Dagstuhl seminar on Dependability of Component Based Systems, 3–8 November 2002

Based on NASA Langley 12, 13 September 2001

Modular Certification

John Rushby

Computer Science Laboratory

SRI International

Menlo Park CA USA

System Certification

- Dependability, safety, and related attributes are system-level (i.e., emergent) properties
- Hence, certification traditionally considers the full system
    - E.g., in civil air transport, the smallest unit of certification is a complete airplane or engine
- However, there is informal reuse of component-level arguments
- For avionics, this has been aided by the federated architectures employed
    - Each function (autopilot, autothrottle, FMS etc.) has its own fault-tolerant platform
    - Little interaction between them, limited fault propagation

Analysis and certification can focus on component-system interactions, rather than component-component

The Trend Towards Integration

- But avionics systems are now becoming integrated
  - Resources shared between functions
  - Stronger interactions among them

  More functionality at less cost
  - Integrated Modular Avionics (IMA)
  - Modular Aerospace Controls (MAC)
- Similar trend in automobiles
  - Integrated steering, brakes, suspension by wire
- And everything else
- New hazards from fault propagation, and unintended emergent behavior

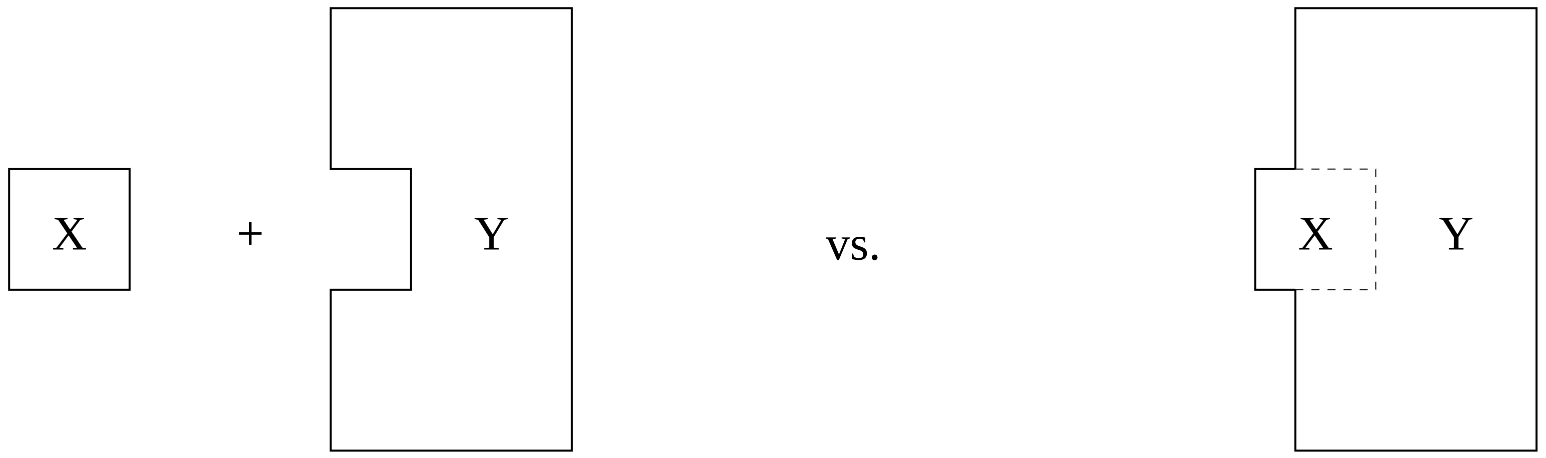
The Trend Towards Components

- Existence of standardized IMA platforms (e.g., TTA)
    - And interfaces (e.g., APEX)

  Encourages development of semi-standardized components (e.g., autopilot or brake-by-wire modules)
- Integrators and customers want these components to be packaged with a certification argument
- Or to come already “pre-certified”
- System certification then largely a composition of component-level pre-certification arguments
- Need a notion of modular certification to support these constructs
- RTCA SC200 and its Eurocae ED60 are holding joint meetings to develop a basis for modular certification

