational branch with R and C external nodes.
% Also defines associated through and across variables.
nodes
R = foundation.mechanical.translational.translational; % R:left
C = foundation.mechanical.translational.translational; % C:right
end
variables
f = { 0, 'N' };
v = { 0, 'm/s' };
end
function setup
through(f, R.f, C.f);
across(v, R.v, C.v);
end
end
R,C


air pump
R,C
R,C
R,C

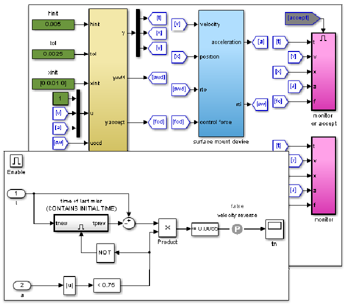

I
R
R

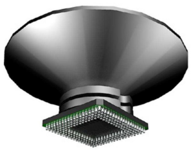

MathWorks
Explicitly modeling the execution engine
Completely modeled solver and rate transition with the discontinuous world …
… all with two basic 'sequential' blocks
ode45 (Dormand-Prince)
discrete (no continuous states)
ode45 (Dormand-Prince)
ode23 (Bogacki-Shampine)
ode113 (Adams)
ode15s (stiff/NDF)
ode23s (stiff/Mod. Rosenbrock)
ode23t (mod. stiff/Trapezoidal)
ode23tb (stiff/TR-BDF2)
67


MathWorks
Control synthesis for a surface mount device using model checking
http://www.mathworks.com/matlabcentral/fileexchange/authors/4449
68

MathWorks

Control synthesis for a surface mount device using model checking



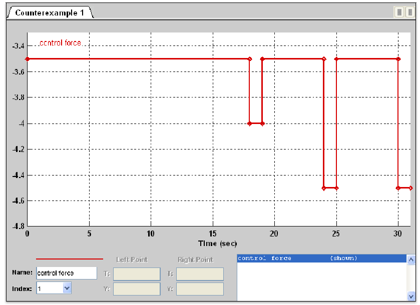


http://www.mathworks.com/matlabcentral/fileexchange/authors/4449

69

MathWorks

Model checking to generate a counterexample

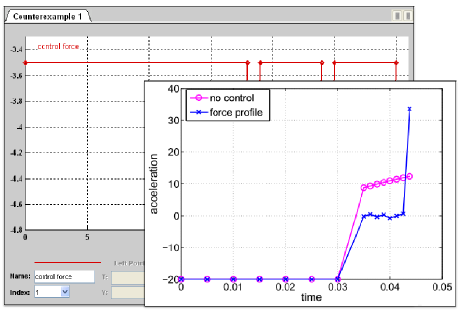


Pieter J. Mosterman, Justyna Zander, Grégoire Hamon, and Ben Denckla, **"A Computational Model of Time for Stiff Hybrid Systems Applied to Control Synthesis,"** in *Control Engineering Practice*, , vol. 20, nr. 1, pp. 2-13, January 2012

70

MathWorks

Model checking to generate a counterexample



Pieter J. Mosterman, Justyna Zander, Grégoire Hamon, and Ben Denckla, **"A Computational Model of Time for Stiff Hybrid Systems Applied to Control Synthesis,"** in *Control Engineering Practice*, , vol. 20, nr. 1, pp. 2-13, January 2012

71

MathWorks

Characteristics of the semantic domain

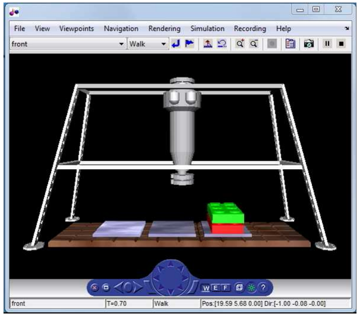
- Declarative
  - Purely functional (no side effects)
- Ordered evaluations
- Untimed
  - Time as explicit function, *t(e)*
  - Time is not strictly increasing
- Broadly applicable to dynamic systems
  - Differential equations, difference equations, discrete events

Pieter J. Mosterman, Justyna Zander, Grégoire Hamon, and Ben Denckla, **"Towards Computational Hybrid System Semantics for Time-Based Block Diagrams,"** in *3rd IFAC Conference on Analysis and Design of Hybrid Systems* (ADHS'09), A. Giua, C. Mahulea, M. Silva, and J. Zaytoon (eds.), pp. 376-385, Zaragoza, Spain, September 16-18, 2009, plenary paper.

72

MathWorks

Scenarios—emerging behavior



73

MathWorks

Agenda

- Cyber-physical systems
- Modeling cyber-physical systems
- Modeling approximations
- A solver model for control synthesis
- Conclusions

74

MathWorks

Conclusions

- Today's systems are open
  - Interact across various modalities
- Computational models include a variety of semantics
  - Many interacting approximations
- We should understand our computational methods
- Model solvers
  - A functional stream-based approach
  - Formalize computational semantics of the execution engine
- Exploit the abstraction
  - Computational methods for analysis, design, and synthesis
- Bring disciplines together
  - Engineering, Computer Science, Physics, Mathematics

75

MathWorks

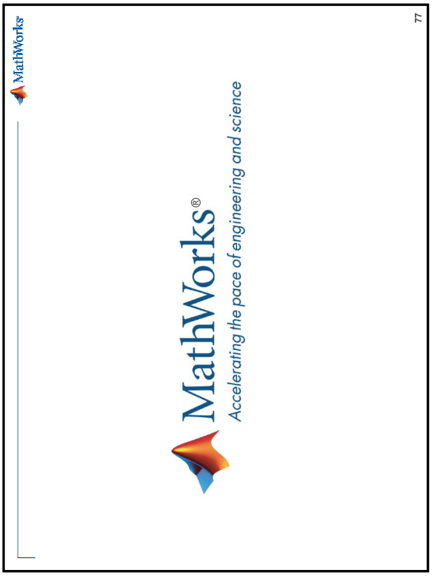
Acknowledgments

Justyna Zander
Harvard University
SimulatedWay, Berlin

Hans Vangheluwe
University of Antwerp
McGill University

Many thanks for their continuing collaboration!

