From the 12th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS), Auckland, New Zealand, July 2007, pp. 15–24; recipient of the “Best Paper” award.

Just-in-Time Certification[*]

John Rushby
Computer Science Laboratory,
SRI International
Menlo Park CA USA
rushby@csl.sri.com

Abstract

Traditional, standards-based approaches to certification are hugely expensive, of questionable credibility when development is outsourced, and a barrier to innovation. This paper is a call and a manifesto for new approaches to certification. We start by advocating a goal-based approach in which unconditional claims delivered by formal methods are combined with other evidence in multi-legged cases supported by Bayesian analysis. We then describe the necessity, and the challenge, of extending this to compositional certification and outline promising directions for accomplishing this. Finally, we consider the provocative possibility of adaptive systems in which methods of analysis traditionally used to support certification at design time are instead used for synthesis and monitoring at runtime, and certification is performed “just-in-time.”

1. Introduction

Certification is intended to provide stakeholders and society at large with assurance that deploying a given system does not pose an unacceptable risk of adverse consequences. Certification is a judgment based on a body of material that, explicitly or implicitly, consists of three elements: *claims*, *evidence*, and *argument*. The claims identify the adverse consequences to be considered and the degree of risk considered tolerable; evidence comprises the results of analyses, reviews, and tests; and the argument makes the case, based on the evidence, that the claims are satisfied.

The traditional approach to software certification may be called “standards based” and largely requires (or strongly recommends) the applicant to follow prescribed processes (e.g., DO-178B [44] for airborne software, the Common Criteria for computer security [8], or IEC-61508 [28] for programmable devices) and to develop specified evidence (e.g., MC/DC tests [23] for DO-178B Level A). In the standards-based approach, the claims and the argument are largely implicit.

In some fields and countries, notably safety-critical systems in Europe, a new approach to certification is emerging that is called “goal based.” In this approach, the applicant develops an “assurance case” in which the claims, evidence, and argument are presented explicitly and are evaluated by the certifying authority or some delegated third party. The exact form of the assurance case is a matter for negotiation by the parties involved, but must generally conform to a given framework (e.g., [59]).

Standards-based certification can be very effective for classes of systems where there is extensive experience to support the efficacy of the prescribed methods, and where there is relatively little innovation, so that one system in the class is very much like another. It is less applicable to novel circumstances and can become a barrier to innovation—not only impeding new kinds of applications, systems, or design and implementation techniques, but also new methods of assurance.

Section 2 advocates a goal-based approach to software certification that builds on recent work on multi-legged assurance cases [5, 33], and on advances in the power of automated formal methods of analysis. Together, I believe these advances put assurance arguments on a firmer foundation and constitute the beginnings of a “science of certification.”

Certification is applied to complete systems, with scrutiny penetrating down into subsystems. However, modern business practices, such as use of COTS, outsourcing, and continuous evolution, lessen the extent to which the system developer and the certifier have full visibility into subsystems, so that a transition to compositional certification of systems based on separately certified components has become urgently desirable. Section 3 examines issues in compositional certification and proposes possible approaches. However, even in this formulation, certification remains a

[*] This research was partially supported by AFRL through a subcontract to Raytheon, by NASA Langley contract NNL06AA07B through a subcontract to ERA Corporation, and by NSF grant CNS-0644783.

design-time activity based on the assumption that the system is fixed and that all circumstances it will encounter can be anticipated and countered in advance. Section 4 examines adaptive systems that configure, assemble, or even synthesize their behavior at runtime and considers the provocative possibility that certification, too, could partly be performed "just-in-time." Section 5 concludes and offers suggestions for further research.

