HCMDSS/MD PnP, Boston, 26 June 2007

Accidental Systems

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Normal Accidents

- The title of an influential book by Charles Perrow (1984)
- One of the Three Mile Island investigators
  - And a member of recent NRC Study "Software for Dependable Systems: Sufficient Evidence?"

  A sociologist, not a computer scientist
- Posits that sufficiently complex systems can produce accidents without a simple cause
- It's the system that fails
- Perrow identified interactive complexity and tight coupling as important factors

AFTI F16 Flight Test, Flight 36

- Control law problem led to a departure of three seconds duration
- Side air data probe blanked by canard at high AOA
- Wide threshold passed error, different channels took different paths through control laws
- Sideslip exceeded $20°$, normal acceleration exceeded -4g, then $+7$g, angle of attack went to $-10°$, then $+20°$, aircraft rolled $360°$, vertical tail exceeded design load, failure indications from canard hydraulics, and air data sensor
- Pilot recovered, but analysis showed this would cause complete failure of DFCS and reversion to analog backup for several areas of flight envelope

AFTI F16 Flight Test, Flight 44

- Unsynchronized operation, skew, and sensor noise led each channel to declare the others failed
- Simultaneous failure of two channels not anticipated
  So analog backup not selected
- Aircraft flown home on a single digital channel
  (not designed for this)
- No hardware failures had occurred

Analysis: Dale Mackall, NASA Engineer
AFTI F16 Flight Test

- Nearly all failure indications were not due to actual hardware failures, but to design oversights concerning unsynchronized computer operation
- Failures due to lack of understanding of interactions among
  - Air data system
  - Redundancy management software
  - Flight control laws (decision points, thumps, ramp-in/out)

You Think Current Commercial Planes Do Better?

- Fuel emergency on Airbus A340-642, G-VATL, 8 February 2005
  - AAIB SPECIAL Bulletin S1/2005
- In-flight upset event, 240 km north-west of Perth, WA, Boeing 777-200, 9M-MRG, 1 August 2005
  - Australian Transport Safety Bureau reference Mar2007/DOTARS 50165

Interactive Complexity and System Failures

- We are pretty good at building and understanding components
- But systems are about the interactions of components
  - i.e., their emergent behavior
- We are not so good at understanding this
- Many interactions are unintended and unanticipated
- Some are the result of component faults
  - Often multiple and latent
  - And malfunction or unintended function rather than loss of function
- But others are simply due to . . . complexity

Systems and Components

- The FAA certifies airplanes, engines and propellers
- Components are certified only as part of an airplane or engine
- That's because it is not currently understood how to relate the behavior of a component in isolation to its possible behaviors in a system (i.e., in interaction with other components)
- So you have to look at the whole system

Designed and Accidental Systems

- Many systems are created without conscious design
  - By interconnecting separately designed components
  - Or separate systems

  These are accidental systems
- The interconnects produce desired behaviors
  - Most of the time
- But may promote unanticipated interactions
  - Leading to system failures or accidents
- PnP facilitates the construction of accidental systems
  - E.g., blood pressure sensor connected to bed height

The Solution

- Is to discover and control or reduce or eliminate unintended interactions
- It's not known how to do that in general
- In designed, let alone in accidental systems
- But I'll describe some partial techniques

Modes of Interactions

- Among computational components
- Through shared resources (e.g., the network)
- Through the controlled plant (the patient)
- Through human operators
- Through the larger environment

Interactions Among Computational Components

- Computer scientists know how to predict and verify the combined behavior of interacting systems (sometimes)
- E.g., assume/guarantee reasoning
  - If component A guarantees P assuming B ensures Q
  - and component B guarantees Q assuming A ensures P
  - Conclude that $A \,||\, B$ guarantees P and Q

  Looks circular, but it is sound
- Can extend to many components
  - Each treats the totality of all the others as its environment, and ensures its own behavior is a subset of the common environment
- Can be used informally
- Or formally: that is, using formal methods

Aside: Formal Methods

- These are ways of checking whether a property of a computational system holds for all possible executions
- As opposed to testing or simulation
  - These just sample the space of behaviors