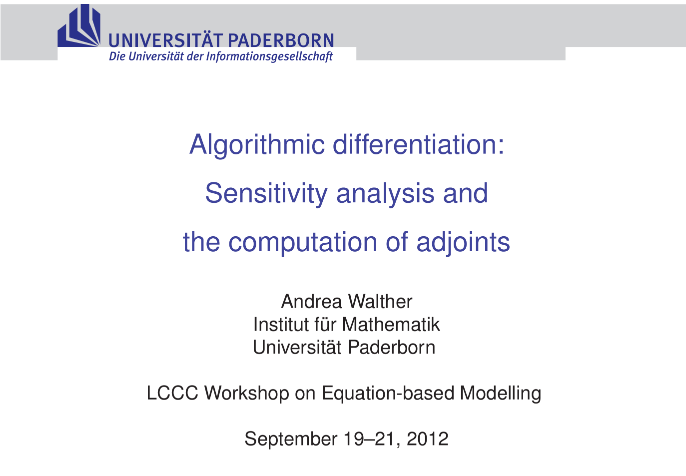
76

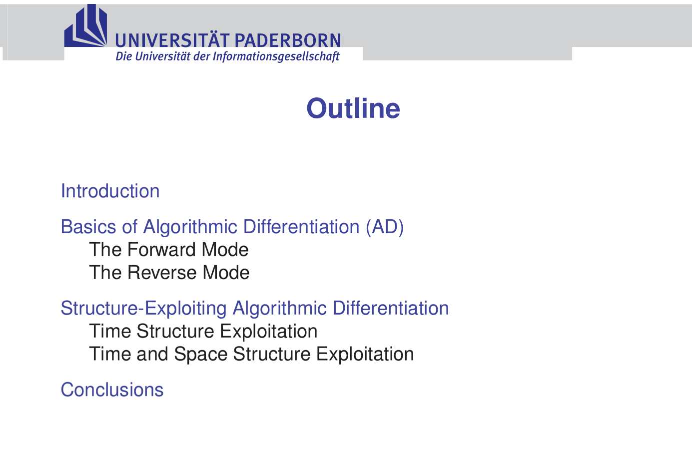

MathWorks
MathWorks®
Accelerating the pace of engineering and science
77

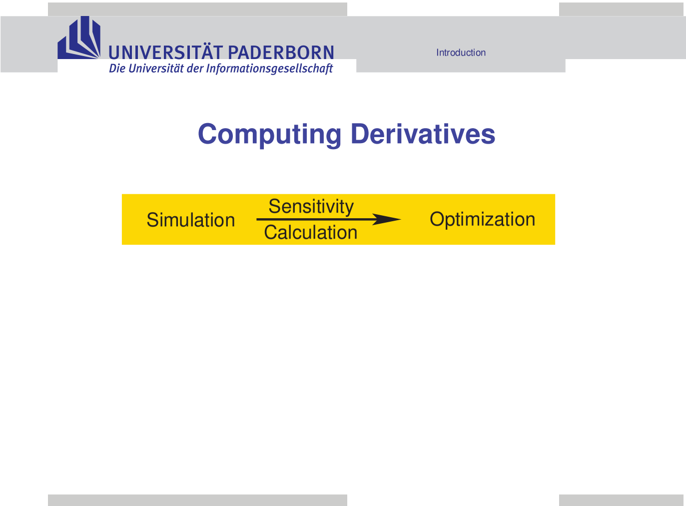
ALGORITHMIC DIFFERENTIATION: SENSITIVITY ANALYSIS AND THE COMPUTATION OF ADJOINTS

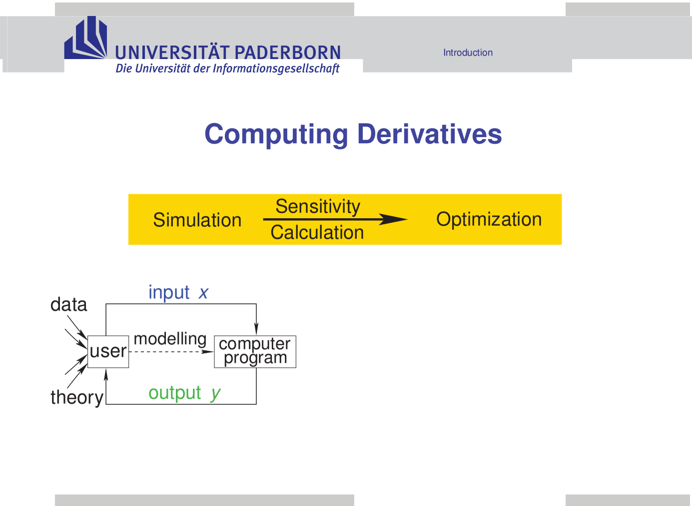
Andrea Walther, Institut fur Mathematik Universität Paderborn

Algorithmic differentiation: Sensitivity analysis and the computation of adjoints The provision of exact and consistent derivative information is important for numerous applications arising from optimisation purposes as for example optimal control problems. However, even the pure simulation of complex systems may require the computation of derivative information. Implicit integration methods are prominent examples for this case. The talk will present the technique of algorithmic (or automatic) differentiation (AD) to compute exact derivative information for function evaluations given as computer programs. This includes a short overview of the history of AD and a description of the main variants of AD, namely the forward mode to compute sensitivities and the reverse mode for the provision of adjoints. A discussion of complexity estimates follows yielding the important cheap gradient result. Then several aspects closely connected with the computation of sensitivity and adjoint information are emphasised. This covers also the structure exploitation in time and space. Some examples stemming optimal flow control problems illustrate the presented aspects.


UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Algorithmic differentiation:
Sensitivity analysis and
the computation of adjoints
Andrea Walther
Institut für Mathematik
Universität Paderborn
LCCC Workshop on Equation-based Modelling
September 19–21, 2012

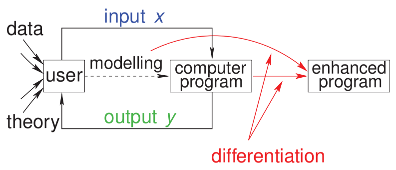

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Outline
Introduction
Basics of Algorithmic Differentiation (AD)
The Forward Mode
The Reverse Mode
Structure-Exploiting Algorithmic Differentiation
Time Structure Exploitation
Time and Space Structure Exploitation
Conclusions


UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction
Computing Derivatives
Simulation
Sensitivity
Calculation
Optimization


UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction
Computing Derivatives
Simulation
Sensitivity
Calculation
Optimization
input x
data
user
modelling
computer program
theory
output y

Computing Derivatives

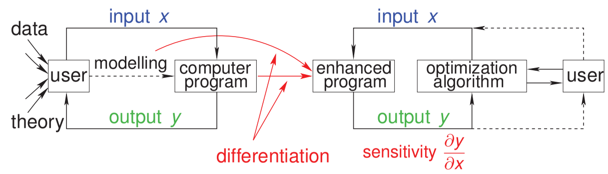
Simulation —Sensitivity Calculation→ Optimization



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction

Computing Derivatives

Simulation —Sensitivity Calculation→ Optimization



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction

Finite Differences

Idea: Taylor-expansion, $f : \mathbb{R} \to \mathbb{R}$ smooth then

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + h^2 f''(x)/2 + h^3 f'''(x)/6 + \ldots \\ \Rightarrow \quad f(x+h) &\approx f(x) + hf'(x) \\ \Rightarrow \quad Df(x) &= \frac{f(x+h) - f(x)}{h} \end{aligned}$$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction

Finite Differences

Idea: Taylor-expansion, $f : \mathbb{R} \to \mathbb{R}$ smooth then

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + h^2 f''(x)/2 + h^3 f'''(x)/6 + \ldots \\ \Rightarrow \quad f(x+h) &\approx f(x) + hf'(x) \\ \Rightarrow \quad Df(x) &= \frac{f(x+h) - f(x)}{h} \end{aligned}$$

- simple derivative calculation (only function evaluations!)
- inexact derivatives
- computation cost often too high

  $F : \mathbb{R}^n \to \mathbb{R} \quad \Rightarrow \quad \mathrm{OPS}(\nabla F(x)) \sim (n+1)\mathrm{OPS}(F(x))$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Introduction

Analytic Differentiation

- exact derivatives
  - $f(x) = \exp(\sin(x^2)) \Rightarrow$
    $f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Introduction

Analytic Differentiation

- exact derivatives
  - $f(x) = \exp(\sin(x^2)) \Rightarrow$
    $f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$
  - $\min J(x,u)$ such that $x' = f(x,u) + \text{IC}$
    reduced formulation: $J(x,u) \to \widehat{J}(u)$
    $\widehat{J}'(u)$ based on symbolic adjoint $\lambda' = -f_x(x,u)^\top \lambda + \text{TC}$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

