From Reviews to Analysis: Challenge and Opportunity Converge

John Rushby
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025 USA
Rushby@csl.sri.com

Abstract

The emergence of complex safety-critical systems such as integrated modular avionics challenges the limits of traditional process-based assurance methods based on what DO-178B calls "reviews." The problem is that interaction between subsystems produces too many possible behaviors. Assurance for these cases needs support from what DO-178B calls "analyses"; in particular, tool-supported calculation of (properties of) all possible system behaviors. These calculations can be performed by automated formal methods, but have been infeasible in the past due to inadequate tools and absence of suitable descriptions of the products. Recent advances have produced much more capable tools, and the emergence of "model based" approaches to system development has made suitable product descriptions available.

1. The Challenge

Assurance is about providing evidence that we know the consequences of the system's behavior in every possible circumstance, and that those consequences are acceptable in some appropriate sense. Knowing the consequences of the system's behavior in every possible circumstance does not necessarily mean that we need to have detailed knowledge of its behavior in every possible circumstance: for example, we may not need to know all the ways an engine controller can get into an overspeed situation if there is a separate overspeed protection mechanism that can shut off the fuel. But as systems become more intimately connected to other systems, and are themselves composed of interacting subsystems, so the provision of easily understood protection mechanisms becomes less feasible and it *does* become necessary to comprehend all possible behaviors of the system concerned.

Many process-based certification methods rest on what DO-178B, the guidelines for certification of airborne software [3], calls *reviews*: that is, "processes that depend on human judgment and consensus." Certainly, human judgment is crucial for many grave decisions, but it is a precious resource, and a fallible one. It is particularly fallible when confronted by huge numbers of possibilities—and this is exactly what arises when systems interact. Traditional aircraft systems were "federated," meaning that each "function" (autopilot, autothrottle, yaw damper, etc.) was largely self-contained and had limited interaction with the others. In this architecture, it was sound and feasible to understand each functional subsystem independently, and to comprehend the consequences of their mutual interaction be means of reviews. But as the industry moves toward "Integrated Modular Avionics" (IMA), so the feasibility and trustworthiness of an approach based on reviews becomes questionable.

In an IMA architecture, the "integration" aspect means that functions that were previously federated now interact more intensively and that functions are "deconstructed" into interacting subsystems that share resources; the "modular" aspect means that there is a desire to develop standardized components that can be "qualified" (meaning they are provided with an approved form of "local" assurance) so that system development is accomplished by combining components and assurance is based on a similar combination of their qualification data. Both these aspects pose significant challenges to the traditional process and review-based approaches to assurance.

Some of the more recent guidelines for assurance of aircraft systems and software recognize the new challenges posed by IMA. For example, ARP-4754 is focussed on "Certification Considerations for Highly-Integrated or Complex Aircraft Systems" and states the issue as follows [5, page 12].

> "Highly-integrated and complex systems present greater opportunities for development error (requirements determination and design errors) and undesirable, unintended effects. At the same time it is generally not practical (and may not even be possible) to develop a finite test suite for highly-integrated and complex systems which conclusively demonstrates that there is no development error residue. Since these errors are generally not deterministic and suitable numerical methods for characterizing them are not available, other qualitative means should be used to establish that the system can satisfy safety objectives."

However, the methods for "development assurance" described in ARP-4754 do not differ greatly from those considered standard in the industry: the main stress is on Functional Hazard Assessment, System Safety Analysis, and Common Cause Analysis. While these methods can help structure reviews of complex systems, they do not, to my mind, directly tackle the main challenge to assurance of these systems: the huge number of potential behaviors that become possible when (sub)systems interact, and that can cause "locally correct" subsystems to combine to produce system failure (as in the first launch of Ariane V). In my opinion, a direct assault on this challenge requires us to examine the actual products (i.e., system and subsystem requirements, specifications and designs), and systematically to examine their interactions and collective behavior. Such an examination cannot be based on human review: the huge number of cases to be considered requires the assistance of tools and hence a shift from reviews to what DO-178B calls *analyses*: that is, "processes that can be repeated and checked by others, and potentially so by machine."