2. A Science of Certification

The conceptual basis for all methods of certification is similar in principle to formal verification: for certification we need to anticipate all possible circumstances that can arise in the interaction of the system with its environment and to show that none of them poses unacceptable risks of adverse consequences; in formal verification we consider all reachable states of the system in interaction with its environment and show that none of them violates desired invariants (i.e., properties specified over the state variables of the system and its environment). The spaces of "all possible circumstances" or "all reachable states" are vast, if not infinite, and so we employ abstraction or approximation to group similar circumstances or states together so that only a feasibly finite number of cases need be considered. Methods of analysis used in safety-critical systems, such as hazard analysis, fault tree analysis (FTA), failure modes and effects analysis (FMEA), and guideword (HAZOP) studies, together with similar methods used in security and other kinds of critical systems, can all be seen as abstracted or approximate means for exploring possible circumstances or reachable states. For example, rather than seek the exact circumstances that can produce unacceptable outcomes, we can instead search for the broader class of circumstances that "could lead to or contribute to an unplanned or undesirable event"—i.e., hazards [14, Appendix 1]—and this is likely to be an easier search, just as formal verification generally becomes easier as invariants are strengthened. FTA, FMEA, and HAZOP can be seen as approximate methods of reachability analysis in which only certain paths are explored.

Safety analysis methods are mostly applied by hand to the design and environment of the system as specified in documents describing its requirements, specifications, and assumptions. These descriptions are mostly informal, but as industry practice moves toward model-based development, so they become increasingly formal and it is feasible that the methods of analysis and abstraction employed in certification can be formalized also, and assisted by automated tools; several authors report significant progress in this direction [6, 29, 39]. The related activities of requirements development and analysis can similarly become mechanically assisted once model-based descriptions are available [38].

The design documents subjected to methods such as hazard analysis are generally quite high level, so we need to be sure that the lower levels of design and implementation do not introduce new risks. In traditional certification practice, this is done by requiring conservative design practices (e.g., no dynamic scheduling) and extensive processes of review and evaluation to show that the detailed design and implementation exactly matches its specification. It is challenging to demonstrate exact compliance between an implementation and its specification, so it is common to require several forms of assurance: for example, conservative design practices, plus reviews, plus testing. Rational choices about how many and which different techniques to employ, and how much of each, are difficult because it is not obvious how the contributions of the different techniques "add up" to reduce uncertainty in the certification judgment. To consider this in more detail, we need first to review uncertainty in certification.

2.1. Uncertainties in Certification

Certification is concerned with risk, which is understood as a combination of the severity of an adverse outcome and its likelihood, and most certification regimes require an inverse relationship between these two measures. In commercial aircraft, for example, catastrophic failure conditions ("those which would prevent continued safe flight and landing") must be "extremely improbable," which means that they must be "so unlikely that they are not anticipated to occur during the entire operational life of all airplanes of one type" [13]. Likelihood of an adverse outcome is naturally expressed and analyzed in terms of probabilities (understood as frequencies of occurrence in the long run), so it is standard to set claims such as 10^{-9} per flight hour for the probability of catastrophic failure in an airborne system,[1] or 10^{-4} for the probability of failure on demand for the primary protection system of a nuclear reactor.

There are actually two kinds of uncertainty in certification: one is the likelihood of adverse outcomes (e.g., the probability of failure on demand), the other concerns the efficacy of the assurance process intended to guarantee the required likelihood. Discussion and analysis of this second concern requires some mathematical framework for reasoning about uncertainty in human judgments. There are several such frameworks (e.g., possibility theory and fuzzy logic, and Dempster-Shafer theory) but probability theory is the best developed and most widely accepted and we can

1 An explanation for this figure can be derived [34, page 37] by considering a fleet of 100 aircraft, each flying 3,000 hours per year over a lifetime of 33 years (thereby accumulating about 10^7 flight-hours). If hazard analysis reveals ten potentially catastrophic failure conditions in each of ten systems, then the "budget" for each is about 10^{-9} if such a condition is not expected to occur in the lifetime of the fleet.

use it to speak, for example, of having 95% confidence[2] that a system achieves 10^{-4} probability of failure on demand. However, whereas uncertainty about adverse outcomes can be understood in terms of frequency of occurrence, uncertainty about the efficacy of assurance processes is best understood in terms of the subjective interpretation of probability. Analyzing how use of several different assurance methods can reduce uncertainty then involves combination of subjective probabilities.