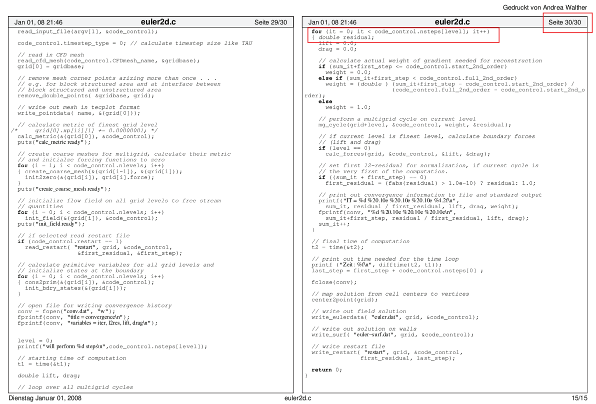
Introduction

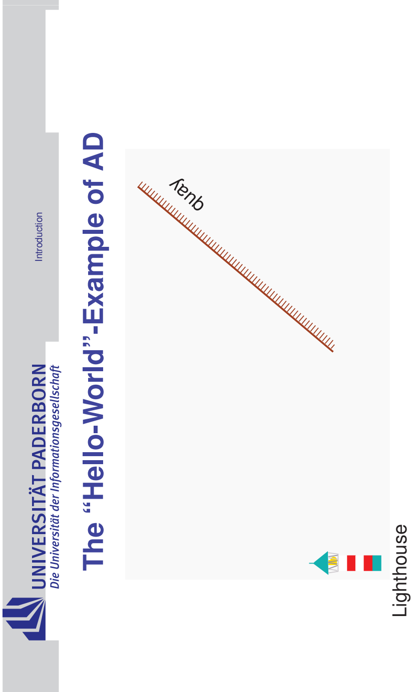
Analytic Differentiation

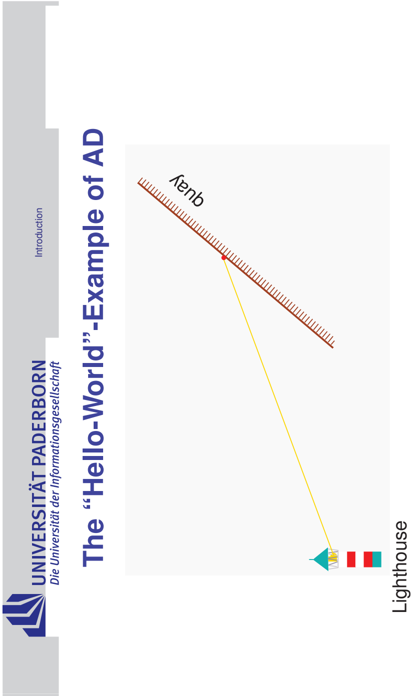
- exact derivatives
  - $f(x) = \exp(\sin(x^2)) \Rightarrow$
    $f'(x) = \exp(\sin(x^2)) * \cos(x^2) * 2x$
  - $\min J(x,u)$ such that $x' = f(x,u) + \text{IC}$
    reduced formulation: $J(x,u) \to \widehat{J}(u)$
    $\widehat{J}'(u)$ based on symbolic adjoint $\lambda' = -f_x(x,u)^\top \lambda + \text{TC}$
- cost (common subexpression, implementation)
- legacy code with large number of lines $\Rightarrow$
  closed form expression not available
- consistent derivative information?!

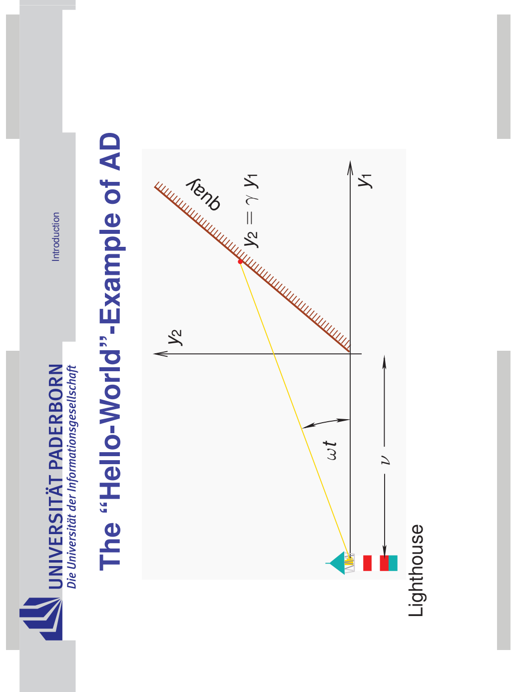
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

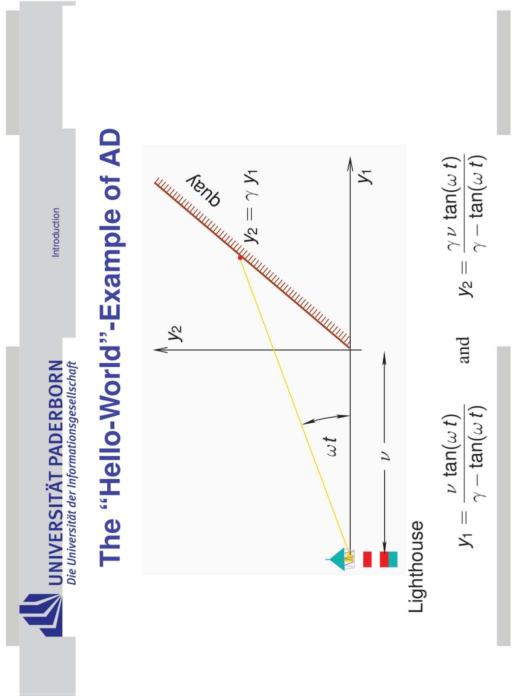
Introduction

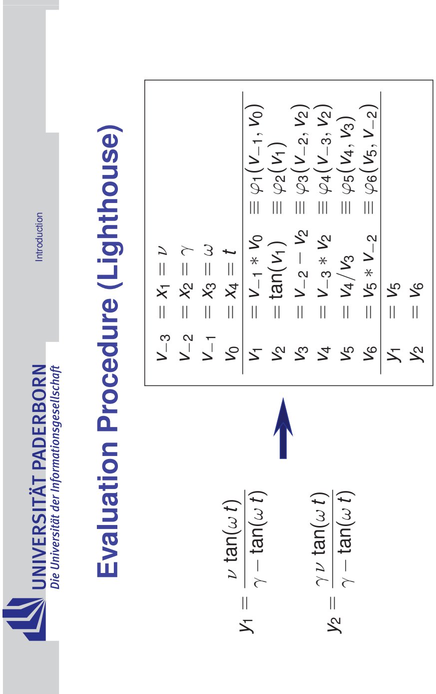



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction
The "Hello-World"-Example of AD
quay
Lighthouse

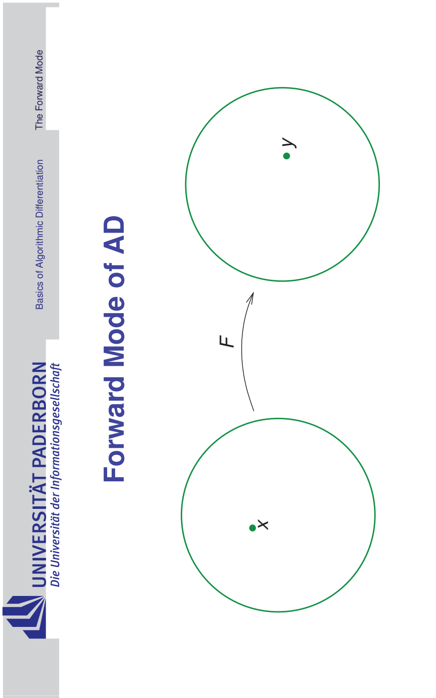

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction
The "Hello-World"-Example of AD
quay
Lighthouse


UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction
The "Hello-World"-Example of AD
quay
y_2
$y_2 = \gamma\, y_1$
y_1
ωt
ν
Lighthouse