  Cf. $x^2 - y^2 = (x - y)(x + y)$ vs. `5*5-3*3 = (5-3)*(5+3)`
- Formal analysis uses automated theorem proving, model checking, static analysis
- Exponential complexity: works best when property is simple
  - E.g., static analysis for runtime errors

  Or computational system is small or abstract
  - E.g., a specification or model rather than C-code

Practical Assume-Guarantee Reasoning

- Develop a model or specification of your component
- And of its assumed environment
  - Cf. controller/plant model in controller design
- The assumed environment can be made part of the component specification
  - Cf. interface automata (IA)
- An IA is more than a list of data types, it's a state machine
- Can automatically synthesize monitors for IAs
- Can formally verify that a collection of components satisfy each others IAs
- Can synthesize the weakest assumptions for which a component achieves specified behavior (IA generation)

Tips To Reduce Interactive Complexity

- Send sensor samples with use-by date rather than timestamp
- For sensor fusion, send intervals rather than point estimates
- Define data wrt. an ontology, not just basic types
  - E.g., raw output of blood pressure sensor vs. corrected for bed height
- Critical things should not depend on less critical
  - E.g., intervention for low blood pressure depends on blood pressure which depends on bed height sensor
  - So now the bed height sensor is as critical as the blood pressure intervention or alarm

Interaction Through Shared Resources

- Cannot get an X-ray to the operating room because the network is clogged with payroll
- Cannot send commands to the ventilator because the blood pressure sensor has gone bad and is babbling on the bus
- Byzantine fault causes devices A and B to have inconsistent estimates of the state of C, so they take inappropriate action
- The user interface gets into a loop and takes all the CPU cycles, so actual device function stops
- Operator entry overflows its buffer and writes into part of memory that affects something else

Partitioning

- Assume-guarantee reasoning about computational interactions relies on there being no paths for interaction other than those intended and considered
- But commodity operating systems and networks provide lots of additional and unintended paths
- Typically, A and B get disrupted because X has gone bad and the system did not contain its fault manifestations
- So safety- and security-critical functions in airplanes, cars, military, nuclear etc. don't use Windows, Ethernet, CAN etc.
- They use operating systems, buses that ensure partitioning
  - IMA: Integrated Modular Avionics
  - MILS: Multiple Independent Levels of Security

  These make the world safe for assume-guarantee reasoning

Partitioning (ctd)

- Partitioning could become COTS with sufficient demand
- But current solutions are Draconian
  - Strict time slicing

  May be too restrictive for medical devices
- Certified to extraordinary levels
  - IMA: failure rate of about 10^{-12}/hour for 16 hours
  - IMA uses DO-178B Level A, which corresponds to CC EAL4
  - High robustness security requires EAL6+ or EAL7

  May be more than needed for medical devices
- Need an adequate partitioning guarantee for dynamic systems

Interaction Through The Controlled Plant

- In medical devices, that's the patient's body
- Device developers probably have controller and plant models
  - Plant model may include only a few physiological parameters
- Different devices have different plant models
  - May be ignorant of the others' parameters
- Yet will interact in actual use
- Obvious perils in normal but unmodeled interactions
- And in the presence of faults
- But also inferior outcomes from lack of beneficial interaction
  - E.g., harmonic relation between heart and breathing rates (Buchman)

Interaction Through The Controlled Plant

- Should have at least a minimal model of the rest of the physiological environment
- And appropriate behavior under all its interactions
- Assumption generation would be cool—might be able to calculate the weakest plant model under which the controller achieves certain properties

Interactions Involving Humans

- As cognitive agents rather than the plant
- Well known that poor human interface design leads to errors
- E.g., Role of Computerized Physician Order Entry Systems in Facilitating Medication Errors, J AMA Vol 293, No. 10 (March 2005), pp. 1197–1203
- Even safety interlocks can introduce errors if the operator does not understand why an action is (not) happening
- E.g., automatic speed protection on A320
  - Causes unexpected mode change, and plane starts climbing when pilots expect it to descend—force fight
- These kinds of problems suggest we may not be able to rely on skilled human intervention once we introduce automation
  - Unless we design it right