2.2. Multi-Legged Arguments in Certification

Standards-based approaches to certification generally distinguish different levels of criticality (e.g., Levels A to E in DO-178B, Evaluation Assurance Levels (EALs) in the Common Criteria, and Safety Integrity Levels (SILs) in IEC-61508) and require different kinds and amounts of evidence for the different levels. Different levels of criticality might identify, say, 10^{-9}, 10^{-7}, and 10^{-5} as tolerable probabilities of adverse outcomes, but do not generally state goals for confidence in the assurance process itself. When dealing with physical systems, where historical failure data are available for materials and components, and very accurate modeling and realistic testing methods are available for complete systems, it is possible to calculate the probability of system failure and separately to estimate confidence in this calculation. With software intensive systems, however, it is not feasible to predict or measure failure rates accurately beyond about 10^{-3} (for which direct measurement is feasible) and confidence in the assurance process is seldom identified as a separate concern. Instead, the two kinds of uncertainty are conflated and more evidence is required for higher levels of criticality without explicit arguments relating each item of evidence to the kind of uncertainty that it addresses—the implicit argument seems simply to be "more is better," where "more" can refer to more evidence of a particular type, or to more different types of evidence.

Under DO-178B, for example, Level A software (the most critical) requires more testing evidence (namely, MC/DC coverage) than does Level B. However, when static analysis was applied to a variety of Level A and Level B avionics software, significant numbers of anomalies were found (a disturbing finding in itself) and there was no discernible difference in anomaly rates between the two levels [19]. On the one hand, this result is disappointing—it suggests that thorough testing is not as effective as might be hoped[3]—but on the other, it provides grounds for optimism: it suggests there might be some benefit in providing multiple forms of evidence (here, testing plus static analysis).

Most standards-based methods of certification do require multiple forms of evidence. Sometimes these address different facets of the argument, but often they are intended to reinforce each other and to strengthen a particular part of the case—for example, we may offer both testing and code reviews to support the claim that a specification is correctly implemented. The justification for offering or requiring multiple kinds of evidence is typically based on an informal (and often implicit) appeal to diversity. Diversity is the idea, or hope, that different methods (of design, implementation, or analysis) will fail independently and hence their combination should give a multiplicative increase in reliability or confidence. Independence of failures in multiple implementations (as assumed in n-version programming) is viewed skeptically today [30], and its employment in assurance cases should raise similar concerns. A more principled consideration of multiple forms of evidence uses Bayesian Belief Nets (BBNs), which explicitly represent dependence among different items of evidence, to evaluate what are called "multi-legged" assurance cases [5]. Bayes theorem is the principal tool for analyzing subjective probabilities: it allows a prior assessment of probability to be updated by new evidence to yield a rational posterior probability; BBNs allow this computation to be extended to complex models ([16] is an enjoyable introduction). As with all tools, BBNs must be used with skill and care, and can yield surprising results: for example, 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 system reliability [33].

The demand for "more" evidence is one of the reasons why certification is so expensive. Insights from BBNs might allow more rational choices among different methods of analysis and more economy in their application, while also delivering reduced uncertainty in a goal-based certification. This topic is briefly examined in the following section.

2.3. Formal Methods and Multi-Legged Cases

Some forms of evidence confer unconditional claims: for example, suitable static analysis delivers evidence for unconditional absence of run-time exceptions or floating point over/underflow. The static analysis may be flawed, so its evidence is contingent, but the claim that it supports is unconditional (i.e., it is of the form "x% confidence that this

2 Note that this is a statement about confidence in the assurance process; it does not concern sampling theory and is not a confidence interval.

