Approximate Formal Verification Using Model-Based Testing

Rance Cleaveland

Professor of Computer Science, University of Maryland
Executive and Scientific Director, CESE

19 Oct. 2013

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Fraunhofer USA Center for Experimental Software Engineering

- Applied-research institute in software engineering
- Founded 1998; in University of Maryland (UMD) technology park
- Staff: 30 – 16 technical (10 PhDs), 12 students / visitors
- Annual budget: US $4.5m
- Part of Fraunhofer USA / Gesellschaft; affiliated with UMD

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

CESE Overview

- Mission

  Better software-development technologies, practices and processes

- Technical expertise

  Software design, verification and validation, project management

- Target sectors

  Aerospace / defense, medical, automotive

- Biggest customer

Fraunhofer?

- German non-profit network of 80+ applied-research institutes
  - University-affiliated
  - Founded 1949
  - Named for inventor, entrepreneur Joseph von Fraunhofer
  - €2.1bn revenues, 20K employees
  - International presence
- Inventor of MP3
- Fraunhofer USA: subsidiary of FhG; $40m, 220 employees

Joseph von Fraunhofer

(1787-1826)

Fraunhofer USA

Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

The Fraunhofer Model

- Institutes act as conduit between universities, industry and government
  - Institutes affiliated with universities
    - Professors
    - Students
    - Professional staff
  - Single institutes focus on one discipline
- Funding: Some base, mostly projects with industry, government

Formal Methods

- Mathematically rigorous approaches to specifying, verifying systems
- Why? To increase confidence!
  - If the specification is trusted, verification establishes trust in system
  - If specification is not trusted, proving it is consistent with system builds trust in both

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Specifications

- Preconditions / postconditions, e.g.
  - Pre: $|A| > 0$
  - Post: $\forall i.\ 0 \leq i < |A|-1 \Rightarrow A[i] \leq A[i+1]$
- Temporal logic, e.g.

  $G\ (req \rightarrow F\ grant)$
- State machines
- Etc.

Fraunhofer USA

Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Verification = Proof

- Model checking
  - Proof constructed automatically
- Theorem proving
  - Proof constructed “automatedly”

Status of Formal Methods

- Noteworthy successes!
- We are not at the stage where success is expected

Why?

- "Scalability"

  Building proofs is laborious

- Inability to predict level of effort
  - *Difficulty of proof not correlated to usual measures of system complexity*
  - *Effort needed to coax proof out of tools not easy to estimate*

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

A Different Paradigm

- PWYC / TWYC
  - "Prove what you can."
  - "Test what you can't."
- In other words, *approximate verification*
  - Devise formal-specification frameworks that enable proofs
  - Ensure specifications also support less complete methods
    - Testing
    - Inspections
    - Etc.

Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

What This Talk Is About

- An idea for putting PWYC / TWYC into practice
- Key ideas
  - *Model-based testing (MBT)*
    - Models used as software specifications
    - MBT used to check equivalence between specs, software
  - *Instrumentation-Based Verification (IBV)*
    - Specs given in same notation as software
    - Verification = instrument software, determine if errors present

Talk Agenda

- Automotive software and model-based development
  - MATLAB® / Simulink® / Stateflow®
  - Verification questions in MBD
- MBT in MBD
- IBV in MBD
- Conclusions

Some Software Companies

ATI
ACCURATE TECHNOL
PHILIPS
ADVICS
Providing safety and comfort on roads around the world
SUZUKI
CATERPILLAR®
GUIDANT
HYUNDAI
TOYOTA
MBDA MISSILE SYST
Whirlpool CORPORATION
BOEING
Pratt & Whitney
A United Technologies Company
Ford
HELLA
Ideas today for the cars of tomorrow
LOCKHEED MARTIN
HONDA
HONDA CARS HO
MAGNA STEYR
more value • more car
AISIN
BOMBARDIER
mazda
Audi
SAIC
JOHN DEERE
HITACHI
Inspire the Next
LAND ROVER
BAE SYSTEMS
AIRBUS
Continental®
TEMIC
FUJITSU
GENERAL DYNAMICS
Raytheon
DENSO
bertrandt
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Technology
United Technologies
Sheer Driving Pleasure
dSPACE
DETROIT DIESEL CORPORATION
GM
Honeywell
KIA MOTORS
RENAULT
SIEMENS
DELPHI
Tomorrow's Technology
Elbit SYSTEMS
Jatco
VOLVO
NISSAN
Visteon
NORTHROP GRUMMAN
MITSUBISHI HEAVY INDUSTRIES, LTD.
DAIMLER
LOCKHEED MARTIN
DAF
Rockwell Collins
smiths
The MathWorks
TITAN
National Security Solutions
TRW

Automotive Software

- Driver of innovation

  90% of new feature content based on software [GM]

- Rising cost

  50% of Prius cost due to software [Toyota]

- Warranty, liability, quality

  High-profile recalls in Germany, Japan, US

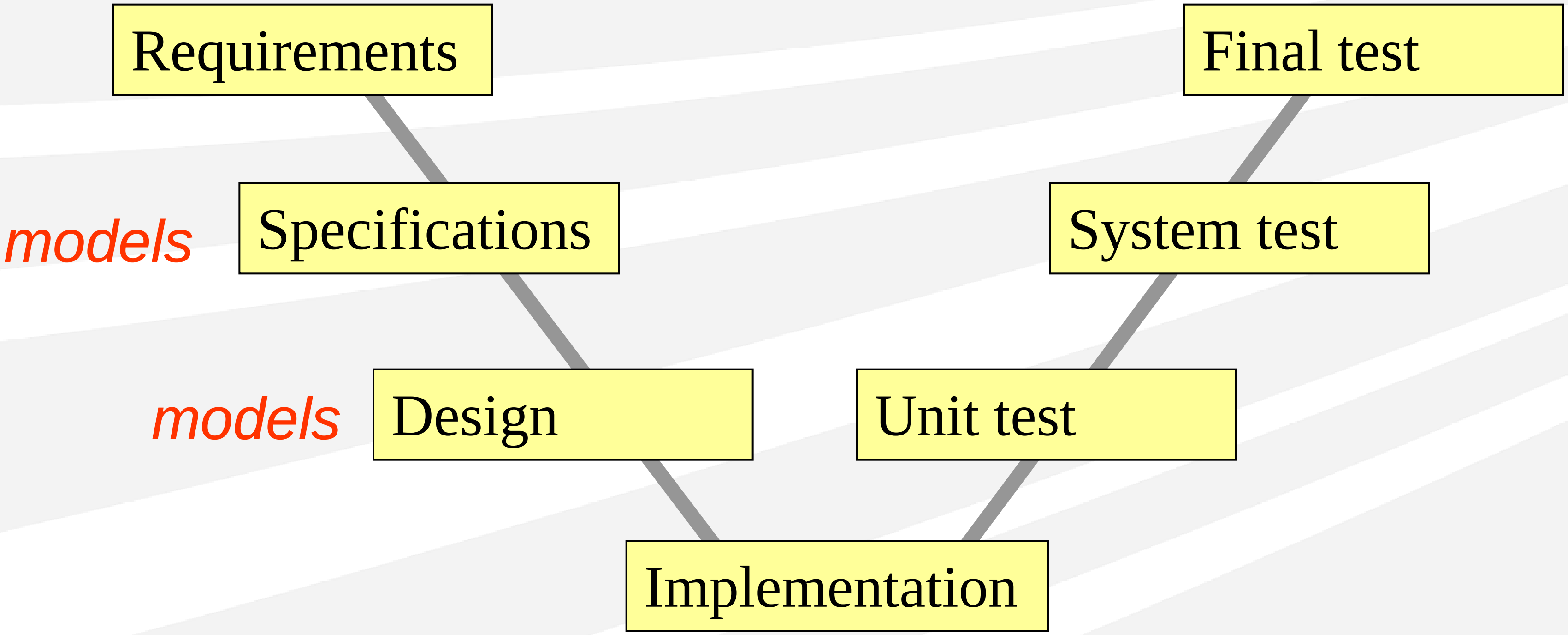
A Grand Challenge

- Ensure high quality of automotive software while
  - ... preserving time to market
  - … containing cost
- Key approach: *Model-Based Development* (MBD)
  - Use executable models during development
  - Dominant modeling language: MATLAB / Simulink / Stateflow