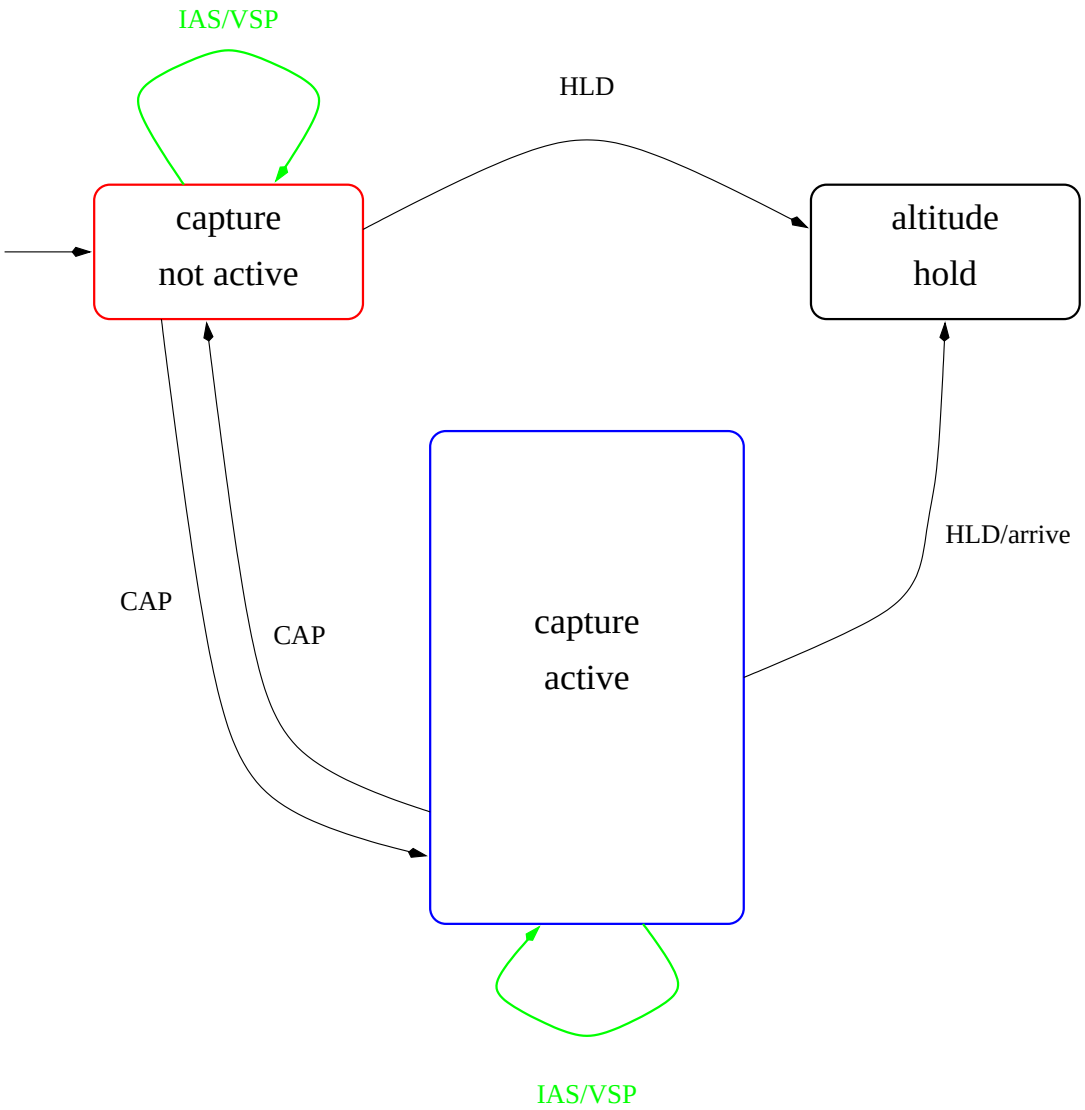
Modeling Mental Models

- Operators use mental models to guide their interaction with automated systems
- Many problems are due to divergence between operator's mental model and actual behavior
- Can represent plausible mental models as state machines
- E.g., use the training manual, then simplify using insights of Denis Javaux
- Then compare all behaviors of the mental model against the actual automation (using model checking)
- Divergences will be likely automation surprises
- Example from MD-88 autopilot