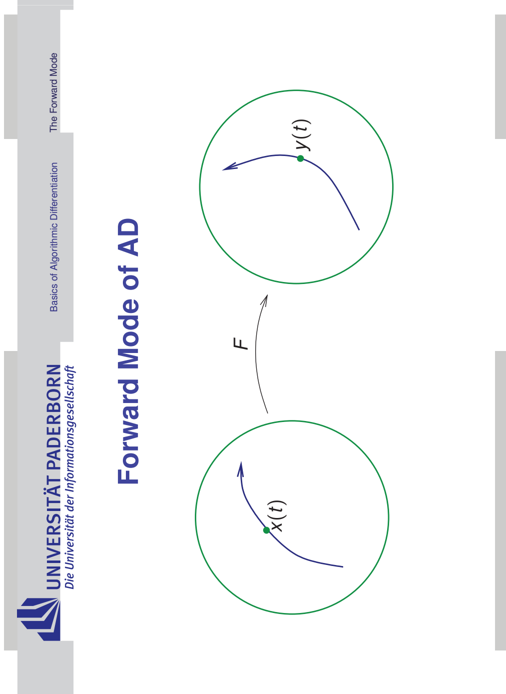

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction
The "Hello-World"-Example of AD
quay
y_2
$y_2 = \gamma\, y_1$
y_1
ωt
ν
Lighthouse
$y_1 = \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)}$ and $y_2 = \frac{\gamma\, \nu \tan(\omega t)}{\gamma - \tan(\omega t)}$



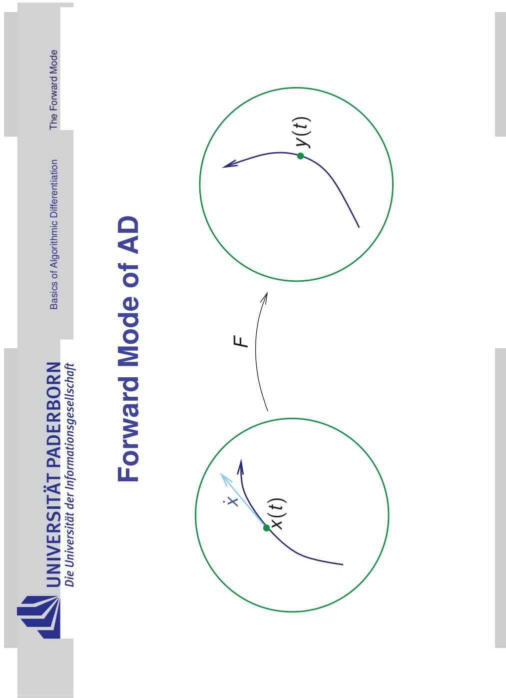
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Introduction
Evaluation Procedure (Lighthouse)
$y_1 = \frac{\nu \tan(\omega t)}{\gamma - \tan(\omega t)}$
$y_2 = \frac{\gamma \nu \tan(\omega t)}{\gamma - \tan(\omega t)}$
$v_{-3} = x_1 = \nu$
$v_{-2} = x_2 = \gamma$
$v_{-1} = x_3 = \omega$
$v_0 = x_4 = t$
$v_1 = v_{-1} * v_0 \equiv \varphi_1(v_{-1}, v_0)$
$v_2 = \tan(v_1) \equiv \varphi_2(v_1)$
$v_3 = v_{-2} - v_2 \equiv \varphi_3(v_{-2}, v_2)$
$v_4 = v_{-3} * v_2 \equiv \varphi_4(v_{-3}, v_2)$
$v_5 = v_4 / v_3 \equiv \varphi_5(v_4, v_3)$
$v_6 = v_5 * v_{-2} \equiv \varphi_6(v_5, v_{-2})$
$y_1 = v_5$
$y_2 = v_6$



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation
The Forward Mode
Forward Mode of AD
x
F
y



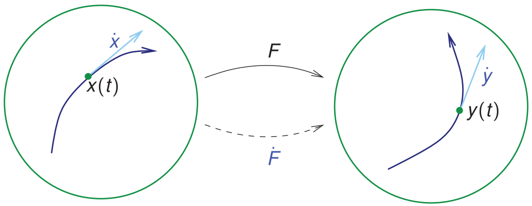
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation
The Forward Mode
Forward Mode of AD
x(t)
F
y(t)



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation
The Forward Mode
Forward Mode of AD
ẋ
x(t)
F
y(t)

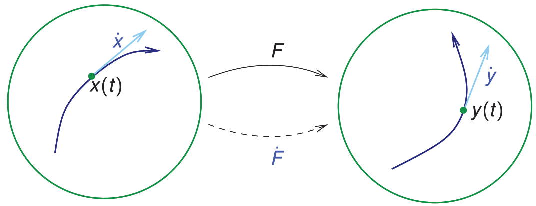
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Forward Mode

Forward Mode of AD



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Forward Mode

Forward Mode of AD



$$\dot{y}(t) \;=\; \frac{\partial}{\partial t}F(x(t)) \;=\; F'(x(t))\,\dot{x}(t) \equiv \dot{F}(x,\dot{x})$$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Forward Mode

Forward AD (Lighthouse Example)

v_{-3}	$=$	$x_1 = \nu$	$\dot{v}_{-3}$	$\equiv$	$\dot{x}_1$
v_{-2}	$=$	$x_2 = \gamma$	$\dot{v}_{-2}$	$\equiv$	$\dot{x}_2$
v_{-1}	$=$	$x_3 = \omega$	$\dot{v}_{-1}$	$\equiv$	$\dot{x}_3$
v_0	$=$	$x_4 = t$	$\dot{v}_0$	$\equiv$	$\dot{x}_4$
v_1	$=$	$v_{-1} * v_0$			
v_2	$=$	$\tan(v_1)$			
v_3	$=$	$v_{-2} - v_2$			
v_4	$=$	$v_{-3} * v_2$			
v_5	$=$	v_4 / v_3			
v_6	$=$	v_5			
v_7	$=$	$v_5 * v_{-2}$			
y_1	$=$	v_6			
y_2	$=$	v_7			

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Forward Mode

Forward AD (Lighthouse Example)

v_{-3}	$=$	$x_1 = \nu$	$\dot{v}_{-3}$	$\equiv$	$\dot{x}_1$
v_{-2}	$=$	$x_2 = \gamma$	$\dot{v}_{-2}$	$\equiv$	$\dot{x}_2$
v_{-1}	$=$	$x_3 = \omega$	$\dot{v}_{-1}$	$\equiv$	$\dot{x}_3$
v_0	$=$	$x_4 = t$	$\dot{v}_0$	$\equiv$	$\dot{x}_4$
v_1	$=$	$v_{-1} * v_0$	$\dot{v}_1$	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	$\dot{v}_2$	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$	$\dot{v}_3$	$=$	$\dot{v}_{-2} - \dot{v}_2$
v_4	$=$	$v_{-3} * v_2$	$\dot{v}_4$	$=$	$\dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2$
v_5	$=$	v_4 / v_3	$\dot{v}_5$	$=$	$(\dot{v}_4 - \dot{v}_3 * v_5) * (1/v_3)$
v_6	$=$	v_5	$\dot{v}_6$	$=$	$\dot{v}_5$
v_7	$=$	$v_5 * v_{-2}$	$\dot{v}_7$	$=$	$\dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2}$
y_1	$=$	v_6			
y_2	$=$	v_7			

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Forward Mode

Forward AD (Lighthouse Example)

v_{-3}	$=$	$x_1 = \nu$	$\dot{v}_{-3}$	$\equiv$	$\dot{x}_1$
v_{-2}	$=$	$x_2 = \gamma$	$\dot{v}_{-2}$	$\equiv$	$\dot{x}_2$
v_{-1}	$=$	$x_3 = \omega$	$\dot{v}_{-1}$	$\equiv$	$\dot{x}_3$
v_0	$=$	$x_4 = t$	$\dot{v}_0$	$\equiv$	$\dot{x}_4$
v_1	$=$	$v_{-1} * v_0$	$\dot{v}_1$	$=$	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	$=$	$\tan(v_1)$	$\dot{v}_2$	$=$	$\dot{v}_1 / \cos(v_1)^2$
v_3	$=$	$v_{-2} - v_2$	$\dot{v}_3$	$=$	$\dot{v}_{-2} - \dot{v}_2$
v_4	$=$	$v_{-3} * v_2$	$\dot{v}_4$	$=$	$\dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2$
v_5	$=$	v_4 / v_3	$\dot{v}_5$	$=$	$(\dot{v}_4 - \dot{v}_3 * v_5) * (1/v_3)$
v_6	$=$	v_5	$\dot{v}_6$	$=$	$\dot{v}_5$
v_7	$=$	$v_5 * v_{-2}$	$\dot{v}_7$	$=$	$\dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2}$
y_1	$=$	v_6	$\dot{y}_1$	$=$	$\dot{v}_6$
y_2	$=$	v_7	$\dot{y}_2$	$=$	$\dot{v}_7$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Forward Mode