Fraunhofer USA
Center for Experimental Software Engineering

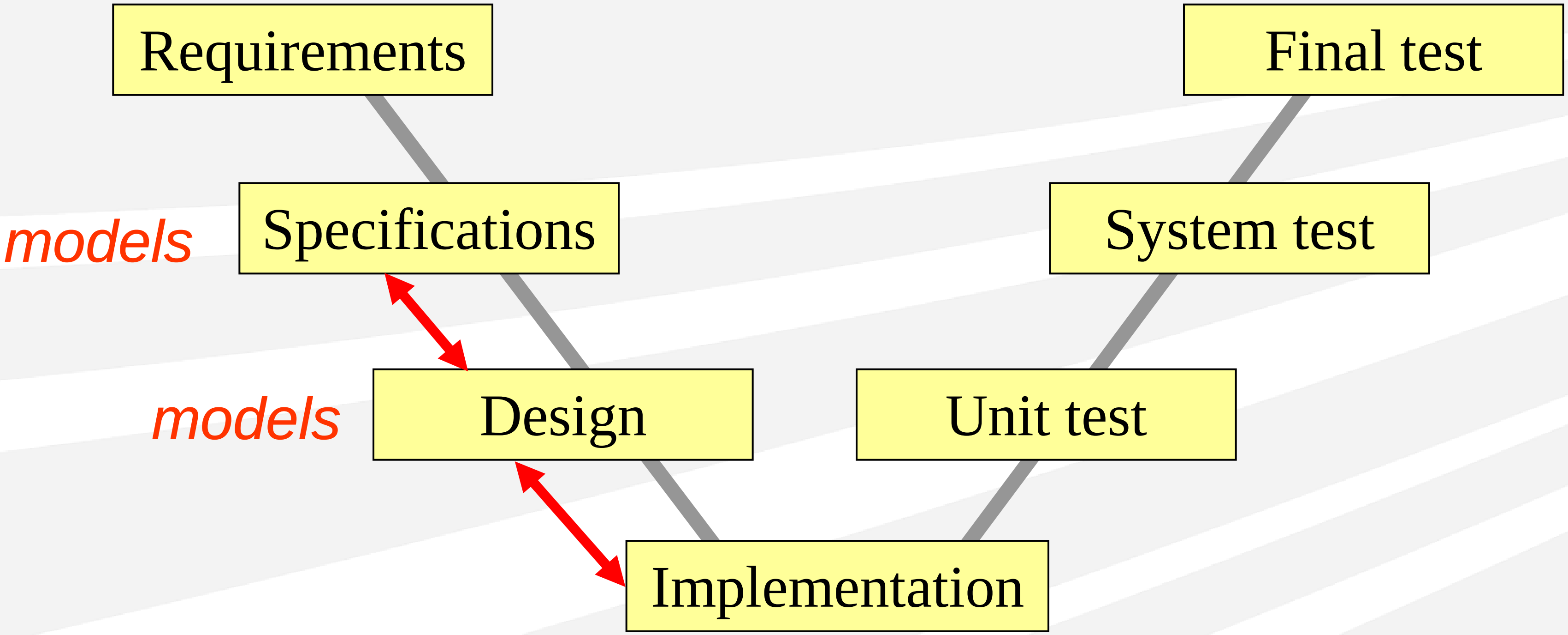
UNIVERSITY OF MARYLAND

Model-Based Development



Main Motivation: Autocode!

More Benefits of MBD

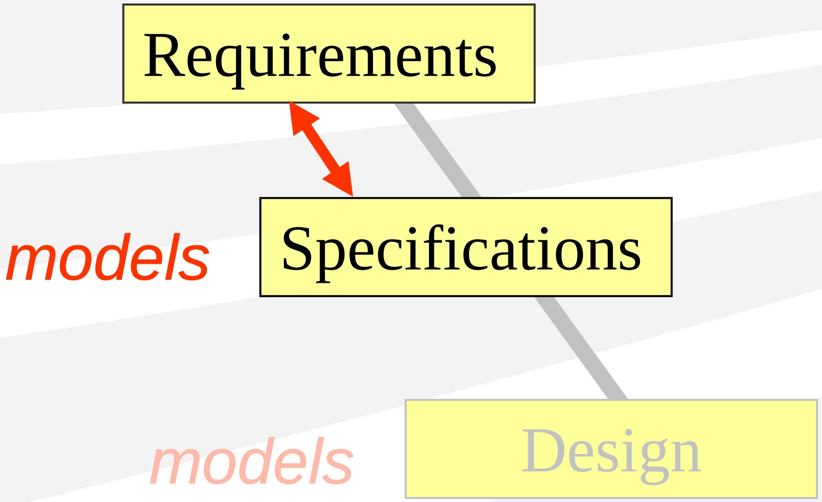


Models formalize specifications, design

Models facilitate communication among teams

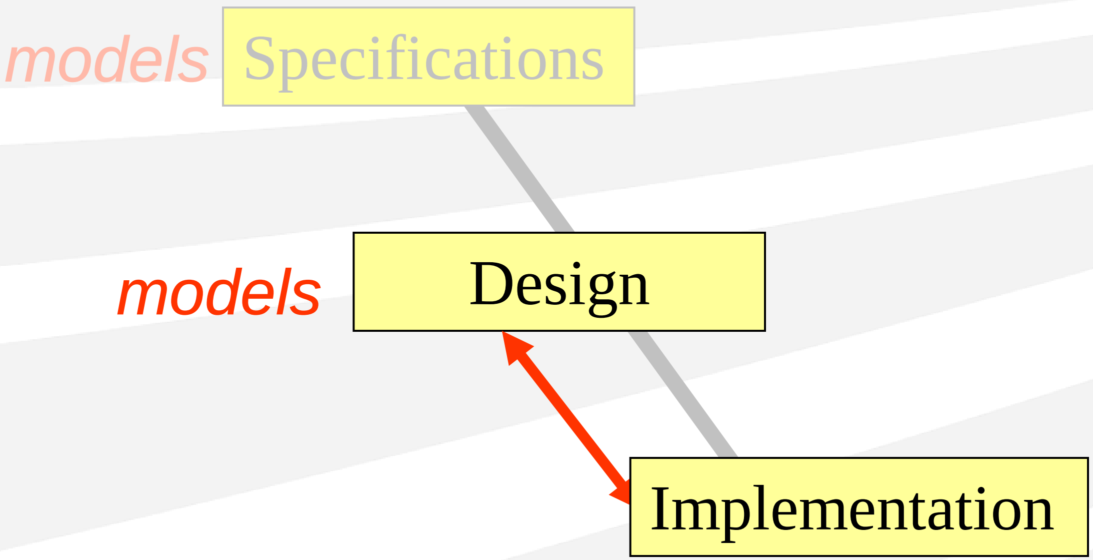
Models support V&V, testing, besides code generation

MBD Verification Problem #1



Do specs satisfy requirements?

MBD Verification Problem #2



Does implementation meet design?