MD-88 Altitude Bust Scenario: Mental Model

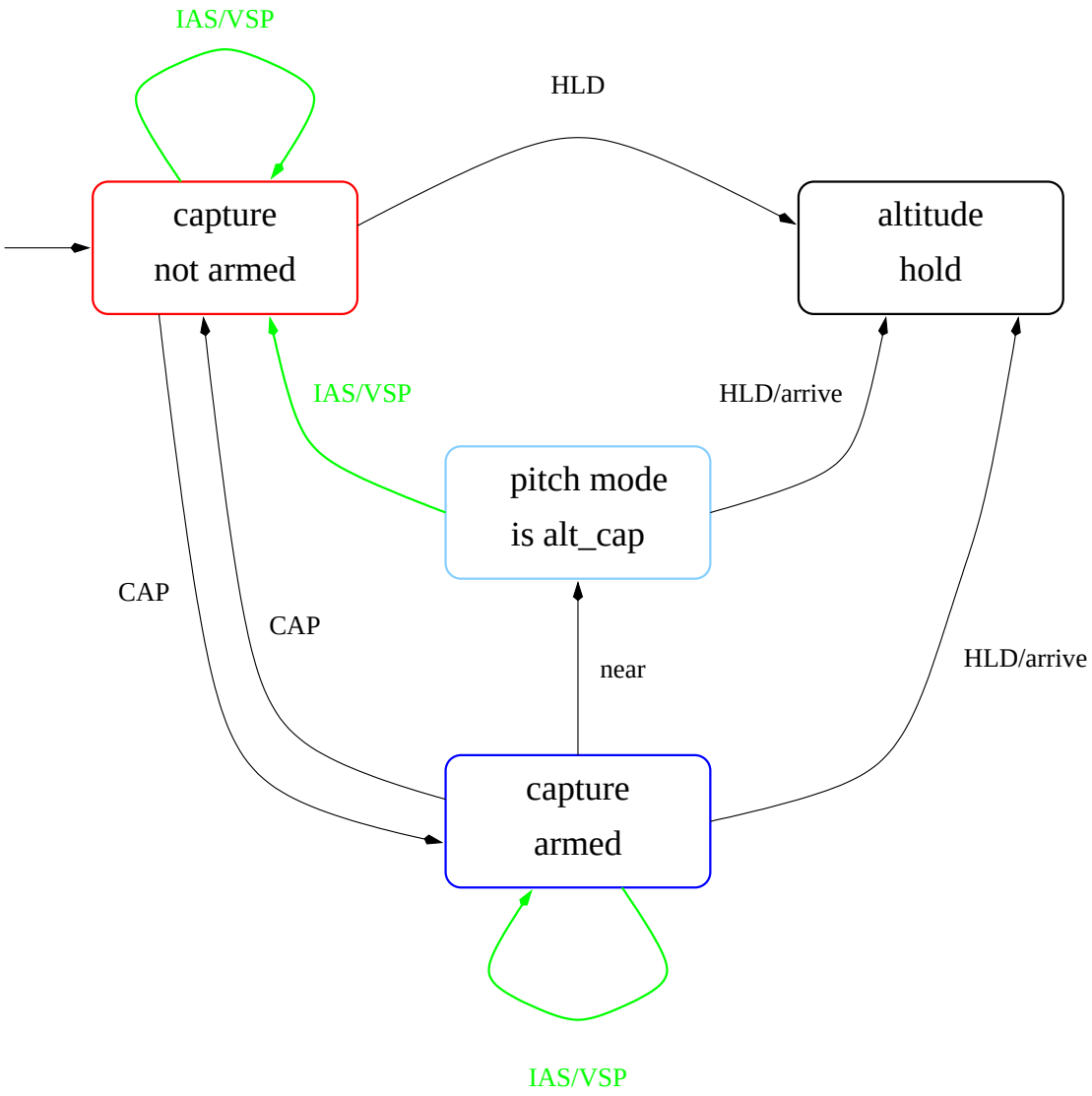
- The pitch modes determine how the plane climbs
  - VSPD: climb at so many feet per minute
  - IAS: climb while maintaining set airspeed
  - ALT HLD: hold current altitude
- The altitude capture mode determines whether there is a limit to the climb
  - If altitude capture is armed
    - ⋆ Plane will climb to set altitude and hold it
    - ⋆ There is also an ALT CAP pitch mode that is used to end the climb smoothly
  - Otherwise
    - ⋆ Plane will keep climbing until pilot stops it

