Beyond Integration:
The Challenge of Compositional Assurance

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA

Assurance and Certification: The Traditional Approach

- The FAA, for example, certifies only
  airplanes, engines and propellers
- The things we care about are system properties
- So certification focuses on systems
- But modern engineering and business practices use massive subcontracting, component-based development, and COTS, so that integrators have less insight than before into subsystem designs
- Strong case for "qualification" of components

  Business case: Component vendors want it (cf. IMA)

  Certification case: Systems-only approach is no longer credible

Compositional Assurance and Certification: The Vision

- Components (and subsystems) are delivered with assurance
    - We'll consider later what that should mean
- Assurance for the system is a calculation based on its design and the assurance of its components
    - Systems are certified without looking inside components
- Notice that steps in this direction would also reduce the integration problem
    - I.e., the problem that you cannot be sure how things will work together based solely on their requirements, specifications, and design documents

Compositional Design and Development

- Compositional assurance will be impossible unless there is a deliberate (and successful!) attempt to control subsystem interactions during design and development
- This is also what is needed for clean integration
- And it is also one of the things needed for safety: cf. Perrow's tight coupling and high interactive complexity
  - Would be manifested through excessively complex mutual assumptions and guarantees
- The alternative is massive testing at every stage, and you still have no guarantee of success

Interfaces and Integration Frameworks

- Components interact through interfaces
- So we need precise specification and assurance for interfaces
  - We'll consider later what that should mean
- And assurance that there are no overlooked interfaces
  - E.g., interaction through the plant
- And assurance that there are no unintended interfaces
  - E.g., interaction through shared resources
  - E.g., interaction due to faults
- The purpose of an integration framework is to eliminate unintended interactions

Integration Frameworks

- Are architectures that guarantee some system-level properties without requiring cooperation from the components they integrate—which may be faulty or actively malicious
- E.g., time and space partitioning in shared processors
    - Architectures for Integrated Modular Avionics (IMA)
    - Separation kernels for security
- E.g., time and space partitioning for shared communications and distributed computation
    - Partitioning Communication System (PCS) for security
        - PCS does CORBA, others do publish-subscribe, or multiplex TCP/IP securely
    - Safety-critical "buses"
        - E.g., Time-Triggered Arch (TTA), FlexRay, SPIDER
- E.g., the MILS architecture for security

Integration (Framework) Anecdotes

Powertrain integration: car engines from one plant, gearboxes from another

- Typically months of work to get them to work together
- A few hours using TTA

Multi-channel FADEC integration: get single channel working, then add second channel

- Typically months of work to get both channels cooperating
- A few hours using TTA

Assurance benefits beyond those in integration

Assurance and Certification

- With integration frameworks we might begin to get a handle on compositional assurance, so let's look at software assurance and certification in a bit more detail
- I'm using assurance to mean the technical judgment that a component or system satisfies some property
- And certification to mean official sanction of some assurance
- In some regimes (e.g., security), judgments whether a system is fit for some purpose are separate from certification of its properties; in others (e.g., civil aircraft) they are combined
- 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
- Internal (DERs) and/or external (NIAP) review
- 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

Critique of Standards-Based Approaches

- Usually define only the evidence to be produced
- The claims and arguments are implicit
- Hence, hard to tell whether given evidence meets the intent
- E.g., use a “safe programming language (subset)”
  - Misra C: no demonstration of effectiveness, some contrary experience (cf. Les Hatton)
  - Coverity, Prefix etc.: strong bug-finding, probabilistic absence of runtime exceptions
  - Spark Ada (with the Examiner): guaranteed absence of run time exceptions
- And the intent (i.e., argument) may not be obvious
- E.g., MC/DC testing
  - Is it evidence for good testing or good requirements?

Do The Standards-Based Approaches Work?

- Fuel emergency on Airbus A340-642, G-VATL, on 8 February 2005 (AAIB SPECIAL Bulletin S1/2005)
- Toward the end of a flight from Hong Kong to London: two engines shut down, crew discovered they were critically low on fuel, declared an emergency, landed at Amsterdam
- Two Fuel Control Monitoring Computers (FCMCs) on this type of airplane; they cross-compare and the "healthiest" one drives the outputs to the data bus
- Both FCMCs had fault indications, and one of them was unable to drive the data bus
- Unfortunately, this one was judged the healthiest and was given control of the bus even though it could not exercise it
- Further backup systems were not invoked because the FCMCs indicated they were not both failed

Safety Culture

- See also incident report for Boeing 777, 9M-MRG
  (Malaysian Airlines, near Perth Australia)
- And several others
- It seems that current development and certification practices may be insufficient in the absence of safety culture
- Current business models are leading to a loss of safety culture
  - Outsourcing, COTS
- Safety culture is implicit knowledge
- Surely, a certification regime should be effective on the basis of its explicit requirements

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
- Generally prefer multiple forms of evidence and their corresponding arguments: multi-legged assurance cases

Formal Methods

- Modern formal methods are automated techniques for calculating properties of software and its (model based) designs and specifications
- Unlike testing, considers all possible execution sequences
- Invariably finds bugs in certified s/w (e.g., DO-178B Level A)
- Tradeoffs between degree of automation, number of false alarms, complexity of the software artifact, and the properties analyzed
- Can do small properties of big programs today: static analysis
    - Absence of runtime errors (Spark Ada Examiner)
    - No loss of arithmetic precision (Astrée for A380)
    - Worst case execution time (AbsInt for A380)
    - Properties of MBD (SCADE for A380)

  These are all European, but the raw technology is better-developed in the USA

Formal Methods (continued)

- Can also do big properties of small systems
  - E.g., protocols, integration frameworks themselves, FDIR

  Maybe a demo?
- And can be used for exploration early in the lifecycle
  - Model-based development makes "machinable" artifacts available in early lifecycle—for the first time

  This is a way to get at requirements
- Formal analysis is repeatable
- New opportunity: formal specification and analysis of interfaces
  - Not just types
    - ⋆ Though extended types would be an advance
  - But the expected behavior (protocol)
    - ⋆ Interface automata

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 . . .