Modular Certification: First Interpretation

- Certification argument for system Y with component/subsystem X makes use of separately certified claims about X's properties at its interfaces
- As opposed to opening up the design of X
- Pictorially:



- Think of X as some onboard function of airplane Y

Modular Certification: The Benefits

- Assuming we have certified X and Y “in isolation”
- The incremental cost of the modular certification X+Y should be less than that of the joint certification $\boxed{XY}$
- Especially if X is reused across many applications
  - Attractive to suppliers of X
- Or if there are many X’s and the owner of Y only has to develop the integration argument
  - Attractive to developer of Y
- Requires that reasoning about properties at the interface is simpler than reasoning about the design

Modular Certification: The Difficulty

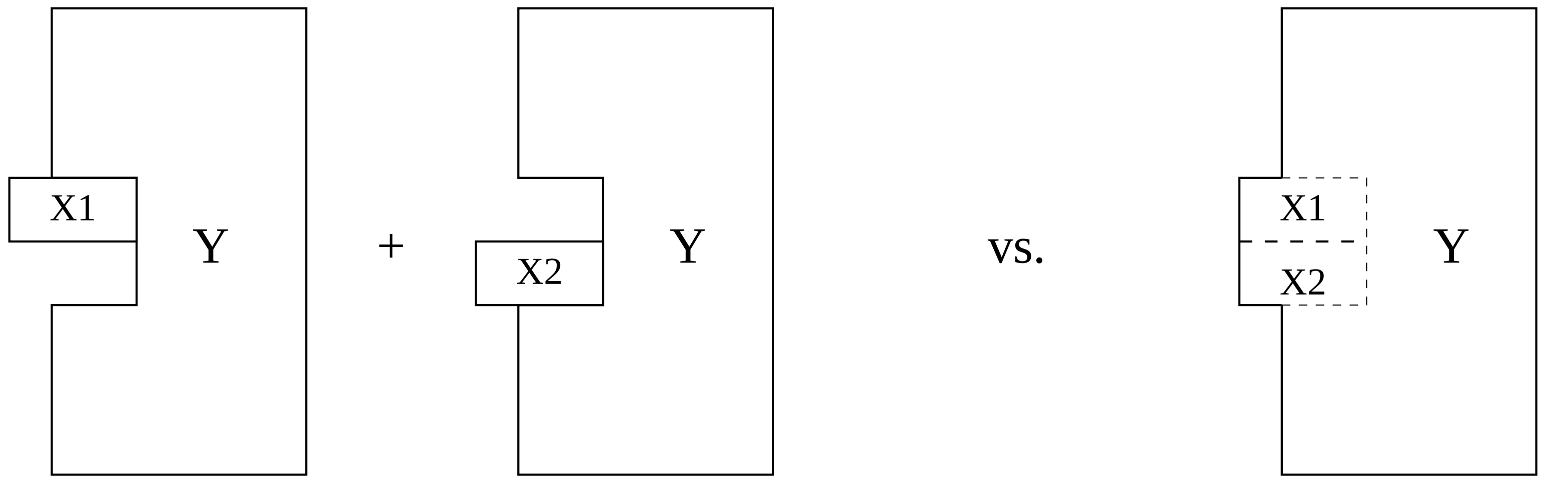
- Much of the effort in certification is not in showing that things work right when all is going well
- But in showing that the system remains safe in the face of hazards
- In the case of X+Y, we have to consider all the hazards that X can pose to Y and vice-versa
- Hazards include malfunction and unintended function as well as loss of function
- Difficult to anticipate all the hazards X might pose to Y without knowing quite a lot about Y, and vice-versa
- Hazards may bypass the traditional notion of “interface”
    - Think of Concorde’s tires

Modular Certification: Need Another Interpretation

- May be infeasible to certify a component X separately from its environment Y
- Difficulty remains when X is software
  - E.g., how can we possibly certify the software for a thrust reverser separate from the reverser?
- This interpretation of modular certification is too ambitious to have useful solutions (yet)
- Though there is interesting work at York and elsewhere on compositional safety analysis