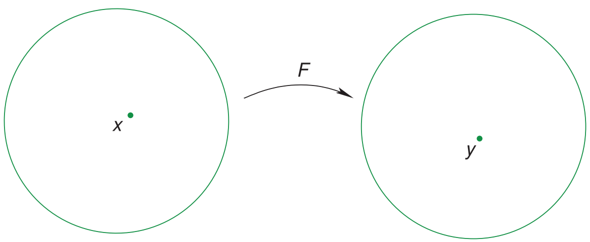
... and the real code

```
void d1_f(double* x, double* d1_x, double* y, double* d1_y)
//$ad indep x d1_x
//$ad dep y d1_y
 {
   double v[2];              double d1_v[2];
   double w1_0 = 0;          double d1_w1_0 = 0;
   ...
   double w1_5 = 0;          double d1_w1_5 = 0;

   d1_w1_0 = d1_x[2];        w1_0 = x[2];
   d1_w1_1 = d1_x[3];        w1_1 = x[3];
   d1_w1_2 = w1_1*d1_w1_0 + w1_0*d1_w1_1;
   w1_2 = w1_0*w1_1;
   d1_w1_3 = 1/(cos(w1_2)*cos(w1_2)) * d1_w1_2;
   w1_3 = tan(w1_2);
   ...
```

using dcc 1.0 (U. Naumann, RWTH Aachen)

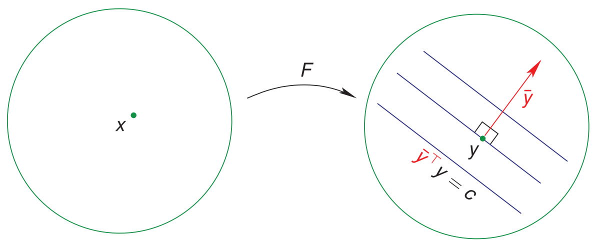
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Reverse Mode

Reverse Mode of AD

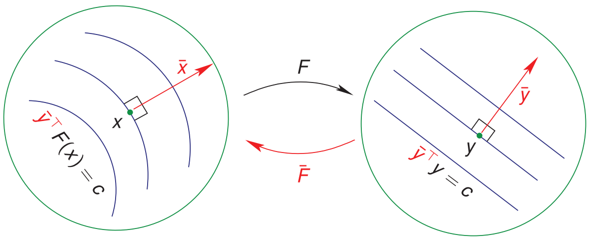


UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation — The Reverse Mode

Reverse Mode of AD

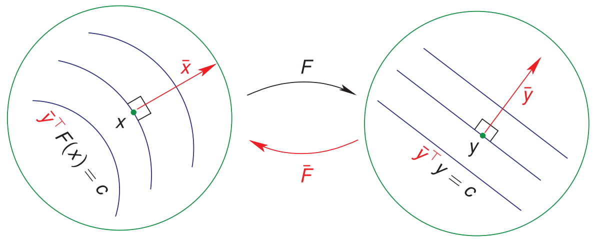


Reverse Mode of AD



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation
The Reverse Mode

Reverse Mode of AD



$$\bar{x}^\top \equiv \bar{y}^\top F'(x) = \nabla_x \langle \bar{y}^\top F(x) \rangle \equiv \bar{F}(x, \bar{y})$$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation
The Reverse Mode

Reverse Mode (Lighthouse)

$v_{-3} = x_1; \quad v_{-2} = x_2; \quad v_{-1} = x_3; \quad v_0 = x_4;$

$v_1 = v_{-1} * v_0;$

$v_2 = \tan(v_1);$

$v_3 = v_{-2} - v_2;$

$v_4 = v_{-3} * v_2;$

$v_5 = v_4 / v_3;$

$v_6 = v_5 * v_{-2};$

$y_1 = v_5; \quad y_2 = v_6;$

$\bar{v}_5 = \bar{y}_1; \quad \bar{v}_6 = \bar{y}_2;$

$\bar{v}_5 \mathrel{+}= \bar{v}_6 * v_{-2}; \quad \bar{v}_{-2} \mathrel{+}= \bar{v}_6 * v_5;$

$\bar{v}_4 \mathrel{+}= \bar{v}_5 / v_3; \quad \bar{v}_3 \mathrel{-}= \bar{v}_5 * v_5 / v_3;$

$\bar{v}_{-3} \mathrel{+}= \bar{v}_4 * v_2; \quad \bar{v}_2 \mathrel{+}= \bar{v}_4 * v_{-3};$

$\bar{v}_{-2} \mathrel{+}= \bar{v}_3; \; \bar{v}_2 \mathrel{-}= \bar{v}_3;$

$\bar{v}_1 \mathrel{+}= \bar{v}_2 / \cos^2(v_1);$

$\bar{v}_{-1} \mathrel{+}= \bar{v}_1 * v_0; \; \bar{v}_0 \mathrel{+}= \bar{v}_1 * v_{-1};$

$\bar{x}_4 = \bar{v}_0; \quad \bar{x}_3 = \bar{v}_{-1}; \quad \bar{x}_2 = \bar{v}_{-2}; \quad \bar{x}_1 = \bar{v}_{-3};$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation
The Reverse Mode

... and the real code generated by dcc 1.0

```
void b1_f(int& bmode_1, double* x, double* b1_x, double* y, double* b1_y)
//$ad indep x b1_x b1_y
//$ad dep y b1_x
 { double v[2];          double b1_v[2];
   double w1_0 = 0;      double b1_w1_0 = 0;       . . .
   double w1_5 = 0;      double b1_w1_5 = 0;
   int save_cs_c = 0;    save_cs_c = cs_c;
   if (bmode_1==1) { // augmented forward section
     cs[cs_c] = 0;         cs_c = cs_c+1;
     fds[fds_c] = v[0];    fds_c = fds_c+1;    v[0] = tan(x[2]*x[3]);
     . . .
     fds[fds_c] = y[1];    fds_c = fds_c+1;    y[1] = x[1]*y[0];
     while (cs_c>save_cs_c) {    // reverse section
       cs_c = cs_c-1;
       if (cs[cs_c]==0) {
         fds_c = fds_c-1;          y[1] = fds[fds_c];
         w1_0 = x[1];              w1_1 = y[0];        w1_2 = w1_0*w1_1;
         b1_w1_2 = b1_y[1];   b1_y[1] = 0; // adjoint assignment
         b1_w1_0 = w1_1*b1_w1_2;    b1_w1_1 = w1_0*b1_w1_2;
         b1_y[0] = b1_y[0]+b1_w1_1;   b1_x[1] = b1_x[1]+b1_w1_0;       . . .
```

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- ...

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- ...

C/C++: (mainly operator overloading)

- ADOL-C (Univ. Paderborn)
- CppAD (Univ. Washington, USA)
- ...

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- ...

C/C++: (mainly operator overloading)

- ADOL-C (Univ. Paderborn)
- CppAD (Univ. Washington, USA)
- ...

Matlab: Adimat, MAD, ...
Modelica: ADModelica by Atya Elsheikh und Wolfgang Wiechert (!)

AD Tools

Fortran 77 (90): (mainly source transformation)

- Tapenade (INRIA, F)
- AD in the compiler (NAG, RWTH Aachen, Univ. Hertfordshire)
- ...

C/C++: (mainly operator overloading)

- ADOL-C (Univ. Paderborn)
- CppAD (Univ. Washington, USA)
- ...