Fraunhofer USA
Center for Experimental Software Engineering

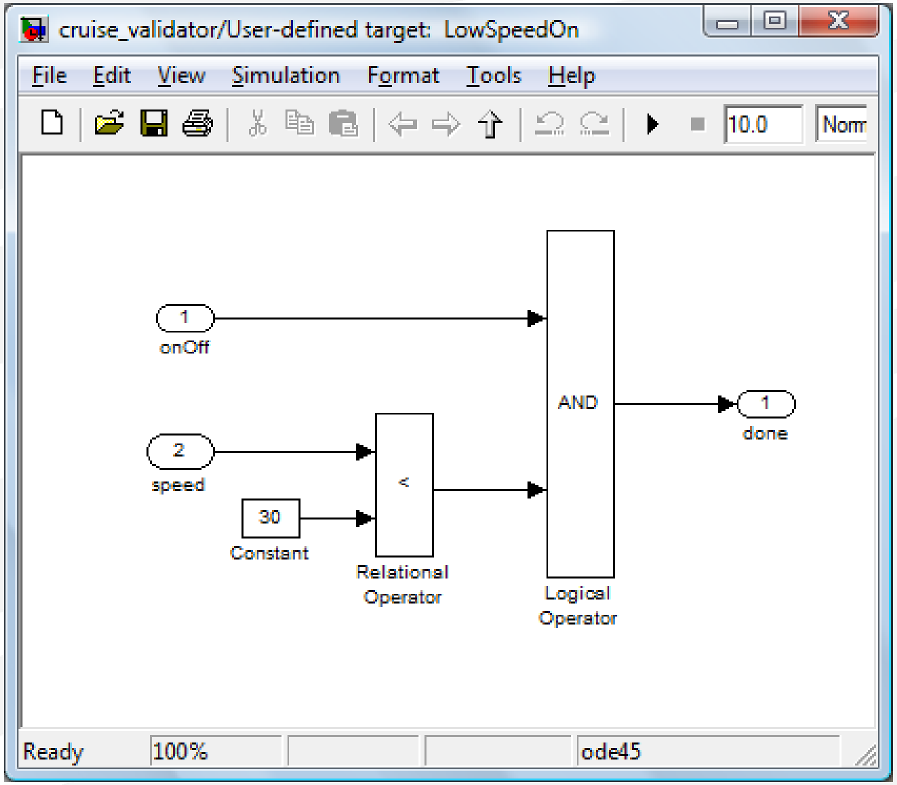
UNIVERSITY OF MARYLAND

PWYC / TWYC for MBD

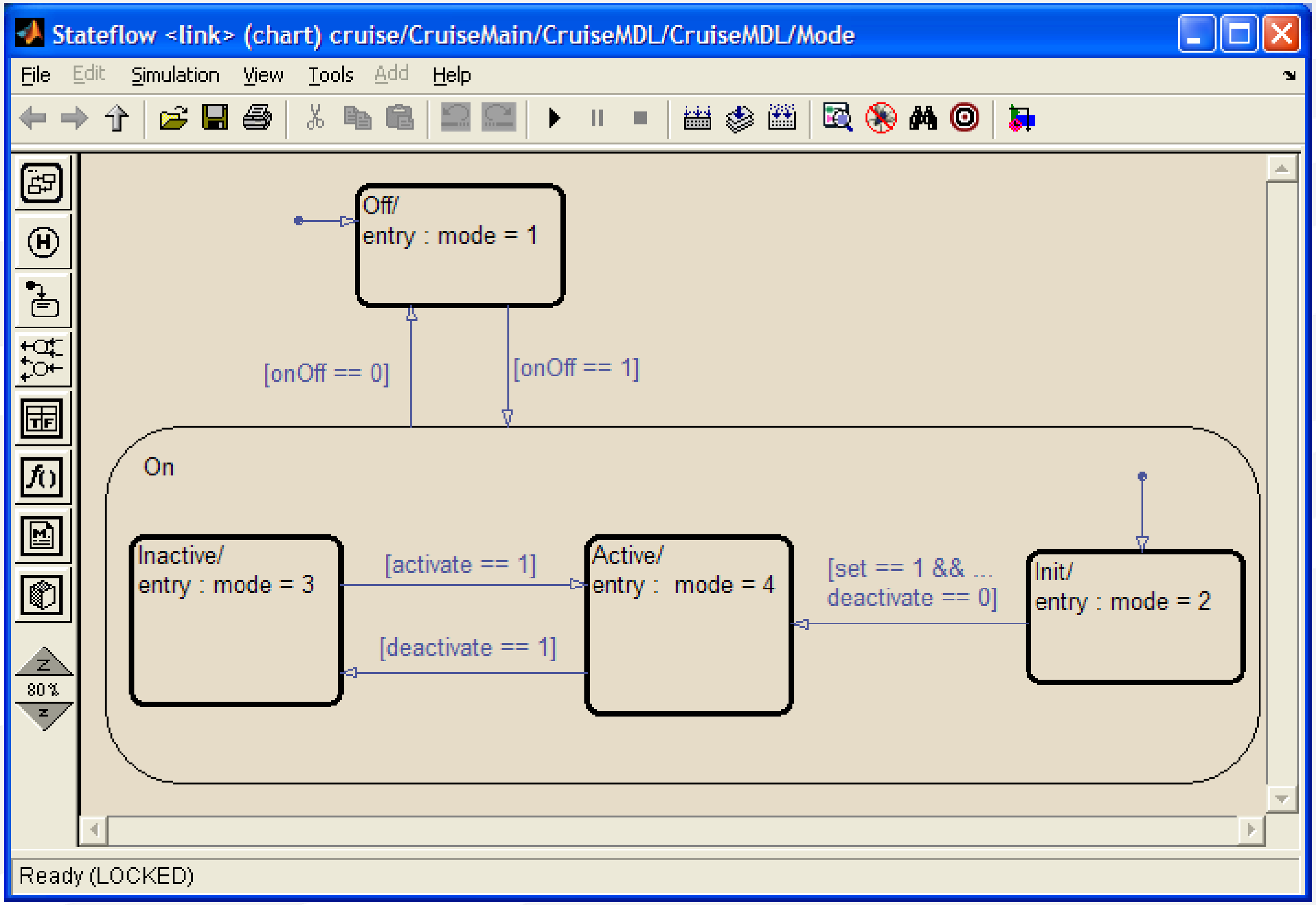
- Formalize verification problems mathematically
- Give testing-based *approximate* verification strategies
- Need Simulink semantics!

Simulink

- Block-diagram modeling language / simulator of The MathWorks, Inc.
- Hierarchical modeling
- Continuous- and discrete-time simulation



Stateflow



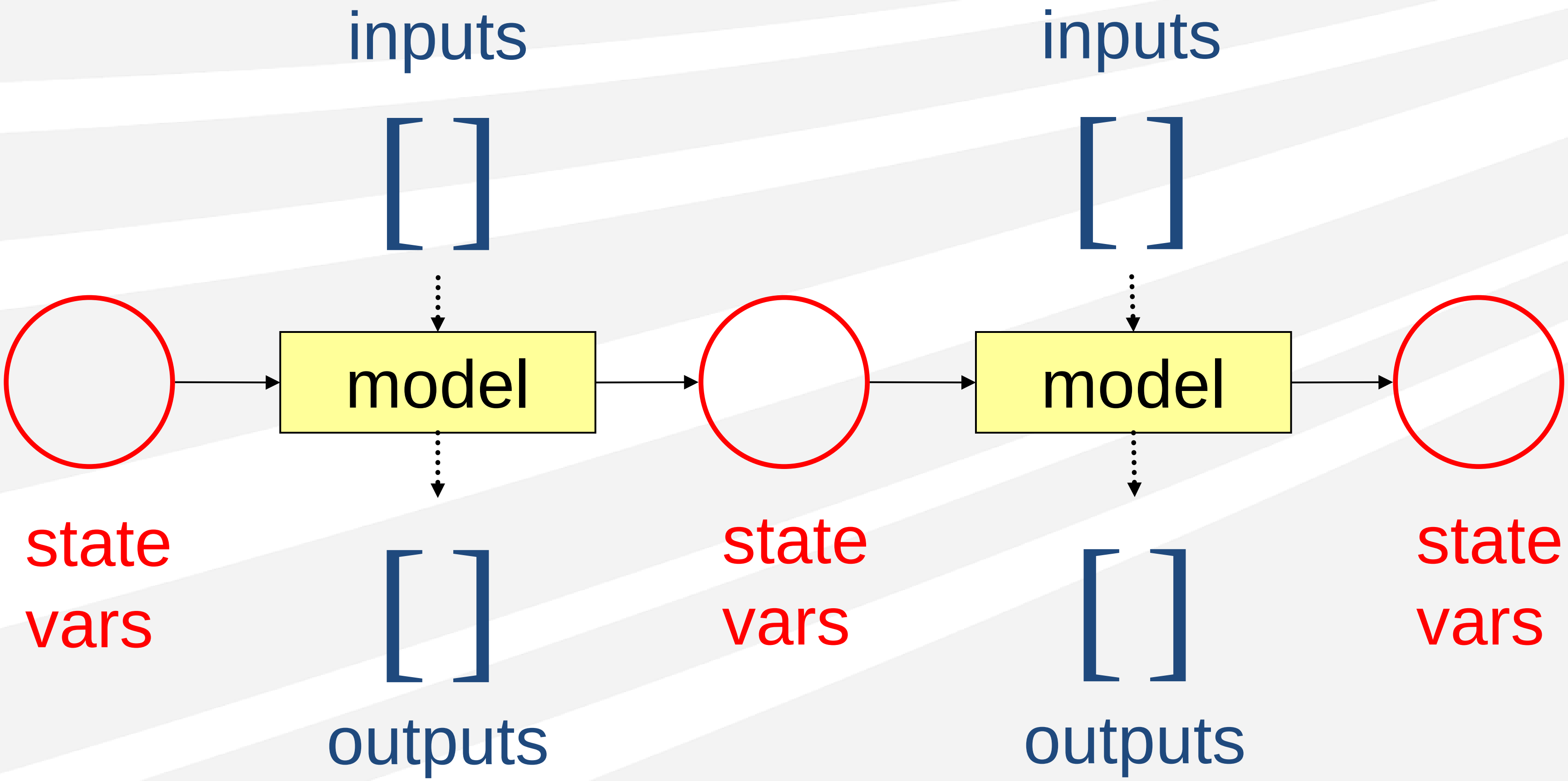
Fraunhofer USA Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Semantics

- Simulink has different "solvers" (= semantics)
  - Continuous: inputs / outputs are signals
  - Discrete: inputs / outputs are data values
- Analog modeling: continuous solvers
- Digital-controller modeling: discrete solvers
  - Synchronous
  - Run-to-completion
  - Time-driven

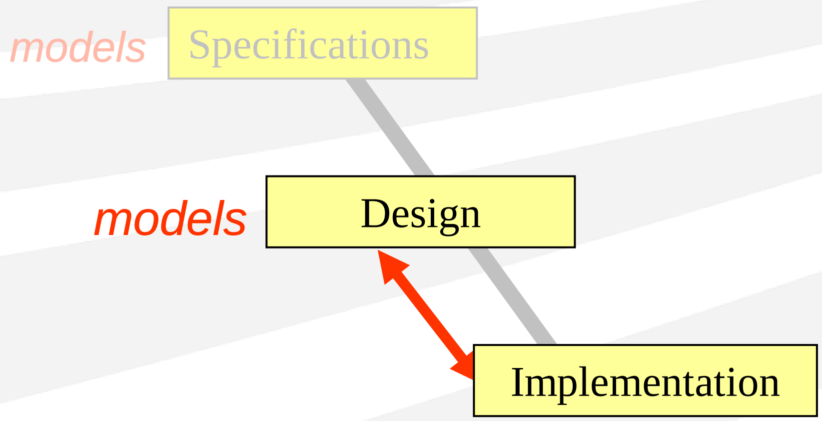
Discrete Solver Execution Model



Discrete Simulink Semantics

- Simulink models are Mealy machines
  - States: persistent state variables
  - Transitions: computed by model
- Can thus speak of *language* of model *M*
  - I = set of possible input vectors for *M*
  - O = set of possible output vectors for *M*
  - $L(M) = \{w \in (I \times O)^* \mid w$ is sequence of transition labels of execution of $M\}$

Formalizing MBD Verification Problem #2



- System *S* has language too!