Mental Model



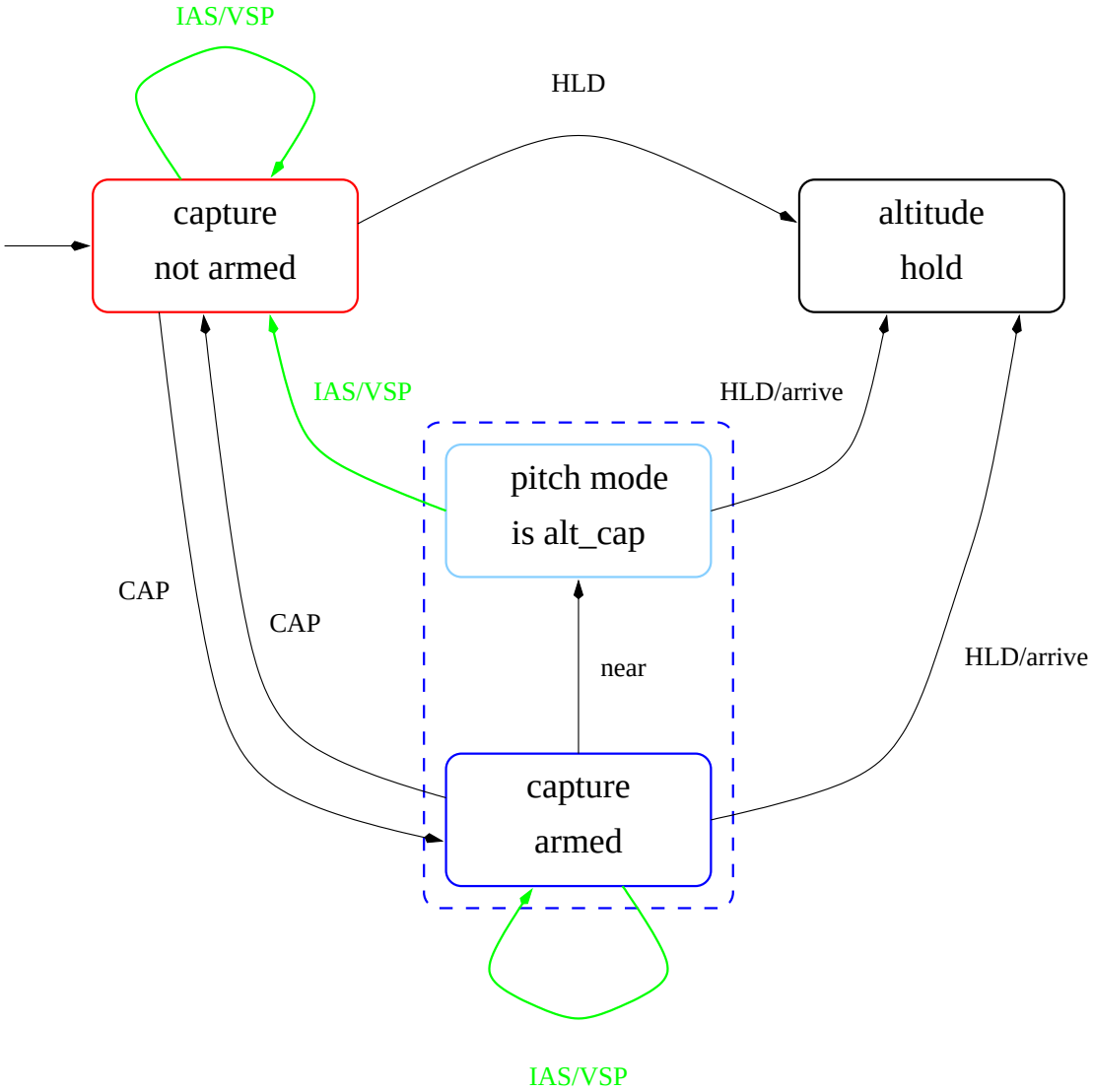
Whether capture is active is independent of the pitch mode