The kinds of analyses I have in mind are those based on mechanized formal methods: that is, methods that calculate properties of the behaviors of systems through symbolic analysis of their formal design descriptions (i.e., requirements, specifications, algorithms, and programs). These methods include static analysis, model checking, automated test-case generation, and formal verification. The great advantage of formal methods is that they can, subject to important caveats, consider *all possible* system behaviors (including those of interacting systems).

2. The Opportunity

Automated formal methods have long been advocated, and even accepted in principle (e.g., both DO-178B and DO-254 allow their use), but their uptake in industries such as aerospace has been slow. In part, this can be attributed to inadequate tools (lacking automation and requiring specialized skill), but a bigger obstacle has been lack of suitable artifacts in the traditional design lifecycle to which formal methods of analysis can be applied directly. The traditional lifecycle begins with (possibly several levels of) requirements documents written in natural language, proceeds through (again, possibly several levels of) specification documents also written in natural language, and then makes the leap to program code. None of these are particularly suitable for formal analyses: the requirements and specification documents are not formal, and the actual software code is too large and at too low a level of abstraction. Technology transfer experiments have successfully translated requirements or specification documents into the formal languages supported by various tools and have demonstrated the utility of formal analyses (e.g., [2]), but practitioners have shown little enthusiasm for adopting those languages.

Recently, however, something of a paradigm shift has occurred as industry has adopted methods based on *model-based development* at a surprisingly rapid pace. Model-based development uses a single model of the system (and its environment) as the focus for all design and development activities. The suite of tools from MathWorks is the most widely-used model-based design environment, but there are others based around UML, and SCADE for safety-critical applications. The attractions are that the modeling environments have a strong graphical element (e.g., Simulink/Stateflow in the MathWorks tools), and they provide very effective simulation and visualization capabilities and, more recently, code generation. This encourages engineers to develop models of their systems very early in the lifecycle, and to maintain them throughout the development. The notations employed in model-based development environments (e.g., Stateflow) often have complex and clumsy semantics and may not be what formal methods developers would wish for, but they are able to support formal analyses.

I therefore see a promising convergence: as the complexity of IMA and similar systems overwhelms review-based methods of assurance and creates a need for product-based assurance using automated forms of analysis, so the shift toward model-based development is creating an environment in which it is feasible to deploy effective tools for formal analysis. Among the many remaining challenges are to develop those effective tools (such as bounded model checkers and test case generators for systems specified over mathematically rich domains [1, 6]), and to develop methods for composing formal certification arguments in a modular way [4].

References

[1] L. de Moura, H. Ruess, J. Rushby, and N. Shankar. Embedded deduction with ICS. To be presented at the National Security Agency's Third High Confidence Software and Systems Conference, Apr. 2003. Available at http://www.csl.sri.com/~rushby/abstracts/hcss03. 2

[2] B. L. Di Vito. High-automation proofs for properties of requirements models. *Software Tools for Technology Transfer*, 3(1):20–31, Sept. 2000. 2

[3] Requirements and Technical Concepts for Aviation, Washington, DC. *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, Dec. 1992. 1

[4] J. Rushby. Modular certification. NASA Contractor Report CR-2002-212130, NASA Langley Research Center, Dec. 2002. Available at http://techreports.larc.nasa.gov/ltrs/PDF/2002/cr/NASA-2002-cr212130.pdf. 2

[5] Society of Automotive Engineers. *Aerospace Recommended Practice (ARP) 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*, Nov. 1996. 1

[6] A. Tiwari, J. Rushby, and N. Shankar. Invisible formal methods for embedded control systems. *Proceedings of the IEEE*, 91(1):29–39, Jan. 2003. 2