Matlab: Adimat, MAD, ...
Modelica: ADModelica by Atya Elsheikh und Wolfgang Wiechert (!)

see www.autodiff.org, (Griewank, Walther 2008), (Naumann 2012)
for more tools and literature

Conclusions: Basic AD

- Evaluation of derivatives with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\mathrm{OPS}(F'(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [2, 5/2]$
  Reverse mode: $\mathrm{OPS}(\bar{y}^\top F'(x)) \leq c\,\mathrm{OPS}(F), \quad c \in [3, 4]$
  $\mathrm{MEM}(\bar{y}^\top F'(x)) \sim \mathrm{OPS}(F)$,

→ Gradients are cheap ~ Function Costs!!

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation | The Reverse Mode

Conclusions: Basic AD

- Evaluation of derivatives with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\mathrm{OPS}(F'(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [2, 5/2]$
  Reverse mode: $\mathrm{OPS}(\bar{y}^\top F'(x)) \leq c\,\mathrm{OPS}(F), \quad c \in [3, 4]$
  $\mathrm{MEM}(\bar{y}^\top F'(x)) \sim \mathrm{OPS}(F)$,

→ Gradients are cheap ~ Function Costs!!

- Combination: $\mathrm{OPS}(\bar{y}^\top F''(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [7, 10]$
- Cost of higher derivatives grows quadratically in the degree
- Nondifferentiability only on meager set
- Full Jacobians/Hessians often not needed or sparse

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation | The Reverse Mode

Conclusions: Basic AD

- Evaluation of derivatives with working accuracy (Griewank, Kulshreshtha, Walther 2012)
- Forward mode: $\mathrm{OPS}(F'(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [2, 5/2]$
  Reverse mode: $\mathrm{OPS}(\bar{y}^\top F'(x)) \leq c\,\mathrm{OPS}(F), \quad c \in [3, 4]$
  $\mathrm{MEM}(\bar{y}^\top F'(x)) \sim \mathrm{OPS}(F)$,

→ Gradients are cheap ~ Function Costs!!

- Combination: $\mathrm{OPS}(\bar{y}^\top F''(x)\dot{x}) \leq c\,\mathrm{OPS}(F), \quad c \in [7, 10]$
- Cost of higher derivatives grows quadratically in the degree
- Nondifferentiability only on meager set
- Full Jacobians/Hessians often not needed or sparse

Questions: Structure Exploitation!!
Time-stepping, sparsity, fixed point iteration, ...

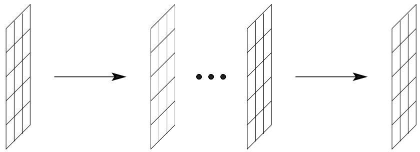
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Basics of Algorithmic Differentiation | The Reverse Mode

Automatic Differentiation by Overloading in C++

- **ADOL-C version 2.3**
- available at COIN-OR since May 2009
- interface to ColPack (Purdue University) and Ipopt (COIN-OR)
- recent developments
  - improved computation of sparsity pattern for Hessians
  - handling of MPI-parallel codes
  - handling of GPU-parallel codes
- future plans
  - generalized derivatives for nonsmooth functions
  - ...

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

Calculating Adjoints



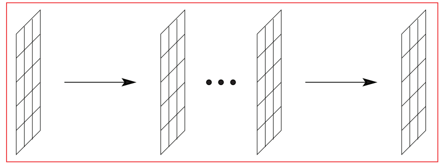
Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1?$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

Calculating Adjoints



Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

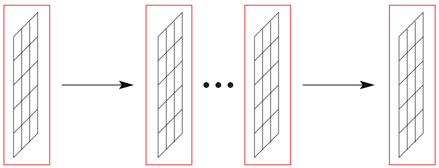
Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1?$

"Black-Box"-approach, e.g. using AD

Memory requirement?? Computing time ??

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

Calculating Adjoints



Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1?$

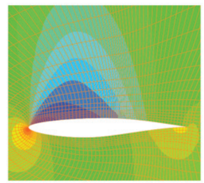
Time Structure Exploitation

Memory requirement?? Computing time ?? Adjoint ??

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

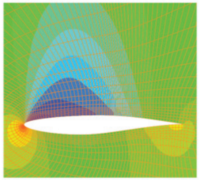
Pseudo Time-dependent Problems

- Example:
  Shape Optimization
  in Aerodynamics
- Target: Minimize drag



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

Pseudo Time-dependent Problems

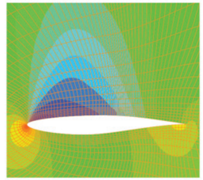


- Example:
  Shape Optimization
  in Aerodynamics
- Target: Minimize drag

Approaches:
- Exploitation of fixed point structure
  $\Rightarrow$ reverse accumulation of gradient (Christianson 1991)
  $\Rightarrow$ TIME(gradient)/TIME(target function) < 9
  (Gauger, Walther, Özkaya, Moldenhauer 2012)

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

Pseudo Time-dependent Problems

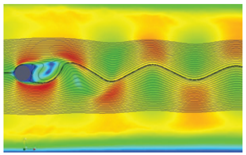


- Example:
  Shape Optimization
  in Aerodynamics
- Target: Minimize drag

Approaches:
- Exploitation of fixed point structure
  $\Rightarrow$ reverse accumulation of gradient (Christianson 1991)
  $\Rightarrow$ TIME(gradient)/TIME(target function) < 9
  (Gauger, Walther, Özkaya, Moldenhauer 2012)
- One-Shot Optimization
  $\Rightarrow$ again adjoint of only one time step required
  N. Gauger, A. Griewank, E. Özkaya

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

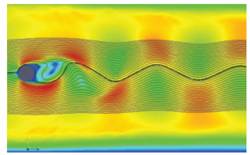
Real Time-dependent Problems



- Example:
  Transient flows
- Target: Minimize drag/turbulence

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation Structure in Time

Real Time-dependent Problems



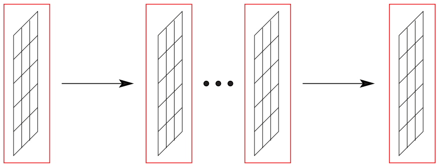
- Example:
  Transient flows
- Target: Minimize drag/turbulence

Approaches: Checkpointing in all variations, adjoint of one time step
- PDE-based optimization: Windowing
  Berggren, Meidner, Vexler, . . .
- Binomial Checkpointing
  Griewank, Walther, Sternberg, Stumm, Moin, . . .
- in general for AD: subroutine oriented checkpointing
  OpenAD, Tapenade

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Structure-Exploiting Algorithmic Differentiation Structure in Time and Space

Calculating Adjoints II



Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1?$

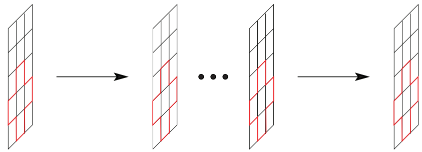
Time Structure Exploitation

Memory requirement?? Computing time ?? Adjoint ??

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Structure-Exploiting Algorithmic Differentiation Structure in Time and Space

Calculating Adjoints II



Integration of forward solution:

$$y_{i+1} = F_i(y_i, u_i), \qquad i = 1, \ldots, l$$

Integration of adjoint $\bar{y}_{i-1} = \bar{F}_i(\bar{y}_i, \bar{u}_i, y_i),\ i = l, \ldots, 1?$

Time and Space Structure Exploitation

Memory requirement?? Computing time ?? Adjoint ??

