ARTIST2 Summer School 2008 in Europe
Autrans (near Grenoble), France
September 8-12, 2008

An Instrumentation-Based Approach to Controller Model Validation

Lecturer: Rance Cleaveland
Professor of Computer Science, University of Maryland
Director, Fraunhofer Center for Experimental Software Engineering

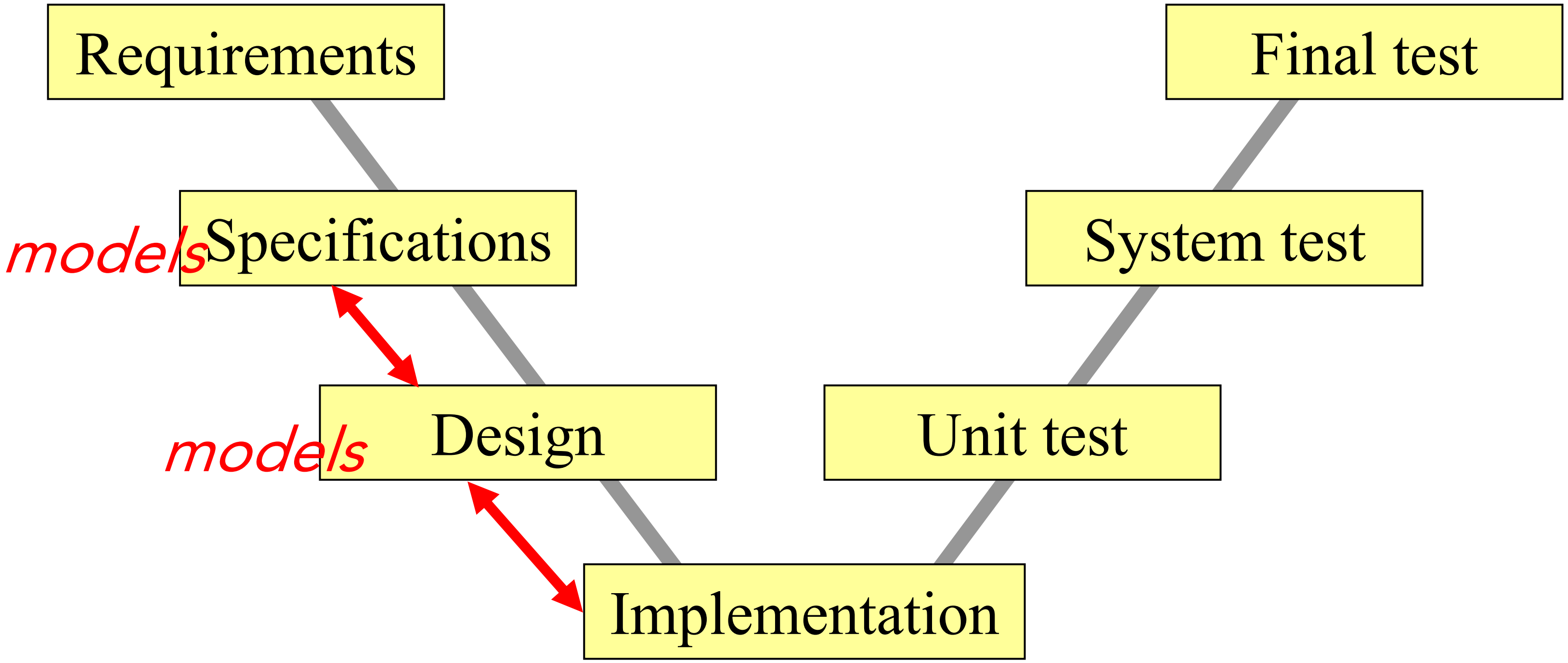
Some Software Companies

A Silent Revolution…

… in product design

- New features = software
- Software must be as reliable as mechanical / electrical components
  - Safety / liability
  - Warranty / recall costs
  - Product quality
- Challenge: productivity in face of
  - Increase in quantity
  - Requirements on quality
  - Uncertainty of software backgrounds of developers

(Model-Based) Development



- Models formalize specifications, design
- Models support V&V, testing, code generation
- Models facilitate communication

Types of Models

- Functional: behavior

  Block diagrams, state machines … (MATLAB® / Simulink® / Stateflow®)

- Non-functional: architecture / interfaces

  Class diagrams, … (UML)