Two Kinds of Uncertainty In Certification

- One kind is failure of a claim, usually stated probabilistically (frequentist interpretation)
    - E.g., 10^{-9} probability of failure per hour, or 10^{-3} probability of failure on demand
- The other kind is failure of the assurance process
    - Seldom made explicit
    - But can be stated in terms of subjective probability
        - E.g., 95% confident this system achieves 10^{-3} probability of failure on demand
        - Note: this does not concern sampling theory and is not a confidence interval
- Multi-legged assurance cases aim at the second of these

Bayesian Belief Nets

- Bayes Theorem is the principle tool for analyzing subjective probabilities
- Allows a prior assessment of probability to be updated by new evidence to yield a rational posterior probability
  - E.g., $P(C \mid E)$ vs. $P(C)$
- Math gets difficult when the models are complex
  - i.e., when we have many conditional probabilities of the form $p(X \mid Q \text{ and } R \text{ or } S)$
- BBNs provide a graphical means to represent these, and tools to automate the calculations
- Can allow principled construction of multi-legged arguments
- Incidentally, philosophers also venture here
  - Confirmation theory: $c(C, E) = P(E \mid C) - P(E \mid \text{not } C)$

BBN Analysis of Multi-Legged Arguments

- Can get surprising results
  - Under some combinations of prior belief, increasing the number of failure-free tests may decrease our confidence in the test oracle rather than increase our confidence in the system reliability
- The anomalies disappear and calculations are simplified if one of the legs in a two-legged case is unconditional
  - Formal methods deliver this kind of claim
- Extends to multiple unconditional claims
- But this analysis assumes formal methods and testing are for checking the same properties: more work needed

Software Assurance in System Safety Cases

- Currently, we apply safety analysis methods (HA, FTA, FMEA etc.) to an informal system description
  - Little automation, but in principle
  - These are abstracted ways to examine all reachable states
- Then, to be sure the implementation does not introduce new hazards, require it exactly matches the analyzed description
  - Hence, DO-178B is about correctness, not safety
- Instead, use a formal system description
  - Then have automated forms of reachability analysis
  - Closer to the implementation, smaller gap to bridge
- Analyze the implementation for preservation of safety, not correctness
  - Favor methods that deliver unconditional claims

Back to Compositional Assurance

- Computer scientists have ways to do compositional verification of programs—e.g., prove
  - Program A guarantees P if environment ensures Q
  - Program B guarantees Q if environment ensures P

  Conclude that $A \,||\, B$ guarantees P and Q
- Assumes programs interact only through explicit computational mechanisms (e.g., shared variables)
- Software and systems can interact through other mechanisms
  - Computational context: shared resources
  - Noncomputational mechanisms: the controlled plant
- So compositional certification is harder than verification

Unintended Interaction Through Shared Resources

- This must not happen
- Need an integration framework (i.e., an architecture) that guarantees composability and compositionality

  Composability: properties of a component are preserved when it is used within a larger system

  Compositionality: properties of a system can be derived from those of its components
- This is what partitioning is about
- Or separation in a MILS security context

Overlooked Interaction Through The Plant

- The notion of interface must be expanded to include assumptions about the noncomputational environment (i.e., the plant)
  - Cf. Ariane V failure (due to differences from Ariane IV)
- Compositional reasoning must take the plant into account (i.e., composition of hybrid systems)
- Must also consider response to failures
  - Avoid domino effect
  - Control number of cases (otherwise exponential)

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

Role For Formal Methods

- The move to model based development presents a (once in a lifetime) opportunity to move analytic methods into the early lifecycle, mostly based on formal methods
- Modern automated formal methods can deliver unconditional claims about small properties very economically
    - Static analysis, model checking, infinite bounded model checking and k-induction using SMT solvers, hybrid abstraction (which uses theorem proving over reals)
- Larger properties will require combined methods
  (cf. the Evidential Tool Bus)
- The applications of formal methods extend beyond verification and refutation (bug finding): test generation, fault tree analysis, human factors,. . .
- Tool diversity may be an alternative to tool qualification

Just-In-Time Certification

- Rather than anticipate all circumstances at design time
- Why not evaluate them at runtime?
  - Maybe with a receding horizon
  - Fewer possibilities to examine, known current state
- Each component makes its model available to others, pursues its own goals while ensuring that possible moves by others cannot trap it into following a bad path, or cause violation of safety
  - Analyzed as a game: guarantee a winning strategy
- Instead of using model checking and other formal methods for analysis, we use them for synthesis
  - Ramage and Wonham: controller synthesis
- Certification would examine the models, trust the synthesis

Summary

- Compositional assurance may not be fully achievable
- But we can vastly increase the use of techniques that support compositional design and assurance
  - Integration frameworks, specification, control and monitoring of interfaces
  - Explicit goal-based assurance cases
  - Automated formal methods
- Would simplify integration
- And probably reduce costs and time