  L(*S*) = possible sequences of input / output vectors
- MBD Problem #2
  - Given: model *M*, implementation *S*
  - Determine: does L(*M*) = L(*S*)?

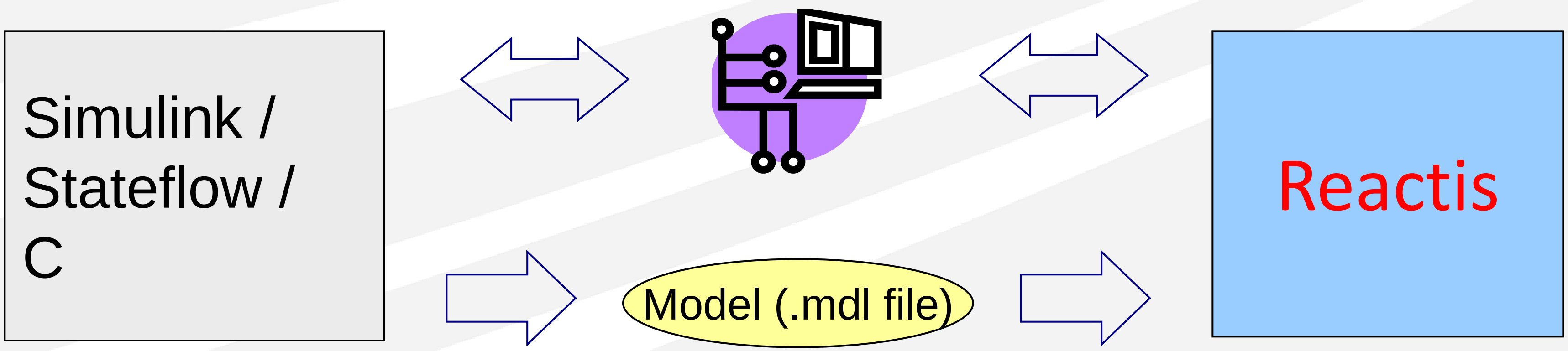
PWYC / TWYC for Problem #2

- Can prove instances of Problem #2
  - *M*, *S* are deterministic
  - Can use bisimulation-equivalence checkers to compute L(*M*) = L(*S*)
  - Not done in practice because state spaces too big
- To approximate verification: use testing
  - Generate test cases from *M*
  - Run them on *S*
  - Compare outputs

Reactis®

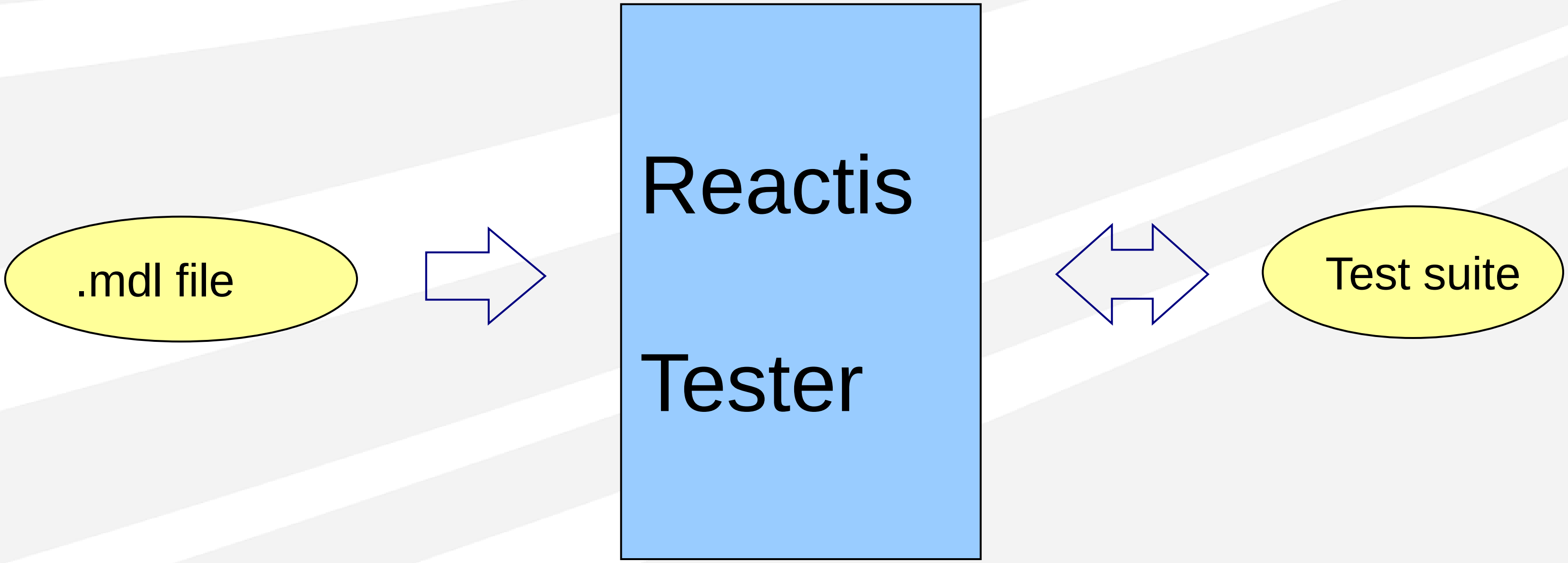
Automatic Simulink V&V tool from Reactive Systems Inc.

Tester	Generate tests from models
Simulator	Run, fine-tune tests
Validator	Validate models