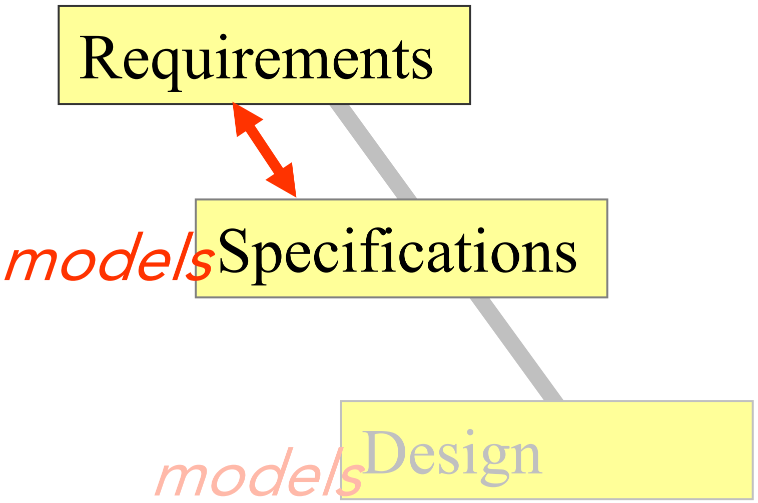
Types of Requirements

- Functional: behavior

  Characterize what should, should not happen when system is in operation

- Non-functional: architecture / interfaces
  - Characterize desired structural properties
  - Used for reusability, comprehensibility, modifiability, performance, …

An Ongoing Project



Functional / non-functional design-time modeling, requirements verification

This Talk

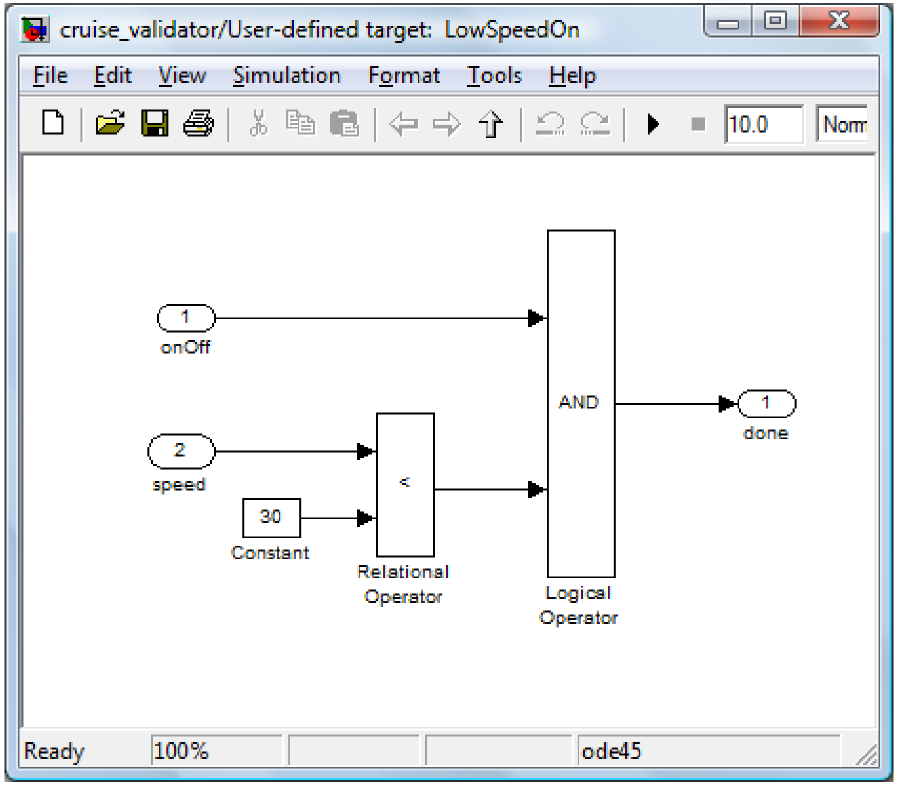
- Automated functional verification
  - Method: *Instrumentation-Based Verification*
  - Tools: *Reactis® / Simulink® / Stateflow®*
  - Case study: *production exterior-lighting control (automotive)*
- Results
  - Method imposed reasonable overheads in design process
  - Problems in requirements uncovered