Second Interpretation: Focus On Combinations

- The issue is not to certify X separately from Y
- But given that X_1 and X_2 have both separately been certified with Y
- Establish that X_1 and X_2 can be used together in Y



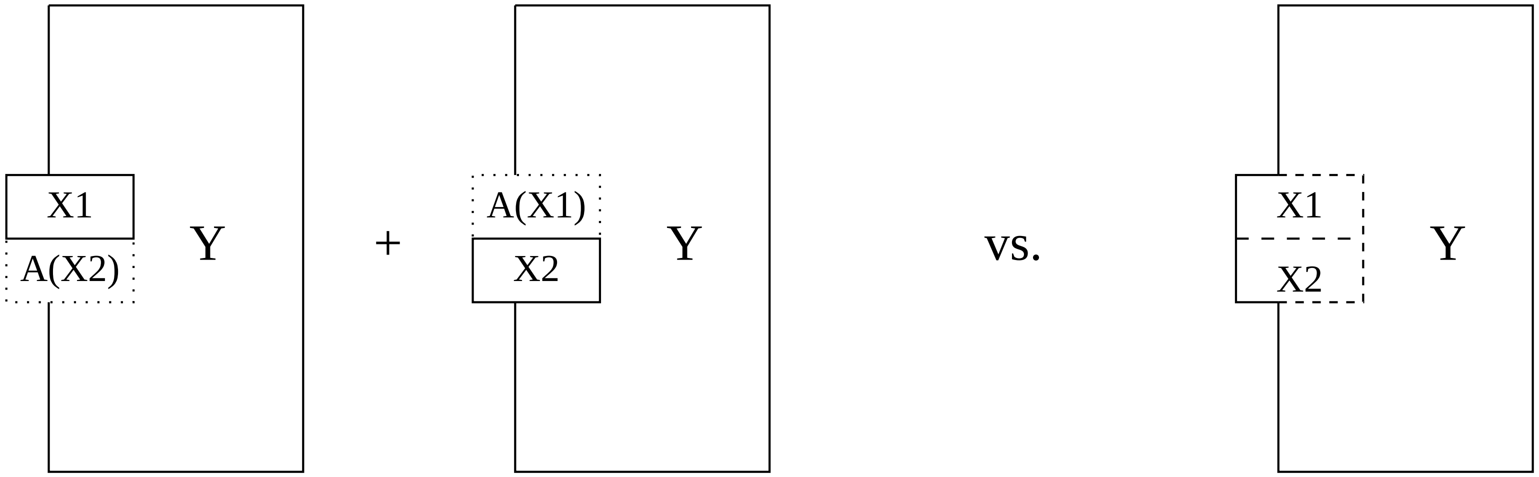
- Still difficult if X_1 and X_2 have strong interactions
  (e.g., engine controller and thrust reverser)

Third Interpretation: Focus On Combinations of Software

- Hazard analysis and the safety case must consider system-component and component-component interaction at the level of the functions concerned
  - E.g., engine controller and thrust reverser
  - No reverse thrust when in flight (system-component)
  - No movement of the reverser doors when thrust above flight idle (component-component)
- The new problem we need to address is the possibility of additional hazards from integration of the software in these function
  - E.g., controller provides thrust data to reverser
  - Both systems share the same computer system
- Want a modular cert'n method that addresses these issues

Modular Certification for Combinations of Software

- Suppose X_1 and X_2 are software for separate functions in Y
  - E.g., X_1 is the software for the engine controller and X_2 is the software for the thrust reverser
- Each is certified for its part in Y on the basis of assumptions about what the other does



- Want to conclude that their actual combination is safe

Modular Certification
By Assume-Guarantee Reasoning

- The idea of verifying properties of one component X_1 on the basis of assumptions about another X_2 (and, in general, X_3, $X_4 \ldots$), and vice-versa is called assume-guarantee reasoning and is fairly well developed in computer science
- But there are two challenges:
  - The approach looks circular, so how do we get a sound method?
  - Assume-guarantee reasoning is used for verification, but we are interested in certification