3 Some certification experts have told me that the real benefit of MC/DC is not increased testing but improved requirements: tests are driven from requirements but coverage is measured on code, so MC/DC coverage is possible only if the requirements are extremely detailed. In fact, DO-178B acknowledges that test preparation can be as effective as test execution. It is possible that some avionics suppliers use the same development process, including highly detailed requirements, for both Level A and B code, and the only difference is the additional testing performed for Level A. If this is the case, it may not be surprising that static analysis finds similar anomaly rates.

property holds unconditionally," rather than "y% confidence that this property holds with probability z%"). Littlewood and Wright [33] (see also [32]) show that BBN analysis is simplified, and anomalies disappear (e.g., more failure-free tests always increase confidence in system reliability) when one leg of a two-legged assurance argument is unconditional (they refer to such a leg as providing evidence for a claim of "perfection").

Modern highly automated formal methods, which include static analysis, can provide evidence for many unconditional properties (possibly contingent on assumptions about other properties). Multi-legged arguments based on this kind of evidence "add up" easily: they deliver the conjunction of their individual properties as an unconditional claim.[4] This conjoined claim may then be combined with other forms of evidence, such as testing and reviews, using BBNs. Furthermore, it seems plausible that when several unconditional claims have been discharged by formal methods, then testing can be more constructive and more precisely targeted [18] (e.g., testing can be used to provide assurance for the assumptions used in the formal verification). When testing is targeted, rather than driven by coverage analysis, automated methods of test generation (e.g., [22]) become very effective, thereby reducing its cost. In the limit, test generation and formal analysis can be tightly integrated to achieve more than either alone [21].

Standards such as DO-178B envisage that hazard analysis and other methods of safety assurance are applied to a selected level of the system specification, and that the goal of assurance for the software development process is to establish that the delivered software exactly matches (i.e., is correct with respect to) those specifications. Thus Conmy [9] and Amey and Hilton [2] argue that DO-178B is a software correctness standard, not a system safety standard ("there is no relation of the software to the system hazards, the developer can only state the whole box has been tested to level A") and Ankrum and Kromholz [3] find no clear link between many of the evidence artifacts required by DO-178B and system requirements.

Currently, hazard analysis and other safety assurance techniques are applied to relatively high level and informal specifications because they are performed by hand. Modern formal methods could allow certain strong claims to be analyzed directly on low-level specifications (e.g., hazard analysis could be performed directly on the implementation model); this could reduce the work required to provide evidence that the implementation is correct with respect to the analyzed specification. Furthermore, the goal of a multi-legged argument could be adjusted to show that the implementation is safe (i.e., adds no new hazards) rather than correct, with respect to its specification. This should be both simpler and more useful than standards-based approaches, which (implicitly) focus on correctness.

Overall, a goal-based approach that makes extensive use of unconditional claims delivered by automated formal methods, and that combines these with other evidence to form multi-legged assurance cases supported by Bayesian analysis, could provide a more affordable, more effective, and more scientifically principled approach to certification than present standards-based methods.

3. Compositional Certification

The FAA certifies only airplanes, engines, and propellers; there is no provision for certifying a software component such as an operating system separately from the certification of a specific airplane in which it is used. Recent advisory circulars on reusable software components [15] and guidelines on integrated modular avionics (IMA) [45] make some provision for taking the assurance case for a software component from one airplane certification into another, but they fall a long way short of endorsing a compositional approach to certification in which components can be separately certified and systems using these do not need to reexamine their content.[5]

The reason why the FAA and other certification authorities consider only whole systems is that safety, for example, is a system property, so the system must be considered in its entirety. This point is, of course, true, but does not explain why system certification could not be largely a compositional argument based on separately certified components. The true reasons are surely that safety is not compositional under current approaches to system design and development, and current approaches to compositional reasoning do not extend to issues considered in certification.