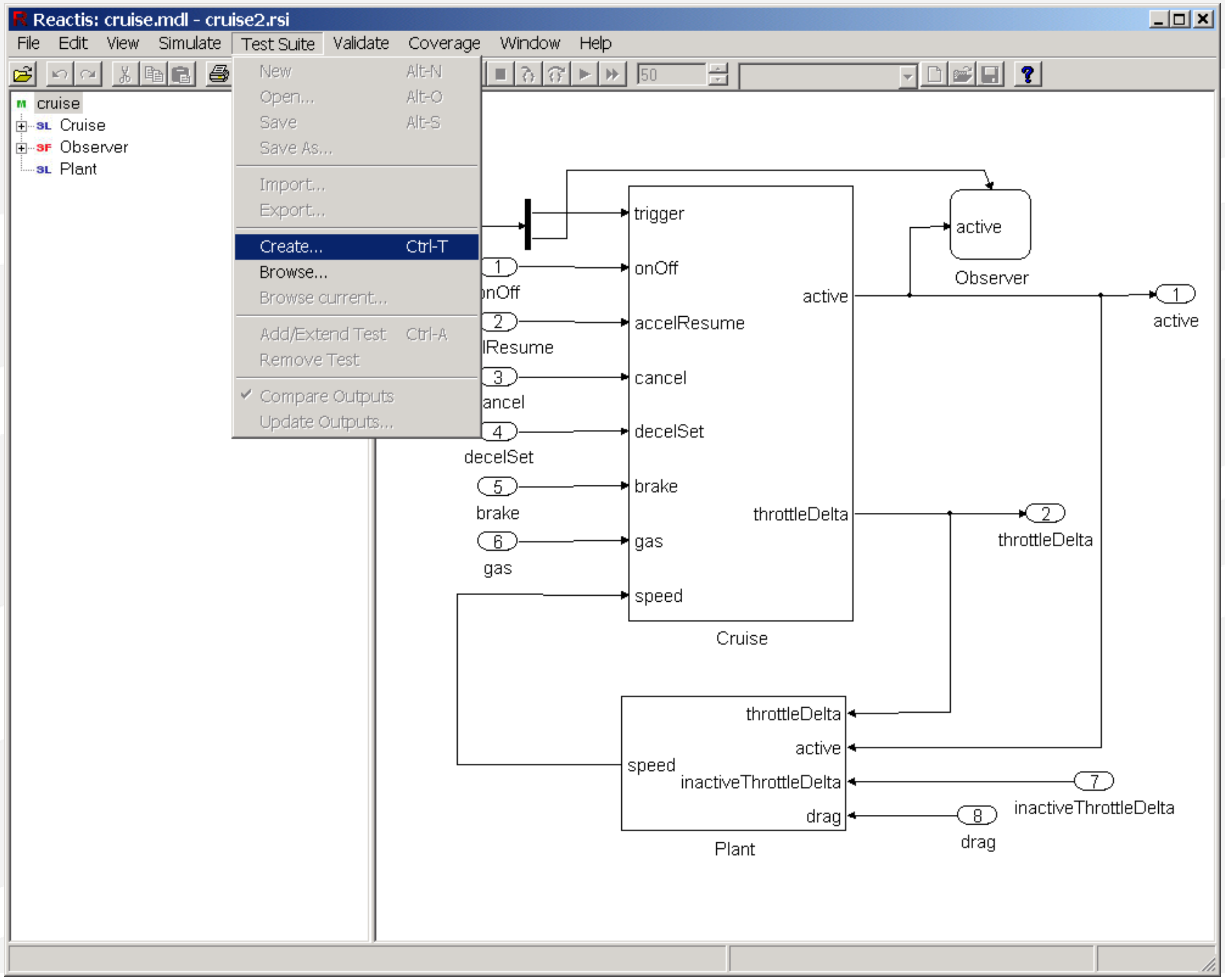


Reactis Tester

- Model in; tests out
- Model, tests in; better tests out



Launching Tester



Generated Test Data

Port	Step 1	Step 2	Step 3	Step 4	Step 5
Inputs					

1: onOff	0.0	1.0	0.0	1.0	1.0
2: accelResume	0.0	1.0	1.0	1.0	1.0
3: cancel	1.0	0.0	0.0	1.0	1.0
4: decelSet	0.0	0.0	1.0	0.0	1.0
5: brake	1.0	1.0	0.0	1.0	0.0
6: gas	1.0	0.0	1.0	0.0	1.0
7: inactiveThrottleDelta	0.1	0.0	0.1	-0.1	0.0
8: drag	-0.0093...	-0.0089...	-0.0094...	-0.0088...	-0.0089...
Outputs					

1: active	0.0	0.0	0.0	0.0	0.0
2: throttleDelta	-0.1	0.0	-0.1	0.0	0.0
___t___	0.0	1.0	2.0	3.0	4.0

Configuration Variable	Value
InitialSpeed	15.79179838897

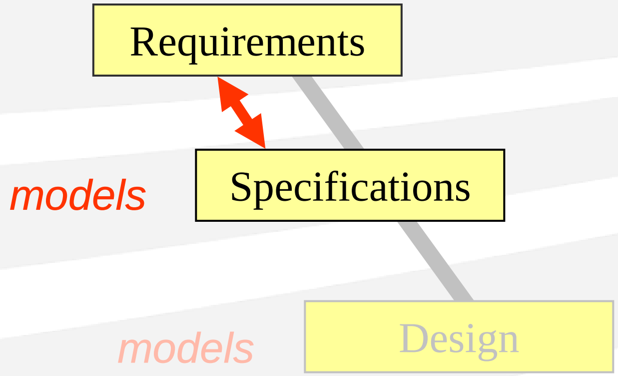
Test Generation with Reactis

- Test = simulation run = sequence of I/O vectors = element of L(*M*)
- Goal: maximize model coverage (e.g. branch, state, MC/DC, etc.)
- Method: guided simulation (US Patent 7,644,398)
  - Think: state-space search
    - Models = Mealy machines
    - Test generation = state-space traversal
  - Choose input data to guide search to uncovered parts of model (= transition computation)
    - Monte Carlo
    - Constraint solving (currently, linear constraints, SAT)

Experience

- Main use case for Reactis
- In use at 75+ companies around the world

Formalizing MBD Verification Problem #1

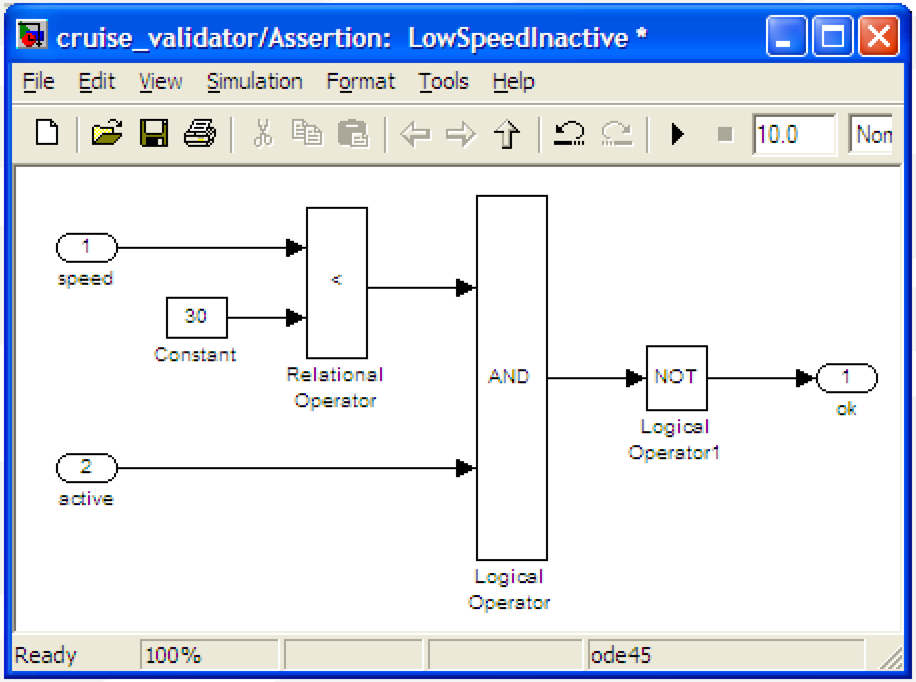


- We would like a PWYC / TWYC approach for this problem
- Need:
  - Formalized requirements
  - Formalized notion of satisfaction
- Approach: *Instrumentation-Based Verification*

Instrumentation-Based Verification: Requirements

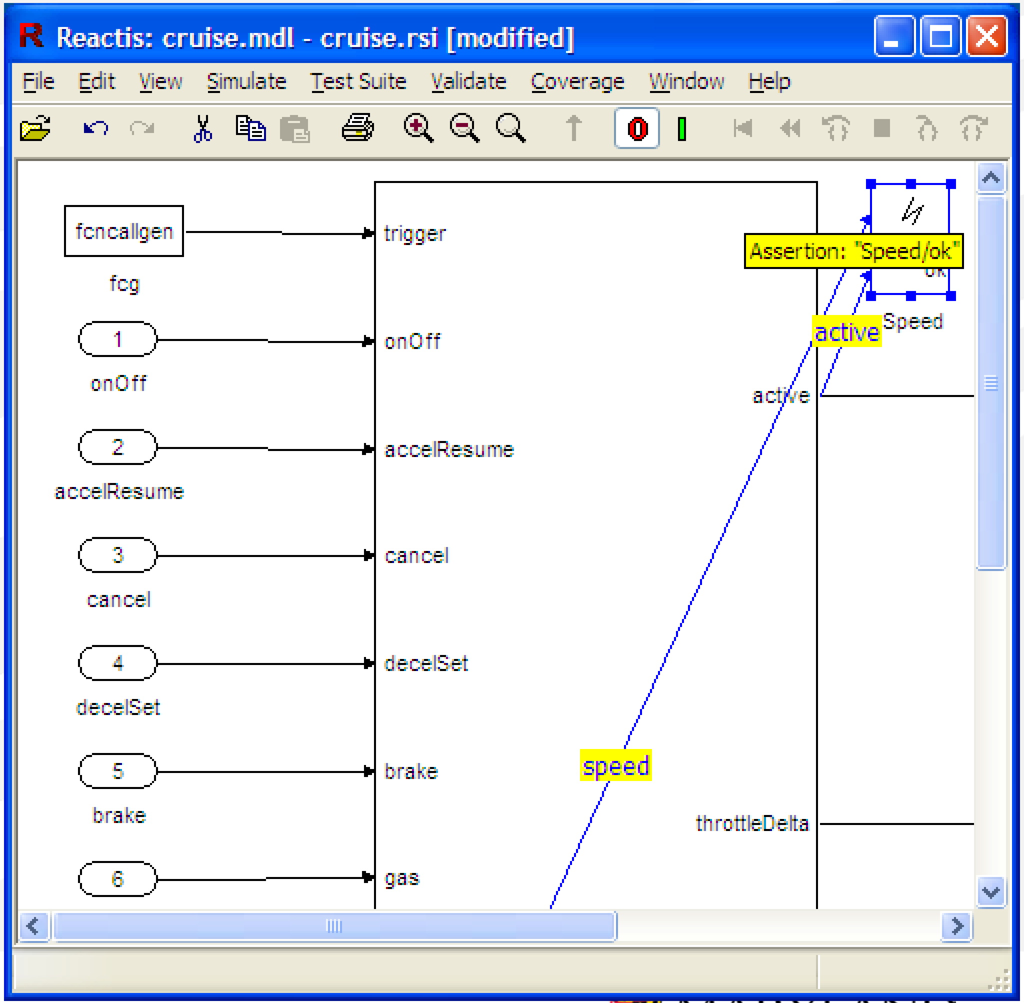
- IBV: formalize requirements as *monitor models*
- Example