Modular Certification for Combinations of Software

- Certification differs from verification in that we have to take failures and hazards into account
- Maybe we can split the problem into normal/abnormal cases
    - Can we certify the normal operation of X_1 and X_2 by assume/guarantee methods?
        - ⋆ E.g., the thrust reverser does its thing assuming the engine controller supplies some sensor readings
        - ⋆ Yes, this is similar to other assume/guarantee applications
    - Can we do something similar for the hazards?
        - ⋆ That is, provide guarantees on the behavior of X_1 when X_2 does not fulfill its assumptions?
        - ⋆ This is the hard one!

Assume/Guarantee Reasoning Over Hazards

How can X_2's failure to fulfill its guarantees disturb X_1?

Behavioral failure: X_1 depends on data or control values from X_2 in such a way that its computation becomes hazardous when X_2 fails to satisfy its guarantees

- E.g., X_1 relies on sensor data from X_2 with no independent backup

Interface failure: X_2 affects X_1 through channels other than their defined interface

- E.g., X_2 corrupts or monopolizes resources used by X_1 (memory, communications bandwidth)

Function failure: X_1 cannot guarantee the safety of its function if X_2 allows the system it controls to malfunction

These are the only hazards X_2 can pose to X_1

Controlling The Hazards

- We must control these three classes of hazards
- Some require that X_1 and X_2 have certain properties
- And some require architectural properties of the environment in which X_1 and X_2 operate

The Environment Must Enforce Partitioning

- Interface failure cannot be allowed; X_1, X_2,... must operate within an environment that enforces partitioning
  - Such as the traditional federated architecture
  - Or an IMA or MAC architecture such as SAFEbus or TTA
- Must ensure that no failure of X_2 can affect the basic operation or timing of X_1, nor its communications with nonfaulty X_3, X_4,...
- Top-level requirement specification for partitioning:
  - Behavior perceived by nonfaulty components must be consistent with some behavior of faulty components interacting with it through specified interfaces
- A partitioning architecture must be certified to satisfy this requirement

Normal and Abnormal Assumptions and Guarantees

- In most concurrent programs one component cannot work without the other
    - E.g., in a communications protocol, what can the sender do without a receiver?
- But the software of different aircraft functions must not be so interdependent
    - In fact, X_1 must not depend on X_2
    - In worst case, X_1 must provide safe operation of its function in the absence of any guarantees from X_2
    - Though may need to assume some properties of the function controlled by X_2 (e.g., thrust reverser may not depend on the software in the engine controller, but may depend on engine remaining under control)

Normal and Abnormal Assumptions and Guarantees (ctd)

- In general, X_1 should provide a graduated series of guarantees, contingent on a similar series of assumptions about X_2
  - These can be considered the normal behavior of X_1 and one or more abnormal behaviors
- A component may be subjected to external failures of one or more of the components with which it interacts
  - Recorded in its abnormal assumptions on those components
- A component may also suffer internal failures
  - Documented as its internal fault hypothesis

Components Must Avoid Behavioral and Function Failure

True guarantees: under all combinations of failures consistent with its internal fault hypothesis and abnormal assumptions, the component must be shown to satisfy one or more of its normal or abnormal guarantees.

Safe function: under all combinations of faults consistent with its internal fault hypothesis and abnormal assumptions, the component must be shown to perform its function safely (e.g., if it is an engine controller, it must control the engine safely)

- Where “safely” means behavior consistent with the safety case assumption about the function concerned

Avoiding Domino Failures

- If X_1 suffers a failure that causes its behavior to revert from guarantee $G(X_1)$ to $G'(X_1)$
- May expect that X_2's behavior will revert from $G(X_2)$ to $G'(X_2)$
- Do not want the lowering of X_2's guarantee to cause a further regression of X_1 from $G'(X_1)$ to $G''(X_1)$ and so on

  Controlled failure: there should be no domino effect.