Now, computer scientists do have methods for compositional reasoning about correctness (e.g., assume-guarantee methods [11]), so why don't these extend to certification? I know of two reasons. The first is that compositional verification of correctness assumes the integrity of the analyzed components. That is, in verifying a protocol, say, composed of a sender and receiver, verification of the sender will be performed on the basis of assumptions about the receiver. These assumptions will include failure modes, such as the possibility that the receiver may drop or duplicate messages, or even that it will respond unpredictably. But the assumptions will not include the possibility that a failure in the

4 Actually, it is not quite that simple, because the claim delivered by static analysis is generally a weak one, possibly unconnected to the real system requirements; I discuss this in more detail in a companion paper [53].

5 EUROCAE, which jointly developed the guidelines for IMA issued in the USA as [45], is proposing to issue them in Europe with a "technical foreword" by Airbus that essentially repudiates their content; the foreword is hard to interpret, but seems to desire a more compositional approach.

receiver will change the program executed by the sender, modify data in its private memory, or prevent it from running at all. Yet if the both sender and receiver share a processor with inadequate isolation between processes, then it is quite possible for a malfunctioning receiver to write into the program or data memory of the sender, or to monopolize the CPU. Thus, certification can make use of compositional verification only if mechanisms are present that guarantee the integrity of the verified components and their interfaces.

The property that must be guaranteed by these mechanisms is referred to as "robust partitioning" in avionics, and "separation" in security [46, 47]. The mechanisms themselves include operating system kernels, and distributed communication systems; their construction and certification is a specialized activity that can be performed independently of the application systems that they support; see for example the protection profile for separation kernels [26] and consideration of safety-critical bus architectures [48, 50].

The role of partitioning in supporting compositional certification of secure systems is being examined as part of a "protection profile"[6] being developed for component integration in the MILS architecture for secure systems [54].[7] Briefly, partitioning mechanisms must provide *composability* for application software and must themselves be *additively compositional*.

Composability means that properties of subsystems are preserved under composition. Thus, the properties of an application subsystem are unchanged when it is composed with (i.e., runs in the environment provided by) a partitioning mechanism; that is, the partitioning environment does not "get in the way." More subtly, if several subsystems are composed with a partitioning mechanism, then composability ensures that no subsystem can interfere with the properties of another. Hence, composability means that properties of subsystems are both *preserved* and *guaranteed* by partitioning.

Partitioning creates an environment in which application subsystems cannot interfere with one another, but they can cooperate—for example, an air data subsystem can provide airspeed and other sensor data to an autopilot. Partitioning guarantees the integrity of these subsystems, so it might seem that we can now reason about their composition using computer science methods for compositional verification—and this raises the second reason why compositional verification does not extend to certification, which is the possibility of interaction through the controlled plant.

6 Protection Profiles are an element in the Common Criteria for certification of secure systems [8]; they specialize the general criteria for a specific class of systems.

7 MILS is being employed in several DoD applications and a COTS marketplace is developing for MILS components.

Compositional verification considers the control and data exchanged between computational entities. But when we contemplate the interaction between, say, a jet engine and its thrust reverser, we realize that the engine must not deliver more than idle thrust when the reverser doors are unlatched (otherwise they could be blown off). The need for a signal between the controller of the reverser and that of the engine that requests idle thrust cannot be discerned by considering only the computational interactions between them. Hence, requirements development and certification of the thrust reverser must be performed relative to a model of its plant and physical environment, in addition to assumptions about its computational environment. Preliminary consideration of these topics is reported in [49]. Plant models are likely to be hybrid systems (i.e., state machines plus differential equations over real-valued variables), so to support certification, compositional verification needs to be extended from state machines to hybrid systems. Modern methods for formal analysis of hybrid systems, in particular hybrid abstraction [56], hybrid assume-guarantee reasoning [17], and methods for computing invariants of hybrid systems [1], suggest that this is feasible, but additional work is needed to fully develop and adapt these methods for certification.