  "If speed is < 30, cruise control must remain inactive"



Instrumentation-Based Verification: Checking Requirements

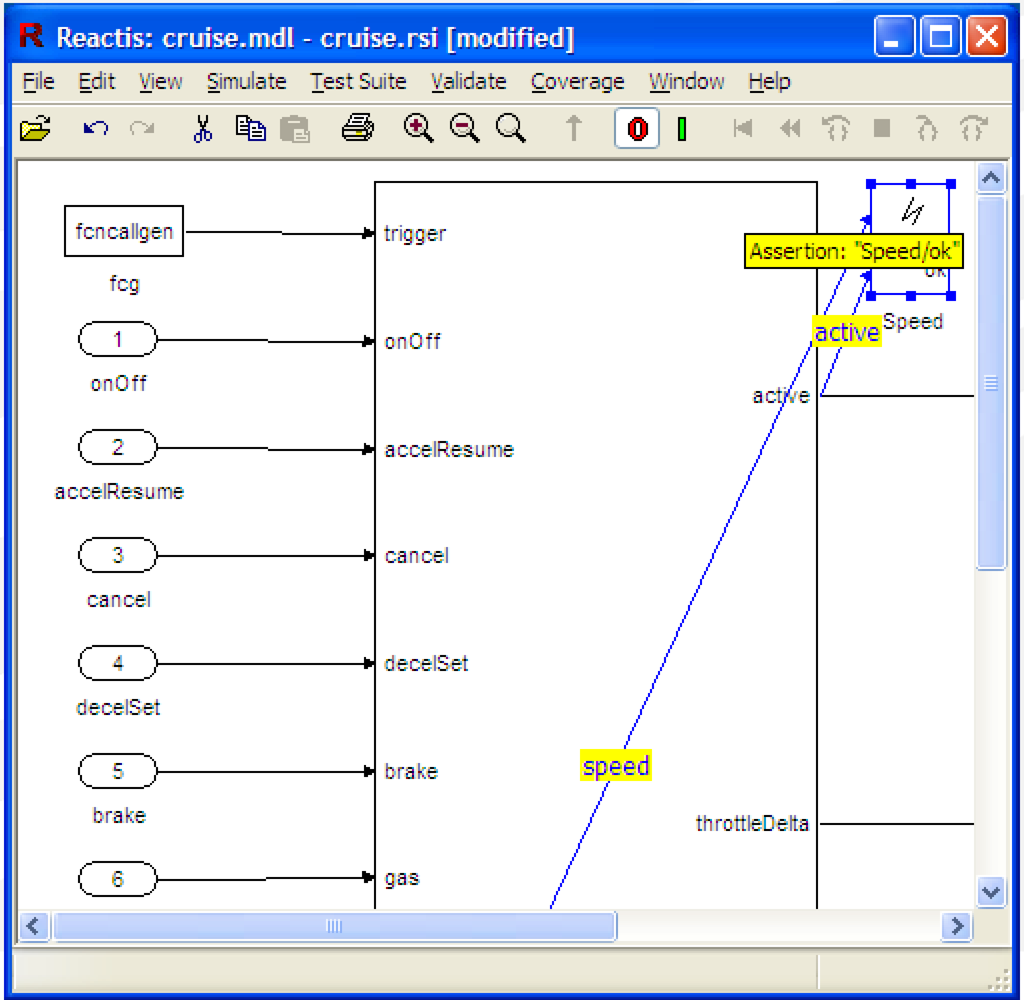
- Instrument design model with monitors
- Model satisfies monitors if:
  - For every input sequence *i*
  - Every monitor model only outputs *true*
- Reachability problem!
  - Proof possible
  - State-space an issue



Fraunhofer USA
Center for Experimental Software Engineering
MARYLAND

Approximate Verification for Problem #1

- Use coverage testing on instrumented model
  - Better scalability
  - If boolean coverage part of coverage criteria:
    - Test generator tries to make monitor outputs false as well as true
    - Skeptical testing!
- Reactis®
  - Supports instrumentation
  - Acts as skeptical tester
  - Reports violations



What About Model Checking?

- Temporal logic often used to formalize requirements
- Model checkers tell whether temporal-logic formulas are true or not
- Can this be adapted to model-based development?

Of Course It Can

- “Whenever the brake pedal is pressed, the cruise control shall become inactive.”

  AG (brake $\rightarrow$!active)

- “Whenever actual, desired speeds differ by more than 1 km/h, the cruise control shall fix within 3 seconds.”

  $AG(|speed–dSpeed|>1 \rightarrow AF_{\leq 3}|speed–dSpeed|\leq 1)$

Common Criticisms of Temporal Logic

- Formulas hard to comprehend for non-specialists

  Compare:

  AG (|speed–dSpeed| > 1 → AF$_{\leq 3}$ |speed–dSpeed| ≤ 1)

$$H(s) = P\frac{Ds^2 + s + I}{s + C}$$

$$\mathrm{Output(t)} = P_{\mathrm{contrib}} + I_{\mathrm{contrib}} + D_{\mathrm{contrib}}$$

$$P_{\mathrm{contrib}} = K_p\, e(t)$$

$$I_{\mathrm{contrib}} = K_i \int_0^t e(\tau)\, d\tau$$

$$D_{\mathrm{contrib}} = K_d \frac{de}{dt}$$

- Complex formulas hard to develop, understand

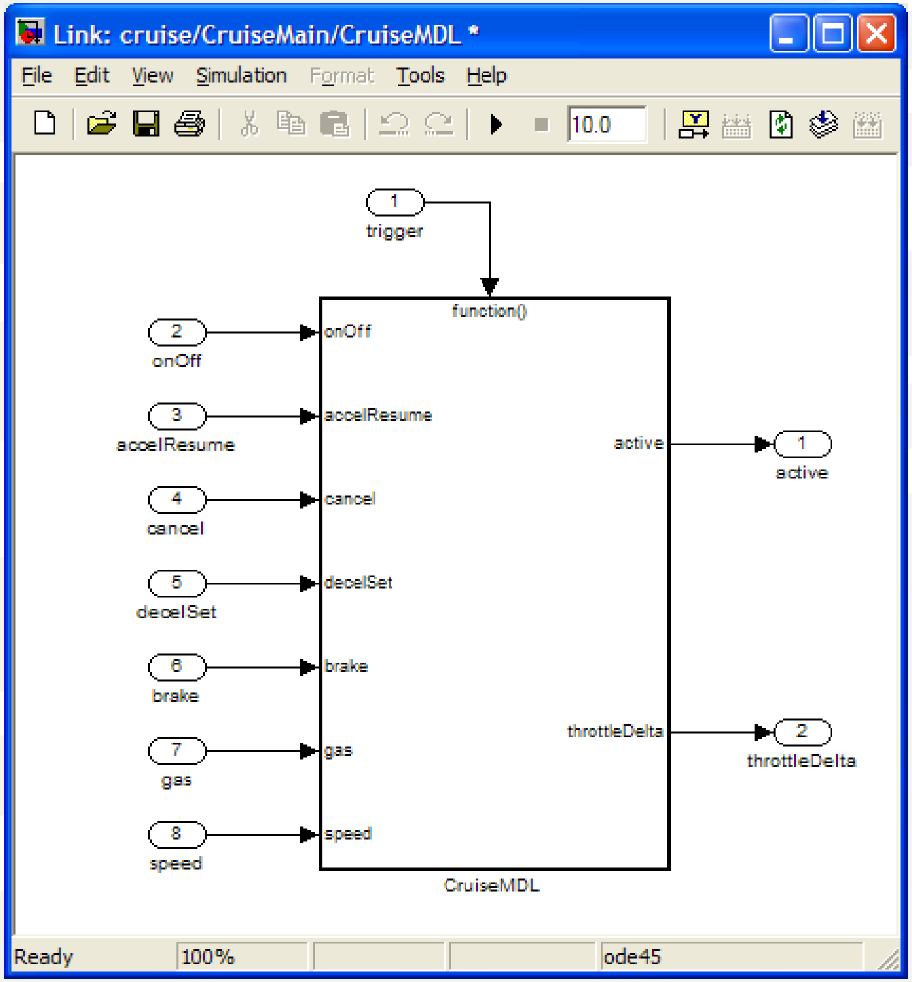
  An argument for simpler requirements?