  > Arrange assumptions and guarantees in a hierarchy from 0 (no failure) to j (rock bottom). If all internal faults and all external guarantees are at level i or better, component should deliver its guarantees at level i or better

  This subsumes true guarantees

Technical Aside

- What is a suitable formulation of assume-guarantee reasoning?
- $\langle p \rangle X \langle q \rangle$ asserts that if X is part of a system that satisfies p, then the system must also satisfy q

$$\frac{\begin{array}{l}\langle P_2 \rangle X_1 \langle P_1 \rangle \\ \langle P_1 \rangle X_2 \langle P_2 \rangle\end{array}}{\langle true \rangle X_1 || X_2 \langle P_1 \wedge P_2 \rangle}$$

- Naïve rules like this are unsound

Sound Assume Guarantee Rules

- Consider a specific example: clock synchronization and group membership in TTA
  - Each depends on the other
  - But the circularity is broken by time
  - Membership at time t depends on synchronization at $t - 1$
- Similar time offset in dependency of X_1 in this frame on assumptions about X_2 in previous frame
- This suggests that we could modify our previous circular rule to read as follows, where P_j^t indicates that P_j holds up to time t

$$\frac{\begin{array}{c}\langle P_2^t \rangle X_1 \langle P_1^{t+1} \rangle \\ \langle P_1^t \rangle X_2 \langle P_2^{t+1} \rangle\end{array}}{\langle true \rangle X_1 || X_2 \langle P_1 \wedge P_2 \rangle}$$

A Practical Sound Circular Rule

- The idea is good, but we want a rule that supports systematic reasoning from one time point to the next
- A formulation that has this character has been introduced by McMillan; here H is a "helper" property, $\Box$ is the "always" modality of Linear Temporal Logic (LTL), and $p \rhd q$ means that if p is always true up to time t, then q holds at time $t+1$ (i.e., p fails before q)

$$\frac{\begin{array}{l}\langle H \rangle X_1 \langle P_2 \rhd P_1 \rangle \\ \langle H \rangle X_2 \langle P_1 \rhd P_2 \rangle\end{array}}{\langle H \rangle X_1 || X_2 \langle \Box (P_1 \wedge P_2) \rangle}$$

Soundness and Utility

- Have formally verified this rule in PVS
  (for the 2-process case)
- Notice that $p \rhd q$ can be written as the LTL formula $\neg(p \,\mathsf{U}\, \neg q)$, where U is the LTL "until" operator, so the antecedents can be discharged by model checking in the case of finite systems

Completeness

- McMillan’s rule is sound, but Namjoshi and Trefler show that it is incomplete
- They show that the following extended version of the rule is both sound and complete

$$\frac{\begin{array}{l}\langle H\rangle X_1\langle (A_2 \wedge P_2) \rhd ((A_2 \supset P_1) \wedge A_1)\rangle \\ \langle H\rangle X_2\langle (A_1 \wedge P_1) \rhd ((A_1 \supset P_2) \wedge A_2)\rangle\end{array}}{\langle H\rangle X_1 || X_2\langle \Box (P_1 \wedge P_2)\rangle}$$

  Here A_1 and A_2 are auxiliary properties; by taking both to be *true*, this more complicated rule reduces to the previous one

- Probably not an issue in practice

Summary

- We have seen that modular certification depends on three properties
  - Partitioning
  - Safe function
  - Controlled failure (which subsumes true guarantees)
- And have seen a verified rule for assume-guarantee reasoning that is suitable for this application
- Sheds light on Kopetz recommendation of elementary over composite interfaces
- And on Perrow’s concern over interactive complexity and strong coupling

Future Development

- Need to try it out in practice
  - Simplified engine controller/thrust reverser
  - MAC applications (subfunctions rather than functions)
  - Membership and synchronization in TTA
- Need to extend the formalization of McMillan's rule to the normal/abnormal case and the controlled failure requirement
- Should try to derive specific partitioning requirements from the top-level specification