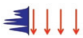
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Structure-Exploiting Algorithmic Differentiation Structure in Time and Space

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



$\longleftarrow$ adaptable light puls $E(t)$

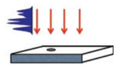
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Structure-Exploiting Algorithmic Differentiation Structure in Time and Space

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



$\longleftarrow$ adaptable light puls $E(t)$

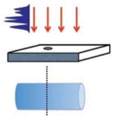
$\longleftarrow$ metal aperture with air hole

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation — Structure in Time and Space

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



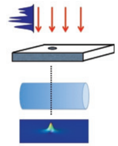
⟵ adaptable light puls $E(t)$
⟵ metal aperture with air hole
⟵ quantum wire

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation — Structure in Time and Space

Optimisation for Nanooptics

Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



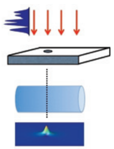
⟵ adaptable light puls $E(t)$
⟵ metal aperture with air hole
⟵ quantum wire
⟵ electron distribution?

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation — Structure in Time and Space

Optimisation for Nanooptics

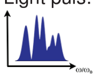
Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



⟵ adaptable light puls $E(t)$
⟵ metal aperture with air hole
⟵ quantum wire
⟵ electron distribution?

Light puls:



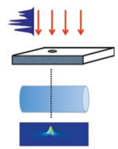
with $E(t) = \sum A_i \exp\left(-\left(\frac{t-t_i}{\Delta t_i}\right)^2\right)\cos(\omega_i t + \phi_i)$

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Structure-Exploiting Algorithmic Differentiation — Structure in Time and Space

Optimisation for Nanooptics

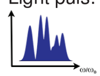
Cooperation with T. Meier, M. Reichelt, Dep. Physik, Uni Paderborn

Generic configuration:



⟵ adaptable light puls $E(t)$
⟵ metal aperture with air hole
⟵ quantum wire
⟵ electron distribution?

Light puls:



with $E(t) = \sum A_i \exp\left(-\left(\frac{t-t_i}{\Delta t_i}\right)^2\right)\cos(\omega_i t + \phi_i)$

Parameter: A_i, ϕ_i $\Rightarrow 60!$

UNIVERSITÄT PADERBORN *Die Universität der Informationsgesellschaft* — Structure-Exploiting Algorithmic Differentiation — Structure in Time and Space

Nanooptics: Optimisation

So far: Genetic algorithms

Now: L-BFGS and efficient gradient computation

- AD coupled with hand-coded adjoints
- Checkpointing (160 000 time steps!!)

$\Rightarrow$ TIME(gradient)/TIME(target function) $<$ 7 despite of checkpointing!

UNIVERSITÄT PADERBORN *Die Universität der Informationsgesellschaft* — Structure-Exploiting Algorithmic Differentiation — Structure in Time and Space

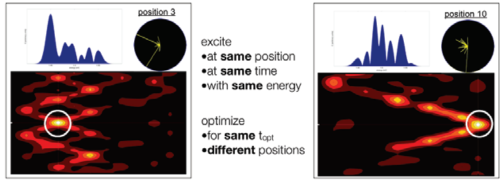
Nanooptics: Optimisation

So far: Genetic algorithms

Now: L-BFGS and efficient gradient computation

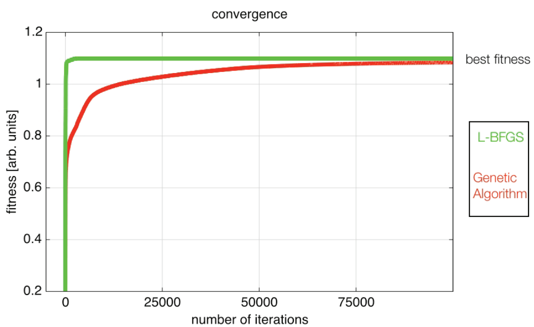
- AD coupled with hand-coded adjoints
- Checkpointing (160 000 time steps!!)

$\Rightarrow$ TIME(gradient)/TIME(target function) $<$ 7 despite of checkpointing!



UNIVERSITÄT PADERBORN *Die Universität der Informationsgesellschaft* — Structure-Exploiting Algorithmic Differentiation — Structure in Time and Space

Nanooptics: Comparison



(Walther, Reichelt, Meier 2011)

UNIVERSITÄT PADERBORN *Die Universität der Informationsgesellschaft* — Conclusions

Conclusions

- Basics of Algorithmic Differentiation
  - Efficient evaluation of derivatives with working accuracy
  - Discrete Analogons of sensitivity and adjoint equation
  - Theory for basic modes complete, advanced AD?

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Conclusions

Conclusions

- Basics of Algorithmic Differentiation
  - Efficient evaluation of derivatives with working accuracy
  - Discrete Analogons of sensitivity and adjoint equation
  - Theory for basic modes complete, advanced AD?
- Structure exploitation indispensable

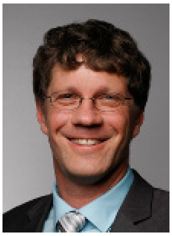
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Conclusions

Conclusions

- Basics of Algorithmic Differentiation
  - Efficient evaluation of derivatives with working accuracy
  - Discrete Analogons of sensitivity and adjoint equation
  - Theory for basic modes complete, advanced AD?
- Structure exploitation indispensable
- Consistent adjoint information? Efficient implementation?

  Suitable combination of continuous and discrete approach!



FUNCTIONAL DEVELOPMENT WITH MODELICA

Stefan-Alexander Schneider, Schneider System Consulting

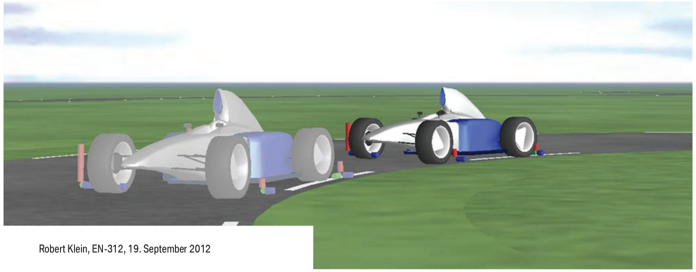
In the early phase of the product development, it is crucial to quickly and accurately evaluate a systems overall performance in order to fully define and optimize viable system and functional architectures. The presentation explains the development steps for an embedded controller. Typically, the behavior of a dynamic system (plant and controller) is in general to complex to treat by theory or formulas. Several simulation methods has established for analyzing such systems.

The presented virtual integration method allows to model and simulate the entire system, and thus the validation of the design decisions in an early phase of the development. This approach is conducted on a model in equation based languages to gain knowledge about the (intended) real system behabior. Such an abstraction typically allows to focus on the main properties and their effects of the studied multi-domain system.

The new approach of virtual integration is demonstrated for the development of a control algorithm for an embedded controller. The entire system – both the plant and the control components – is designed with the modeling language Modelica. All necessary activities are presented for the role of the function developer and explained for the example traffic light controller for a simple intersection.

The virtual integration method usualy combines components that require specific domain solvers for mechanical, electrical, etc. components, and, consequently, is based on the co-simulations, described in [1, 4]. There is a rather hugh literature on the Vee-Model and systems engineering, see e.g. [5, 3, 2]. For more general introduction see, e.g., [6, 7].

[1] Stefan-Alexander Schneider Andreas Maier. Grundlagen, Methoden und Anwendungen in Modellbildung und Simulation. Tagungsband ASIM 2011, 2011.
[2] R. Haberfellner, Olivier L. de Weck, E. Fricke, and S. Vössner. Systems Engineering – Grundlagen und Anwendungen. Orell Füssli Verlag, Zurich, 12th edition edition, January 2012. ISBN 978-3-85743-998-8.
[3] Richard Harwell. Systems Engineering, A Way of Thinking, A Way of Doing Business, Enabling Organized Transition from Need to Product, 1997. [Online; August 1997].
[4] H. Palm Stefan-Alexander Schneider, B. Schick. Virtualization, Integration and Simulation in the Context of Vehicle Systems Engineering. In Embedded World 2012 Exhibition & Conference Proceedings. Weka Fachmedien, 2012.
[5] Tim Weilkiens. Die Rolle des Systems Engineerings.
[6] Wikipedia. Systems Engineering – Wikipedia, the free encyclopedia, 2012. [Online; Status 13 May 2012].
[7] Wikipedia. V-Modell – Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 29. März 2012].