Outline

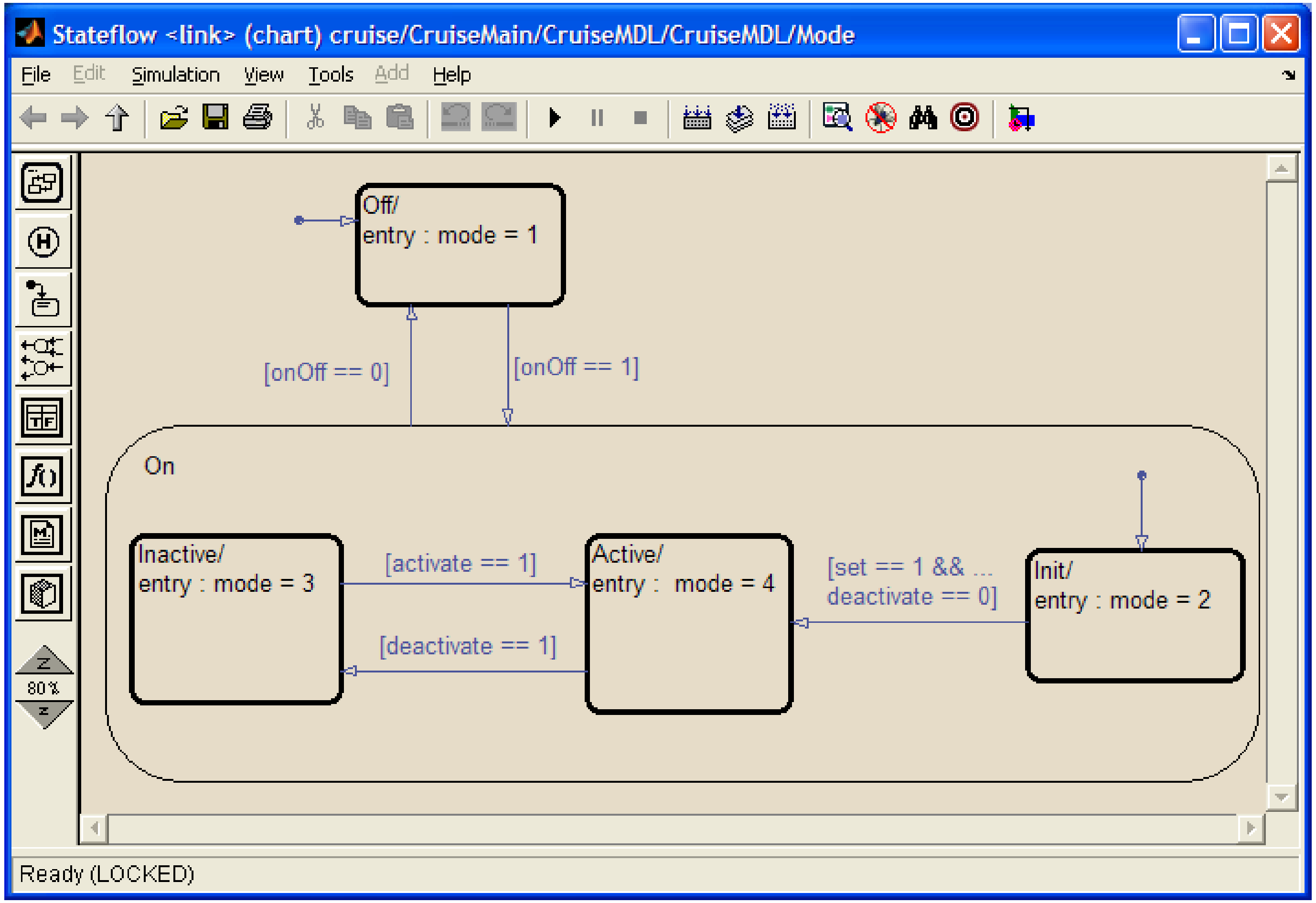
- Functional models: Simulink / Stateflow
- Instrumentation-based verification
  - Requirements as monitors
  - Verification via testing
- Case study
- Conclusions