Another area where classical formal methods for compositional verification need to be adapted for critical systems and their certification concerns systematic avoidance of a cascading "race to the bottom" in the case of component failures. For example, partitioning ensures that a faulty air data subsystem cannot destroy the operation of the autopilot, but the autopilot does use data from the air data subsystem so it cannot be unaffected by its failure. Presumably the autopilot can detect or mask failure of one air data subsystem and has other sources for the sensor data it requires, and so it can still function, but possibly in a degraded manner. This means that compositional reasoning about the air data and autopilot combination is complicated by the need to deal with degraded behaviors; since each component might have m different kinds of degraded behaviors, their composition may need to consider m^2 combinations (and an exponential m^n in the case of n components). When systems mutually interact, degraded behavior by one may force another to drop down to a degraded mode of its own, which may trigger further degradation in the first, and so on. The exponential number of fault cases, and the domino behavior of cascading faults, can both be controlled by imposing a hierarchy of degraded behavior levels. The assume-guarantee model of compositional verification is then elaborated so that assumptions and guarantees distinguish normal and degraded cases, and the degraded cases are hierarchically ordered. Normal guarantees are at level 0, and degraded guarantees are assigned to levels greater than zero. Internal faults are also allocated to levels in a similar manner. The elaborated assume-guarantee reasoning must show that if a component has internal faults at severity level i, and if every component with which it interacts delivers guarantees on level i or better (i.e., numerically lower), then the component delivers a guarantee of level i or better.

Future systems will exist in many configurations (e.g., product families), will be reconfigurable, and will undergo major evolution during their lifetimes (e.g., NASA's Project Constellation which will be developed first to fly in Earth orbit, then to the Moon, and finally to Mars). The current certification process that considers only entire, fixed, and finished systems is not viable for these cases. It is of questionable credibility even for traditional systems such as commercial airplanes, where massive outsourcing, use of commercial off the shelf components and other modern business practices, make it unlikely that even the system integrator, let alone the external certifier, has comprehensive visibility into all the components of the complete system; see for example the recent incident reports [4, 58]. Hence, a compositional certification framework that builds system-level assurance cases from component-level subcases is urgently required, and I hope the approach outlined here points in a promising direction. I suspect that impediments to system safety such as tight coupling and high interactive complexity [40] will be manifested through excessively complex mutual assumptions and guarantees.

4. Runtime Synthesis and Just-In-Time Certification

The main drivers for revisiting the way certification is performed are changes in the time at which a system's configuration and behavior are finally determined. Traditionally, critical systems were developed and assembled as bespoke artifacts and certified as a definitive unit. Compositional certification is a refinement to this approach, which recognizes that systems are now assembled from components that will be used in many different systems; hence, it is attractive to pre-certify the components so that system certification then becomes focused on their integration and the specific configuration and unique attributes of the final system. But the final configuration of many systems is now determined later than the time of design and certification: system instances may configure themselves at installation or load time, and may even reconfigure themselves at runtime.

The software that performs configuration or reconfiguration could be considered "just another program" and certified like all the other software components that constitute the system, but this may overlook a potential difficulty, and an opportunity. The difficulty is that configuration or reconfiguration usually concerns aspects of the basic system architecture: for example, it may define the number of partitions to be created by an operating system kernel, together with their memory and timing attributes, software to be loaded, and the interpartition communication channels. (The configuration data for current airplane IMA systems can be 50,000 lines of XML.) If this is done incorrectly, then many functions and critical properties may be adversely affected and we will have a common-mode failure. Furthermore, there can be many configurations, but any individual system may configure itself only once (at installation time); hence testing can be very difficult.

If the system had only a single configuration, then certification would consider that configuration specifically—directly checking its attributes against requirements or a model. And this is the opportunity: if we could mechanize this check and transfer it to load time or runtime, then it would retain many of the characteristics of certification; in particular, it would be performed against requirements or a model, and it need consider only the single configuration that is about to be installed. We would have achieved "just in time certification."