Actual System



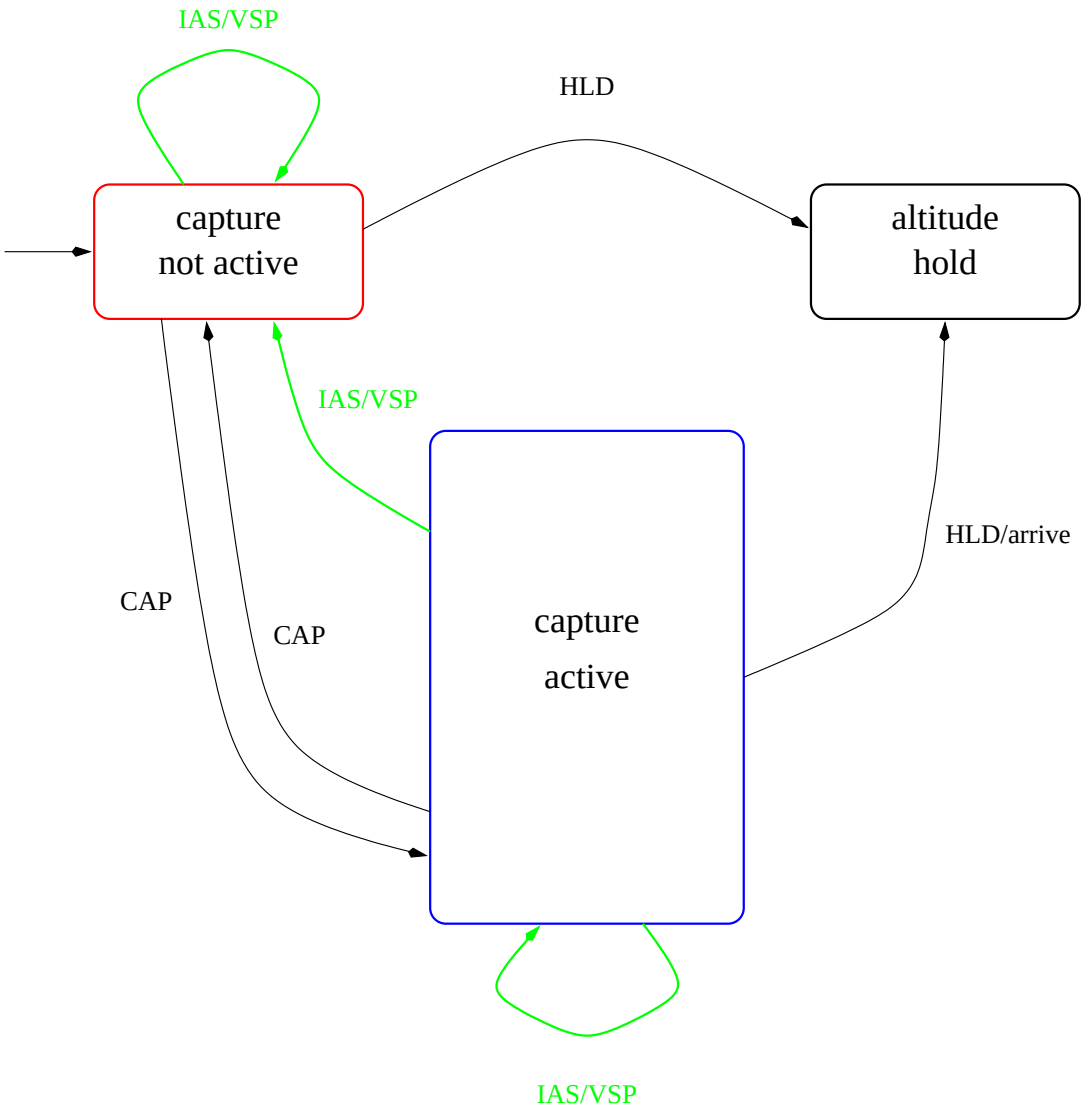
There is an `alt_cap` pitch mode that flies the final capture

Focus (Abstract) on Whether Capture Is Active



Capture is active if it is armed or if pitch mode is `alt_cap`

Abstracted System



Can compare this description directly with the mental model

Interaction Through The Larger Environment

- The purpose of a system is to change some relationships in the environment external to the system
- So requirements specification should focus on those changes
- But changing intended relationships may also change unintended ones
- Requirements engineering should focus on these issues
- E.g., by building models of the environment and exploring interactions
- Model checking and other formal methods allow exploration of all possible behaviors