Functional Models: Simulink

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



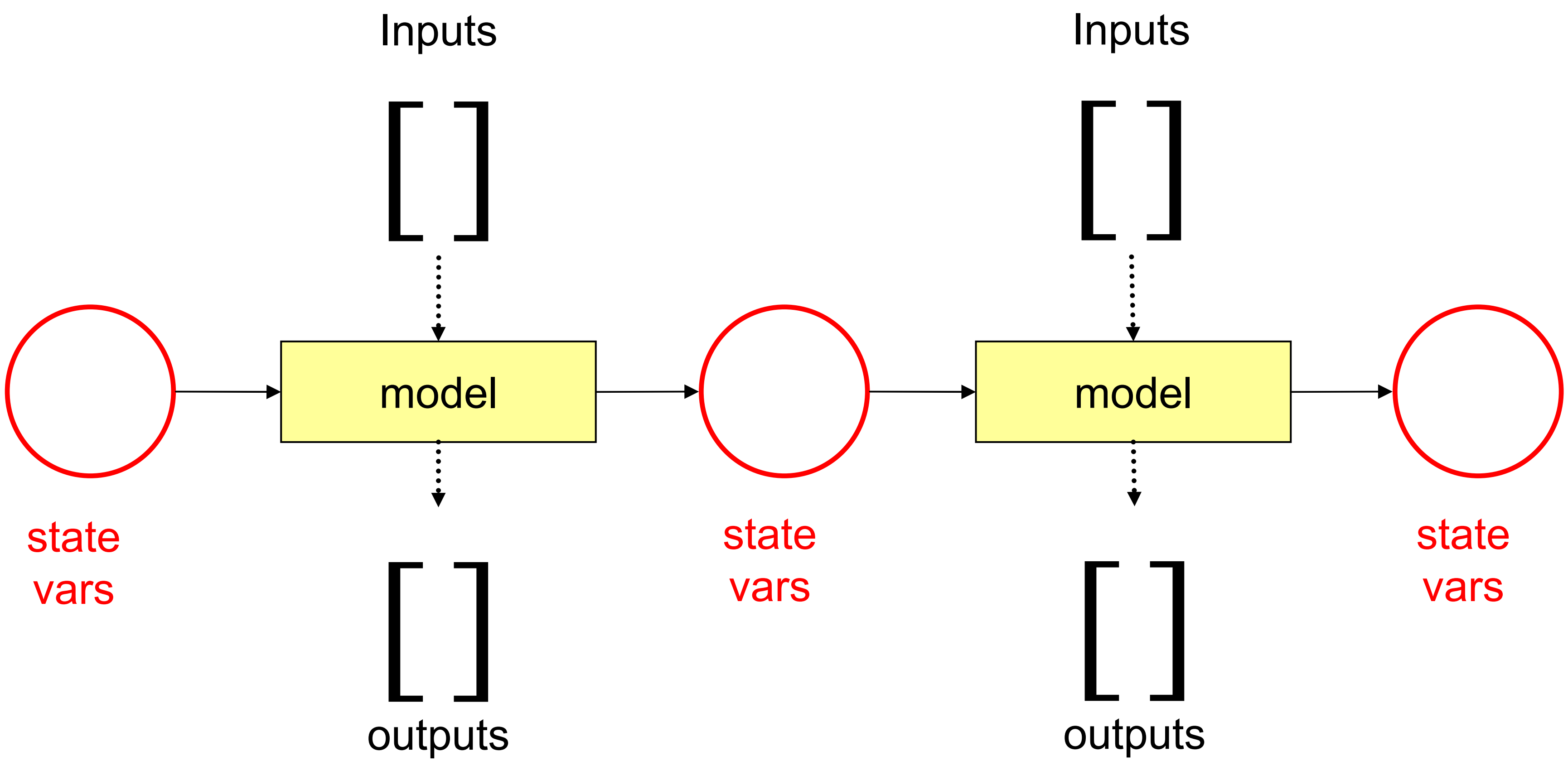
Models: Stateflow



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

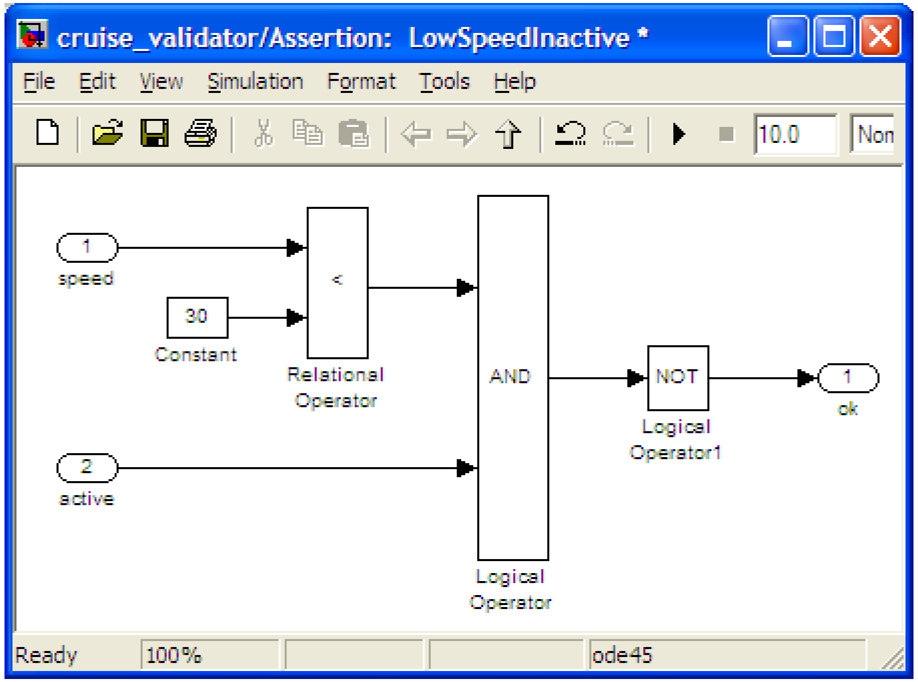
Discrete Solver Execution Model



Instrumentation-Based Verification: Requirements

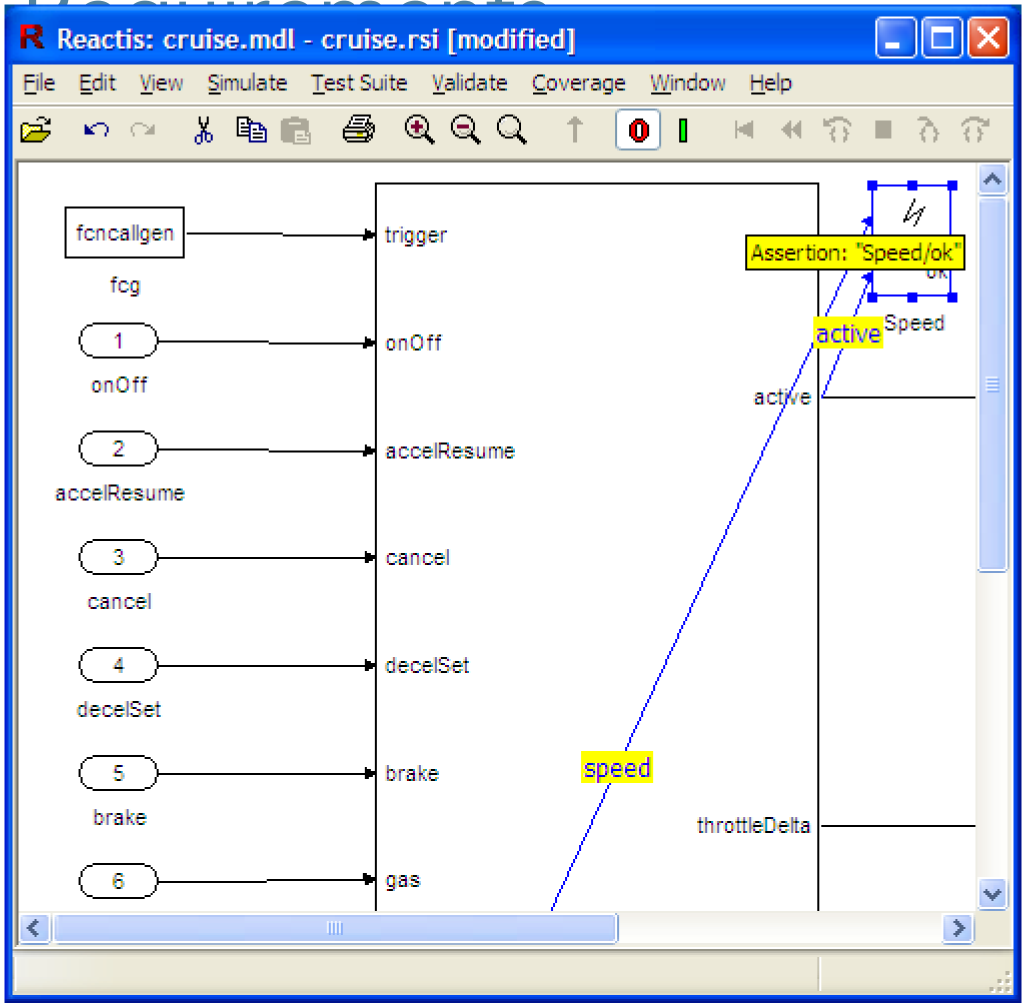
- Automatic verification requires formalized requirements
- IBV: formalize requirements as *monitor models*
- Example

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



Instrumentation-Based Verification: Checking Requirements

- Instrument design model with monitors
- Use coverage testing to check for monitor violations
- Tool: Reactis®
  - Product of Reactive Systems, Inc.
  - Separates instrumentation, design
  - Automates test generation



What about Temporal Logic 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?