I believe this scenario can be generalized and applied in many different contexts. The general idea is that certain elements of a conventional certification case can be transferred to runtime. Obviously, these elements must be automated calculations rather than human judgments, and I will focus on those that apply formal analysis (e.g., automated verification) to representations of a software component and its local safety or other critical requirements. I will refer to these representations as "models." As an example, we might have a software function whose own purpose is not critical but that uses a critical resource; the safety requirement could be that its interactions with the resource must follow a certain protocol (so that the resource is not put in an inconsistent state). A design-time certification approach might formally verify that the component follows the protocol, while a just-in-time approach could generate a runtime monitor that blocks any interactions that violate the protocol. The runtime monitor would be synthesized from the model that specifies the protocol using very similar—and equally trustworthy—techniques as those used in formal verification.

Synthesis of monitors and checkers is widely researched and practiced under the name *runtime verification* (see `www.runtime-verification.org`) and should be relatively uncontroversial. However, the usual interpretation has the monitor synthesized at design time and applied at runtime. We can instead imagine adaptive systems where component compositions are created dynamically. In traditional design-time composition, each component has built-in knowledge about the components with which it interacts (e.g., the algorithm of the sender in a communication protocol is designed—and certainly verified—using knowledge about the expected behavior of the receiver). For runtime composition, this knowledge must be acquired dynamically and so I propose that components make explicit models of their behavior and of their requirements for safe operation available to other components. This idea—that components publish a model of their capabilities and that these can be used to check or synthesize component interactions—is already found in restricted forms in the interface automata of de Alfaro and Henzinger [10] and the assumption generation technique of Giannakopoulou, Pasareanu, and Barringer [20]. Interface models could be furnished with runtime monitors, or we could synthesize the monitors (and possibly other functions, such as protocol adapters) at runtime, or whenever the configuration becomes known.

Beyond runtime synthesis of monitors lies synthesis of the component interactions themselves. Service Oriented Architecture (SOA) is an approach to system composition that already does this: components are assembled (using deductive methods) to achieve specified goals based on their published service descriptions [57]. Current realizations of the approach use weak models (little more than descriptions of data formats and ontologies) but could evolve using richer forms of logic and deduction into a certifiable form of self-assembly.

SOA assembles existing components described by static models. Additional opportunities become available when components can adapt their behavior; in these cases, the model published by a component must be dynamic: for example, a component that is operating in a degraded mode will publish a different model than one that is functioning normally. Use of a dynamic model of another component's behavior is seen in recent concepts for aircraft conflict detection and alerting [7]. Current systems, which rely on onboard radar to detect the close approach of another aircraft, are liable to false alarms because the intent of the other aircraft is unknown. Future systems will be able to access the flight plan filed by the other aircraft, or even its current flight management and autopilot parameters, and will use these models to inform their own response.

This example naturally raises concern that an enemy aircraft might file a false flight plan, or broadcast misleading autopilot data. Suitable security and authentication mechanisms should be able to detect and discount the models of masqueraders, but we do need to be concerned that faulty components might publish incorrect models. One approach would be to trust the models only of those components that are certified to have sufficient internal mechanisms (i.e., redundancy and fault tolerance) to publish accurate (even if increasingly weak) models as they degrade, and finally to fail cleanly. Another approach would be to vote the models of multiple redundant components. This can be nicely illustrated in the case of intelligent sensors.

The model published by an intelligent sensor can be an interval (i.e., a pair of real numbers) such that the sampled parameter is guaranteed to lie within that interval if the sensor is nonfaulty. The sensor can adjust the width of its interval according to the dynamics of the sampled process and estimates of its own health (e.g., a narrow interval for a good sample, a wider one for a sample taken under difficult circumstances, and an infinite one to indicate controlled failure). The problem is that a faulty sensor may deliver a false interval (i.e, one on which the actual value of the sampled parameter does not lie in the interval). To overcome this, intervals from multiple sensors can be combined (using Marzullo's fault-tolerant interval construction [37], or the improvement due to Schmid and Schossmaier [55] that maintains the Lipschitz condition) in ways that guarantee a correct consensus interval from $2f + 1$ sensors of which as many as f may be faulty. I suspect that a more robust certification case can be built around this model-based approach than around the complex variations on mid-value-select in current use.