Better Criticisms

- PWYC / TWYC?
- A second notation
- Development environment
- Scope issues

  AG (|speed – dSpeed| > 1 →
  AF$_{\leq 3}$ |speed – dSpeed| ≤ 1)

  "dSpeed"?
  - Not an input
  - Not an output
  - Internal variable!



IBV Addresses These Criticisms

- Instrumentation approach yields approximate verification opportunities
- One notation; existing tools can support requirements formalization, debugging
- Scope issues addressed implicitly
- Instrumentation is executable, hence debuggable
- Testing currently scales better than proof
  … but proof still possible with right tools

Related Work

- Run-time monitoring

  Havelund et al., Lee et al., Godefroid

- Automaton-based model-checking

  Holzmann et al., Vardi et al., Kurshan et al.

- Statistical model checking

  Clarke et al., Legay et al., Smolka et al.

Automotive Pilot Study #1

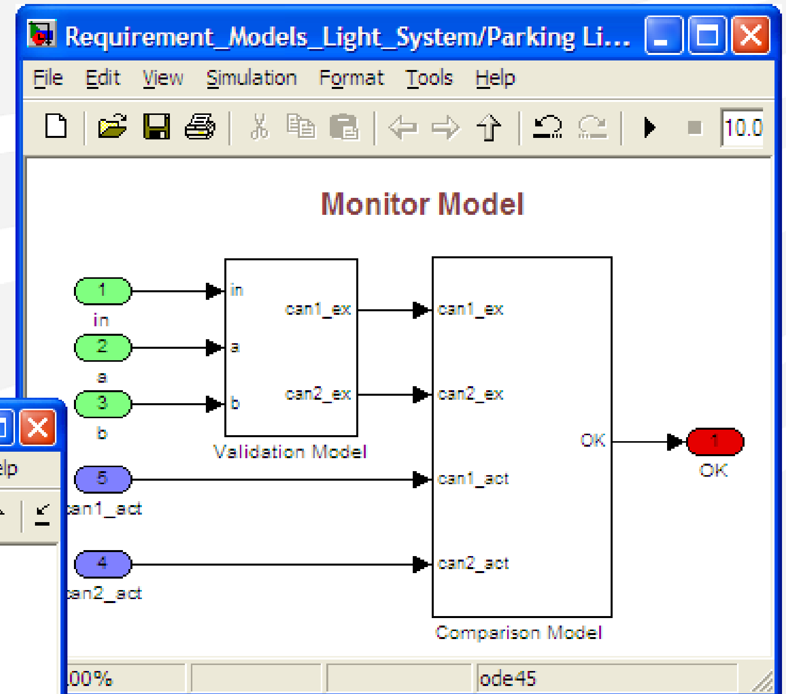
- Emergency Blinking Function (EBF)
  - Part of production body computer module (BCM)
  - Artifacts
    - BCM requirements document (300+ pages)
    - C code (200+ KLOC)
- Question: Will IBV work?

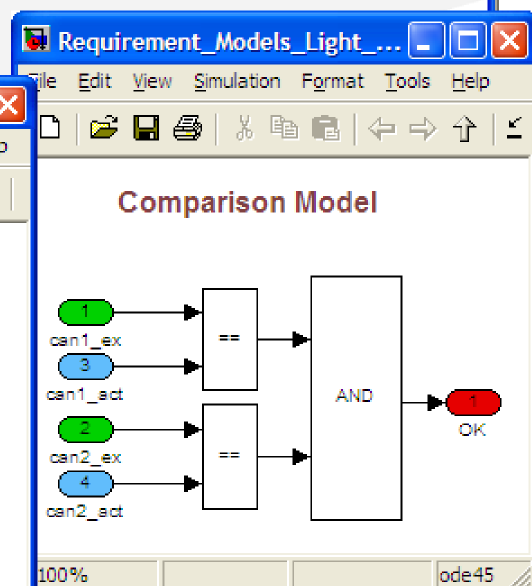
Pilot Study #1 (cont.)

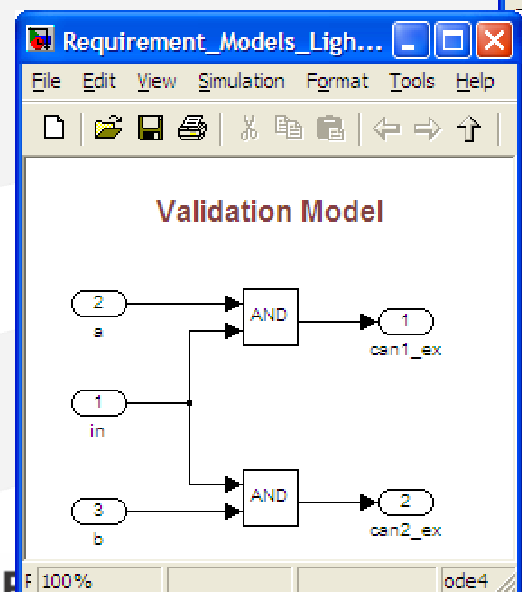
- Tasks
  - Code monitors from requirements
  - Code Simulink design model from C
  - Use Reactis to compare design, requirements
- Study details
  - Time frame: 3 months
  - Personnel: PhD student, Fraunhofer employee

From Requirements to Monitors: A Monitor Model Architecture

"[This] is the complete description of the control of the CAN output signals can1 and can2 produced by Function A. Function A can be activated only with in = 1. The activation takes place when either the CAN bus messages a or b is present…."







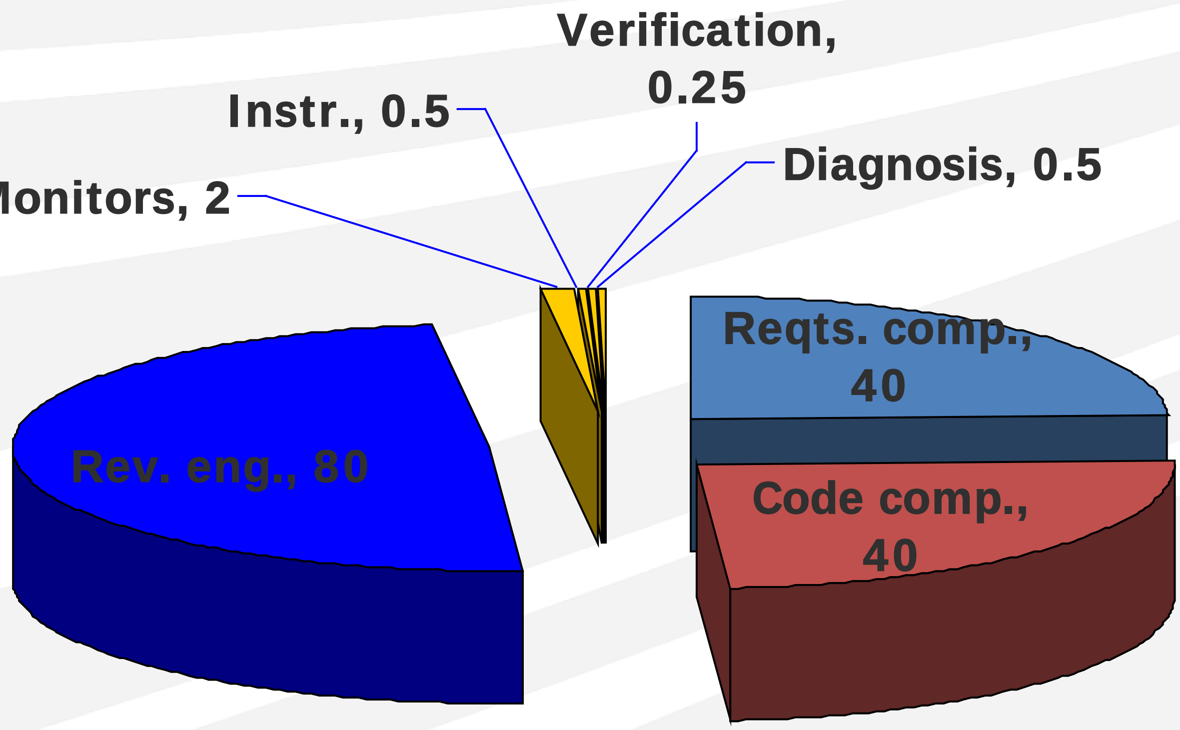
From Code to Models

- Goal: reverse-engineer model from code
  - Model-based design not used in development
  - Will IBV work for “production-strength” design?
- Part of EBF (250 SLOC) converted
  - Inports / state variables: read-before-write vars.
  - Outports: vars. written, not read
  - Resulting model: about 75 blocks