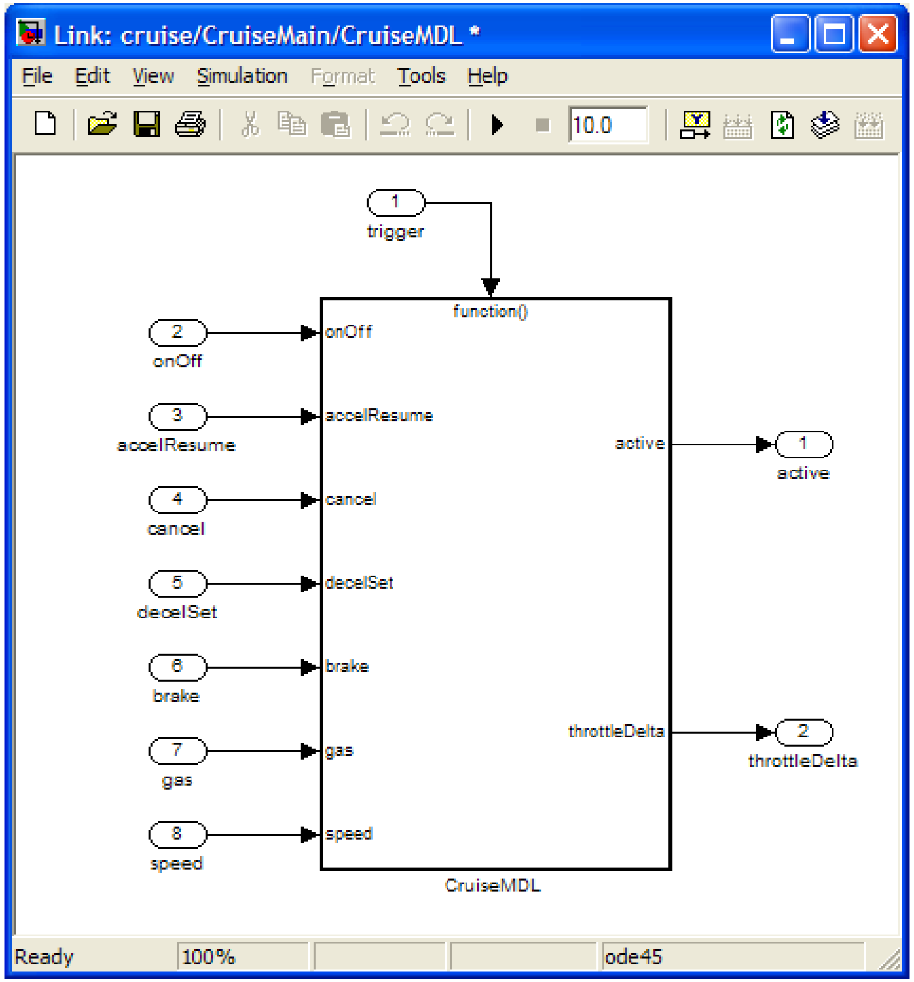
Better Criticisms

- A second notation
- Scope issues

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

"dSpeed"?

- Not an input
- Not an output
- Internal variable!



Model Checking

- Pros:
  - Full proofs of correctness, plus
  - Automatic!
- Cons:
  - Combinatorial complexity

    State-explosion: number of states grows exponentially in number of bits
  - Finite-state restrictions

IBV Addresses Criticisms

- One notation; existing tools can support requirements formalization
- Scope issues addressed implicitly
- Testing currently scales more easily than proof

Bosch Pilot Study

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

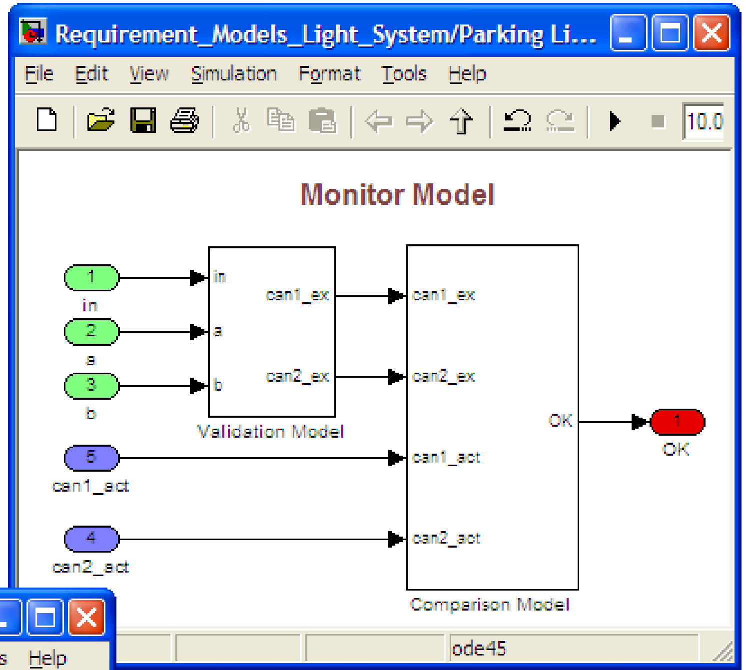
Pilot Study (cont.)

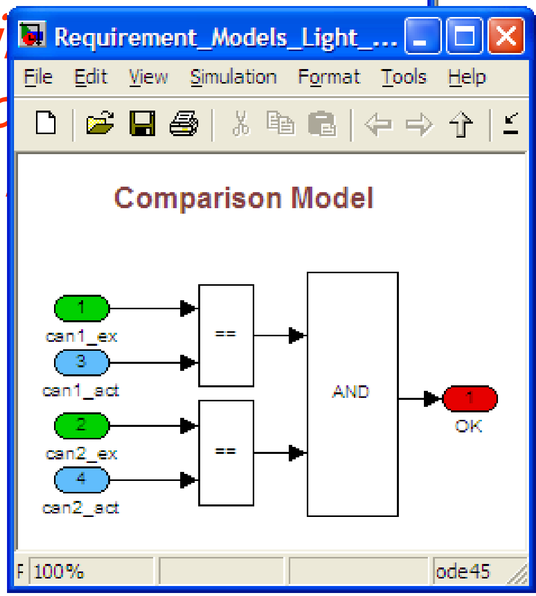
- Tasks
  - Code monitors from requirements
  - Code Simulink design model from C
  - Use Reactis to check design model against requirements
- Study details
  - Time frame: 3 months
  - Personnel: PhD student / Fraunhofer employee