Robert Klein, EN-312, 19. September 2012

VIRTUAL SYSTEM PROTOTYPING.

RESULTS OF THE MASTER THESIS ROBERT KLEIN
DR. STEFAN-ALEXANDER SCHNEIDER, BMW GROUP AND
PROF. DR. HERBERT PALM, HOCHSCHULE MÜNCHEN.

BMW GROUP

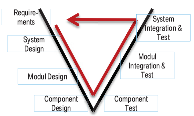
AGENDA.

- Problem Definition
- Virtual System Prototyping
  - Virtual System Design
  - Virtual System Integration
- Test of Numerical Stability
- Design Evaluation

PROBLEM DEFINITON.

As Is

Late system level response



→ The later failures are found, the more expensive they are!

Problem

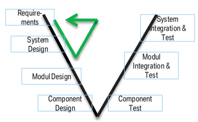
How to achieve an earlier system level response?

Solution

Virtual System Prototyping

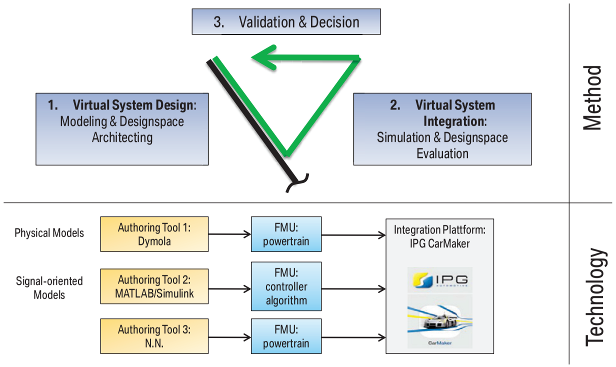
To Be

Early system level response

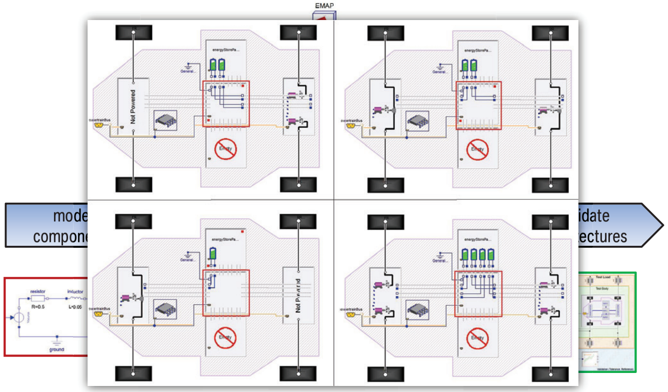


→ Early evaluation of system reduces the risk of missing the „best solution"!

VIRTUAL SYSTEM PROTOTYPING.

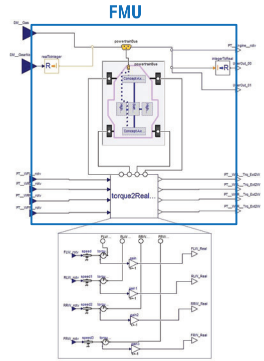


VIRTUAL SYSTEM DESIGN.



VIRTUAL SYSTEM INTEGRATION.

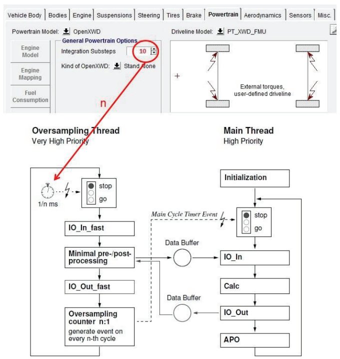
1. Definition of FMU with signal-oriented, CarMaker-specific interfaces
2. Converting physical interfaces into signal-oriented interfaces with modelica sensor component
3. Synchronizing axes with modelica speed component



VIRTUAL SYSTEM INTEGRATION.

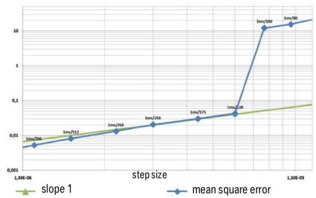
4. Exporting FMU using the Modelon FMI Toolbox
5. Integrating the powertrain model using the OpenXWD framework

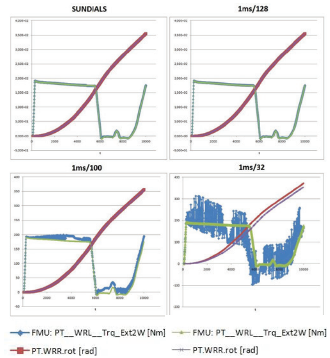
→ Use of „Integration Substeps“ with Euler solver possible



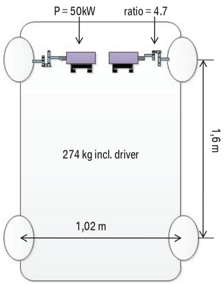
TEST OF NUMERICAL STABILITY.

- Analysing error with different step sizes
- Mean square error over step size shows euler convergence in stabilized area



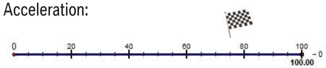


DESIGN EVALUATION.

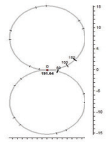


How does the variation of Engine-Power influences the 3 FSE competitions?

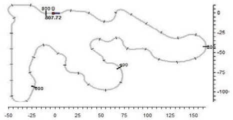
Acceleration:



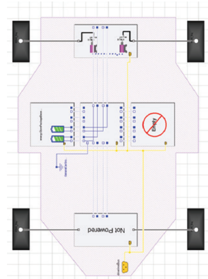
Skidpad:



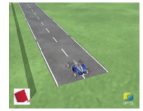
Endurance:



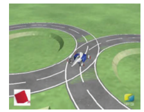
DESIGN EVALUATION.



Acceleration:



Skidpad:

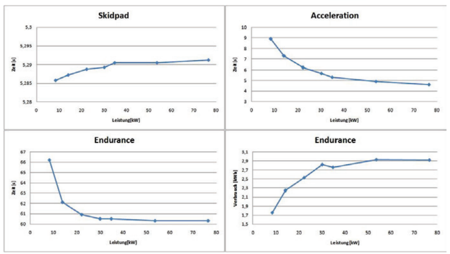


Endurance:

DESIGN EVALUATION.

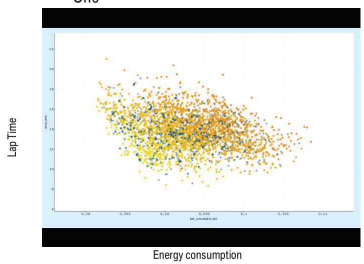
	M1	M2	M3	M4	M5	M6	M7	M5
Motor-Speed[rpm]	7600	6000	4000	4320	3300	2900	2300	3300
Motor-Torque[Nm]	90	80	80	64	60	43	32	60
Motor-Power[kW]	76,5	53,6	34,8	30,1	22,2	14	8,3	22,2
Acceleration-Time[s]	4,62	4,91	5,31	5,67	6,2	7,32	8,91	6,1
Skidpad-Time[s]	5,29	5,29	5,29	5,29	5,29	5,29	5,29	5,29
Endurance-Time[s]	60,3	60,3	60,5	60,5	60,9	62,12	66,2	60,9
Endurance-Consumption[kWh]^2	2,93	2,93	2,76	2,82	2,53	2,24	1,76	2,39

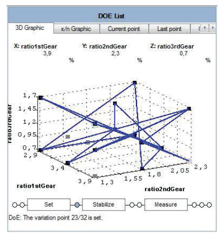
with reduced weight by 20 kg at 274 kg total weight



PROSPECT.

- Systematic Evaluation with Design of Experiments
- Localize optimized Trade-Offs





THANK YOU.

Fakultät für Elektrotechnik
und Informationstechnik

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

BMW Group