Conducting the Verification

- Reactis used to
  - Instrument model with monitors
  - Generate tests automatically
- Results
  - Test suites contained 80-120 test vectors
  - Time needed: .±20 sec
  - Omission in requirement discovered

Effort Data (Person-hours)



Preliminary Conclusion

- “It worked” …
- … for one feature
- … one (very complex) requirement
- … using PhDs

Fraunhofer USA
Center for Experimental Software Engineering

University of Maryland

Automotive Pilot Study #2

- More exterior-lighting functions
- More monitor models
- No PhDs: one intern
  - B.S. in Computer Science
  - Significant expertise in Simulink
  - No automotive experience

Approach

- Identify number of requirements for each exterior-lighting function
  - Count sentences
  - Read sections, beginning with fewest sentences
- Formalize requirements as monitor models
- Develop design models for functions
- Verify

Results

- 62 monitor, 10 design models created
- Enhancements to the monitor architecture
- Verification results
  - All IBV checks completed
  - Average check: 45 sec.
  - 11 issues in requirements

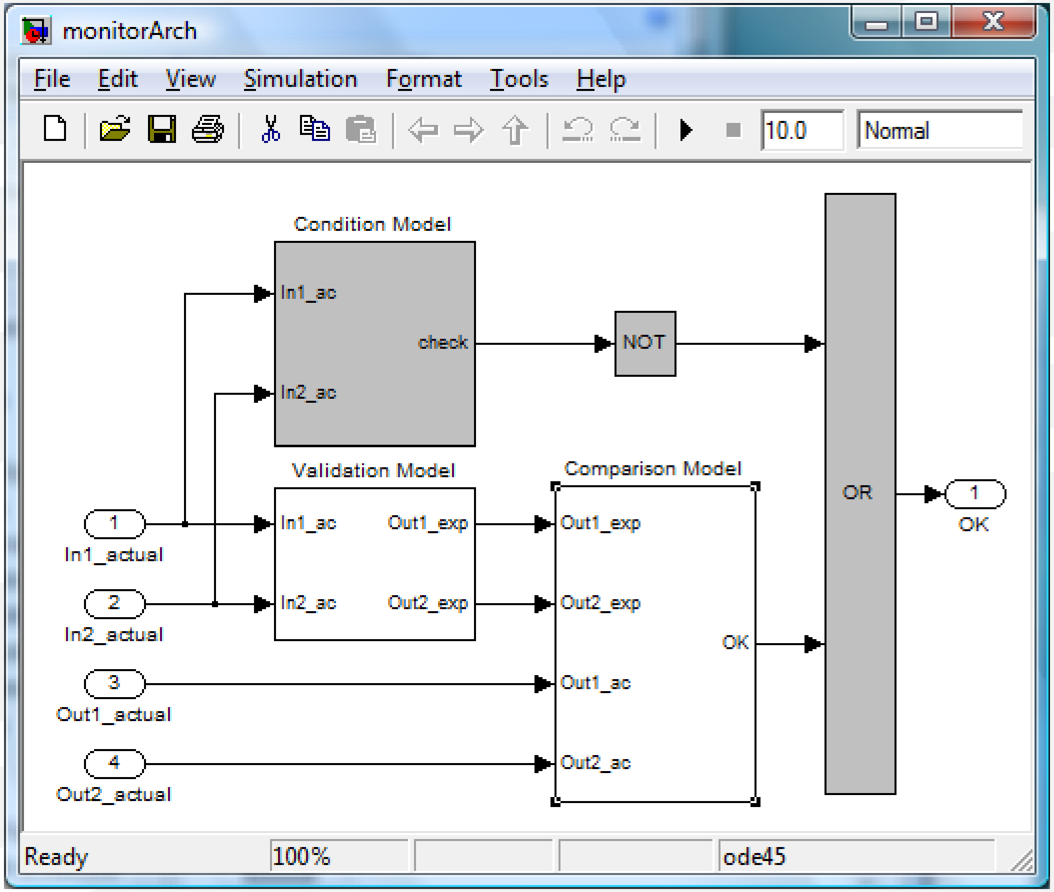
Fraunhofer USA
Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

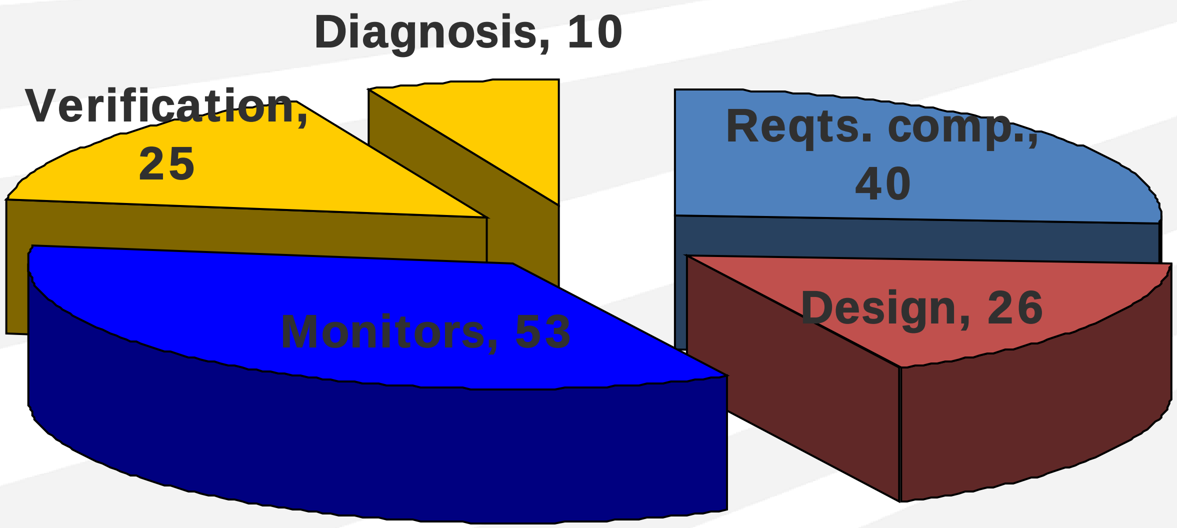
Monitor Model Architecture Change

Needed for *conditional requirements*

- Behavior only specified for certain situations
- “If timeout occurs switch off light”



Effort (Person-hours)



Discussion

- Requirements modeling
  - First study: 2 hours (1.2% of total) — 1 reqt. (2 hrs. / reqt.)
  - Second study: 53 hours (34.4% of total) — 62 reqts. (50 min. / reqt.)
- Design model development
  - First study: 80 hrs. (49.0% of total) — Reverse engg. (80 hrs. / model)
  - Second study: 26 hours (16.9% of total) — Forward engg. (2.6 hrs. / model)
- Verification
  - First study: 45 min. (0.5% of total) — 1 reqt. (45 min. / reqt.)
  - Second study: 25 hours (16.2%) — 62 reqts. (25 min. / reqt.)
- Fault diagnosis
  - First study: 30 min. (0.3% of total) — 1 reqt., 1 issue (30 min. / issue)
  - Second study: 10 hours (6.5% of total) — 62 reqts., 11 issues (55 min. / issue)

Yet More Discussion

- When did we “prove what we could”?
  – We didn’t …
  – … because of lack of IBV model checker and evident scaling problems
- Did we debug monitor-models while developing them?

  Yes!

Conclusions

- PWYC / TWYC approaches developed, applied for model-based development
  - *Model-based testing: equivalence checking*
  - *Instrumentation-based verification (IBV): requirements checking*
- Idea of PWYC / TWYC: gain benefit from formal specs even if formal verification infeasible
  - *Model-based testing: models serve as source of test data, oracles*
  - *Instrumentation-based verification (IBV): monitors act as oracles*
- Monitor models formalize requirements "efficiently"

  Reference architecture was a big factor
- Modeling "upstream" pays off "downstream"

  Design modeling easier after requirements modeling
- Requirements are not always what is required

  Requirements documents are often "just another description"

Fraunhofer USA

Center for Experimental Software Engineering

UNIVERSITY OF MARYLAND

Future Work

- An IBV model checker for Simulink
- How do you measure precision of approximate verification?
  - Reactis approach: test coverage
  - How does this correlate with "remaining bugs"? Or "proof remaining to be done"?
- System comprehension via testing, machine-learning
- Real-time model checking for IBV
- IBV for code

Thanks for your attention!

Rance Cleaveland

rance@cs.umd.edu

www.cs.umd.edu/~rance

For more on Simulink: www.mathworks.com

For more on Reactis: www.reactive-systems.com