Socio-Technical Systems

- These are systems that interact with humans or organizations performing complex tasks
- E.g., computer Aided Detection (CAD) tool for interpretation of mammograms
- Improved performance of inexperienced operators with easy-to-detect cancers
- But reduced that of skilled operators in hard-to-detect cases
- I don't know how to predict this kind of thing
- But modern human factors rejects simple failure models for human behavior: there's a range of performance
- The topic of resilient systems explores some of this

Assurance and Certification

- I've described various sources of unintended interactions and suggested some ways to detect and avoid them
- But how do we provide assurance that we've done so?
- All assurance is based on **arguments** that purport to justify certain **claims**, based on documented **evidence**
- There are two approaches to assurance: implicit (standards based), and explicit (goal-based)

The Standards-Based Approach to Software Certification

- E.g., airborne s/w (DO-178B), security (Common Criteria)
- Applicant follows a prescribed method (or processes)
  - Delivers prescribed outputs
    - ⋆ e.g., documented requirements, designs, analyses, tests and outcomes, traceability among these
- Standard usually defines only the evidence to be produced
- The claims and arguments are implicit
- Hence, hard to tell whether given evidence meets the intent
- Works well in fields that are stable or change slowly
  - Can institutionalize lessons learned, best practice
    - ⋆ e.g. evolution of DO-178 from A to B to C
- But less suitable with novel problems, solutions, methods

The Goal-Based Approach to Software Certification

- E.g., air traffic management (CAP670 SW01), UK aircraft
- Applicant develops an assurance case
  - Whose outline form may be specified by standards or regulation (e.g., MOD DefStan 00-56)
  - Makes an explicit set of goals or claims
  - Provides supporting evidence for the claims
  - And arguments that link the evidence to the claims
    - ⋆ Make clear the underlying assumptions and judgments
    - ⋆ Should allow different viewpoints and levels of detail
- The case is evaluated by independent assessors
  - Claims, evidence, argument

What Should the Evidence Look Like?

- Evidence about the process, organization, people
- Evidence about the product

  Reviews: based on human judgment and consensus
  - e.g., requirements inspections, code walkthroughs

  Analysis: can be repeated and checked by others, and potentially by machine
  - Formal methods/static analysis
  - Tests

Multiple Forms of Evidence

- More evidence is required at higher Levels/EALs/SILs
- What's the argument that these deliver increased assurance?
- Generally an implicit appeal to diversity
  - And belief that diverse methods fail independently
  - Not true in n-version software, should be viewed with suspicion here too
- Need to know the arguments supported by each item of evidence, and how they compose
- Want to distinguish rational multi-legged cases from nervous demands for more and more and . . .

A Science of Certification

- Certification is ultimately a judgment that a system is adequately safe/secure/whatever for a given application in a given environment
- But the judgment should be based on as much explicit and credible evidence as possible
- A Science of Certification would be about ways to develop that evidence

Making Certification "More Scientific"

- Favor explicit over implicit approaches
  - i.e., goal-based over standards-based
  - At the very least, expose and examine the claims, arguments and assumptions implicit in standards-based approaches
- Be wary of demands for more and more evidence, with implicit appeal to diversity and independence
  - Instead favor explicit multi-legged cases
  - Use BBNs to combine legs
  - Favor methods that deliver unconditional claims
- Use formal ("machinable") design descriptions
  - Automate safety analysis methods
  - Analyze implementation for preservation of safety

The Challenge of HCMDSS and MD PnP

- For the time being, any device interoperability is likely to be better than none
  - Cf. consumer grade GPS in GenAv cockpits
- But once the low-hanging fruit is taken, you'll start to see system accidents
- So let's develop some effective methods and tools for HCMDSS
  - With a rational goal-based assurance framework
- And an approach to PnP that ensures **system properties**
  - That supports compositional certification

Further Reading

- You can find these on my web page (just Google me)
- NRC Study Software for Dependable Systems: Sufficient Evidence?
- Just-In-Time Certification
- What Use Is Verified Software?
- Bus Architectures for Safety-Critical Embedded Systems (2001)