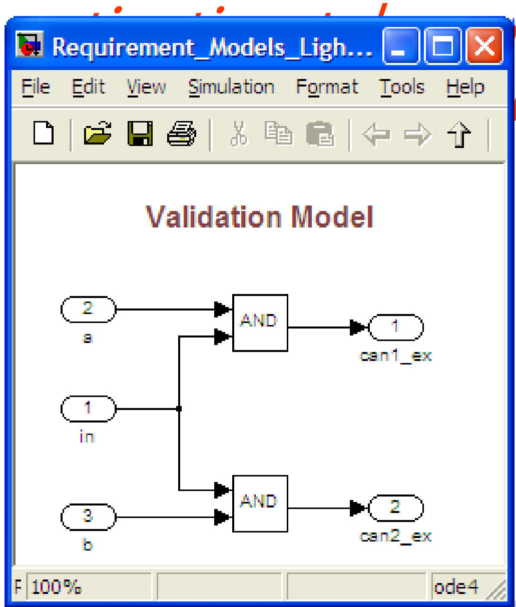
artist

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 i[illegible] [illegible]lac[illegible] [illegible]us [illegible]"







Information Society

From Code to Models

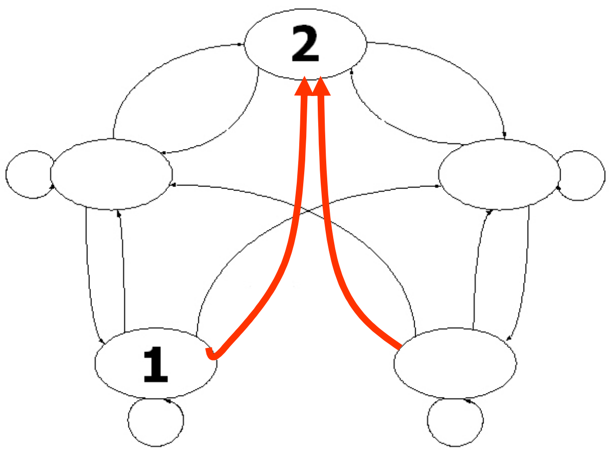
- Goal: reverse-engineer model from code
  - Model-based design not used in development
  - Desire was to see how IBV works for “production-strength” design
- Part of EBF (250 SLOC) converted
  - Inports / state variables: read-before-write variables
  - Outports: variables written, not read
  - Resulting model: about 75 blocks

Conducting the Verification

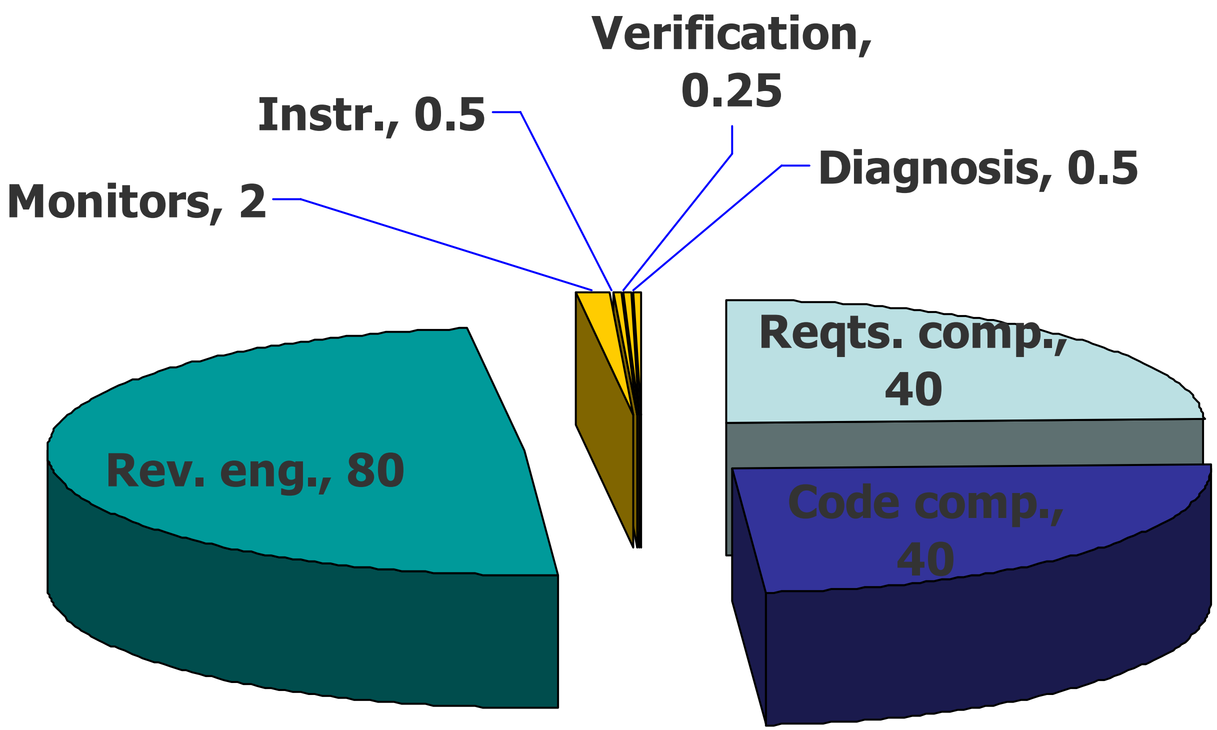
- Reactis used to
  - Instrument reverse-engineered model with monitors
  - Generate tests automatically
- Results
  - Generated test suites contained 80-120 test vectors
  - Omission in requirement discovered

Requirement Issue

- Missing reset transitions in state machine in requirements
- Code was correct



Effort Data (Person-hours)



Preliminary Conclusion

- “It worked” …
- … for one feature
- … a few requirements
- … using PhDs!

Another Study

- More exterior-lighting functions
- More monitor models
- No PhDs: one intern
  - BS in computer science
  - Significant expertise in Simulink®
  - No automotive experience

Approach

- Identify number of requirements for each exterior-lighting function
  - Count sentences; use as initial rough estimate for number of requirements
  - Read sections carefully in reverse order of number of sentences
- Formalize requirements as monitor models
- Develop design models for functions
- Verify

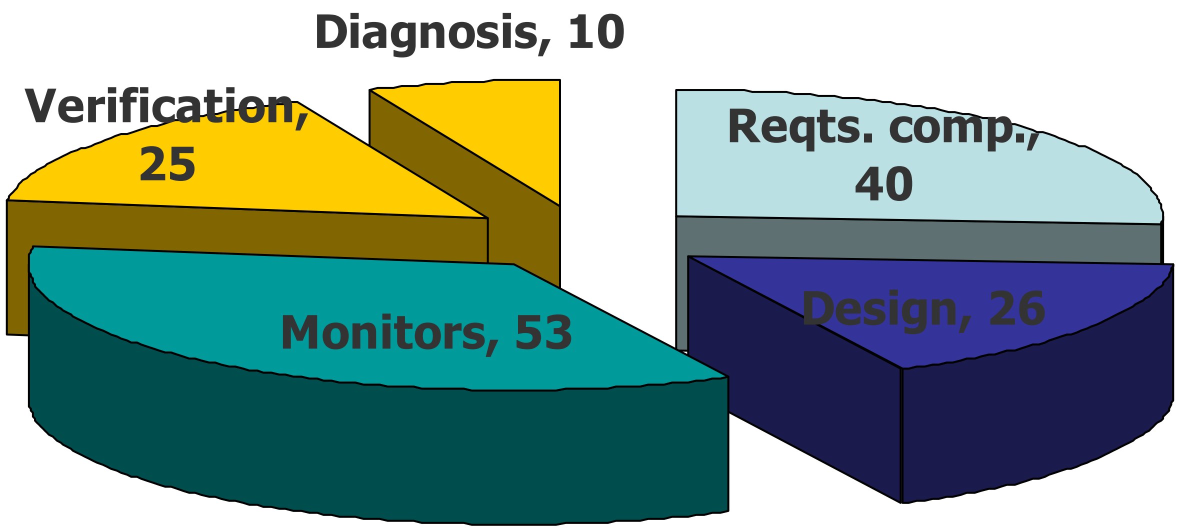
Results

- 62 monitor, 10 design models created
- Verification results
  - 11 inconsistencies in requirements

    "If the handbrake is on turn off the light"

    "If the light switch is on turn on the light"
  - Why?
    - Evolving document
    - Multiple teams
    - "The implementors will know what to do"

Effort (Person-hours)



Discussion

- Requirements modeling
  - First study: 2 hours (1.2% of total) 1 requirement (2 hours / reqt.)
  - Second study: 53 hours (34.4% of total) 62 requirements (50 minutes / reqt.)
- Design model development
  - First study: 80 hours (49.0% of total) Reverse engineering (80 hours / model)
  - Second study: 26 hours (16.9% of total) 10 models (2.6 hours / model)
- Verification
  - First study: 45 min. (0.5% of total) 1 requirement (45 min. / reqt.)
  - Second study: 25 hours (16.2%) 62 requirements (25 min. / reqt.)
- Fault diagnosis
  - First study: 30 min. (0.3% of total) 1 reqt., 1 error (30 min.

Conclusions

- Monitor models formalize requirements efficiently
  - Reference architecture for such models was a big factor
- Reverse engineering design models is very time-consuming
- Automated testing-based checking of requirements uncovered requirements inconsistencies
  - Tests also useful for diagnosing problems
- Effort “up-front” in modeling requirements pays off “downstream” in design
  - Design models easier to construct after requirements models

Discussion (II)

- Results directed toward practical concerns
  - Process
  - Tools
  - Repeatability
  - Artifacts
- There are benefits to formalization even when formal verification is impossible

Ongoing Work

- An IBV-based development process
  - Requirements, then
  - Monitor models, then
  - Design models, then
  - Verification
- Combining non-functional, functional modeling and verification
  - Extract architectural information from Simulink models
  - Apply software architecture analysis tools
- Real-time model checking on instrumented models

Thanks for Your Attention

Rance Cleaveland: rcleaveland@fc-md.umd.edu

Simulink / Stateflow: www.mathworks.com

Reactis: www.reactive-systems.com