Beyond adaptation lies full behavioral synthesis. In this scenario, the environment of a component is represented by the sum of the models published by the components with which it interacts; each component then strives to discharge its claims, while avoiding behaviors that lead to unacceptable outcomes. This is the idea of *controller synthesis*, introduced by Ramage and Wonham [43]. Here, component models will include attributes of their controlled plant and will typically be hybrid systems. We also need a model of the external environment; this could be built-in (via the interpretation of sensor data) or it could be synthesized or learned by a component specialized to that task (cf. the "health monitoring" components of many current systems) and published along with component models.

Formally, the general synthesis problem is that of constructing a transition system satisfying a given temporal logic property [12, 36]. For an open system, the transition system is given as $M(\vec{x}, \vec{y})$ with inputs $\vec{x}$ satisfying a given property Q. The solution to a synthesis problem has to be formulated in terms of a game where the synthesized system has a strategy for satisfying the temporal property no matter what inputs are given by the environment. Controller synthesis [43] is a specialization in which parts of the system, the plant $T(\vec{r}, \vec{y})$, control inputs $\vec{r}$, and plant state $\vec{y}$, are fixed, and the task is to construct a controller $R(\vec{y}, \vec{r})$ such that $R \| T \models P$. The controller R must be constructed so that for any transition of T, the controller generates a control input $\vec{r}$ that maintains the temporal property P. The set of states with a winning strategy for the controller are those states satisfying P in which for any transition of T, there is a transition of R that leaves the system in a winning state.

Simple instances of the synthesis problem can be reduced to satisfiability—for example, when the system is deterministic with respect to the inputs and the task is to find a sequence of inputs that places the system in some specific state. Such instances include test case generation, scheduling, and planning. In more complicated cases, the controller must really be a strategy that reacts to the environment rather than a simple sequence or a schedule. In this situation, the controller synthesis problem can be solved using techniques based on model checking [42]. The basic paradigm can be extended to real-time [41] and hybrid systems [24, 35].

Calculation of a winning strategy requires a search over a large space. Hence, the computational cost of solving a synthesis problem is formidable, but is made feasible by advances in mechanized deduction (notably, SMT solvers [52], and planning, model checking, and verification techniques based on these), and the power of modern processors. Time-constrained ("anytime") alternatives to full synthesis include selection from a number of preplanned strategies, or limited search that seeks improvement on a safe default strategy. In robot planning, rather than synthesize a true controller that can cope with arbitrary moves by the environment (a "contingent plan"), it has been found effective to build a simple schedule that assumes cooperative behavior by the environment, to monitor its execution, and then replan if the environment departs from expectations [27].

Notice that tools, or techniques, formerly used for verification, such as model checkers, are here being used for synthesis and monitoring. Certification can build on this: we trust these techniques in the analysis of safety at design time, so why not trust them to synthesize and/or monitor safety at runtime? In fact, runtime methods could be more credible than their design-time ancestors—for at design time we must anticipate all possible future states of the system, whereas in controller synthesis we need only consider those states reachable from the current state, and possibly only so far into the future (a "receding horizon").

Synthesis is driven by the models made available by interacting components, and these—or their methods of construction—need to be analyzed at design time. We will need to demonstrate that each component publishes models that are consistent with its actual behavior, given assumptions about the models of the components with which it interacts, that the delivered and assumed models are consistent across interacting components, and that the composite behavior of the interacting models satisfies the required safety (or other certification) claims. These analyses all concern models; the actual implementation is synthesized at runtime and its certification is a consequence of sound synthesis operating on sound models. We could imagine the synthesis generating a certificate, rather as some theorem proving techniques can generate an independently checkable "proof object." Such a certificate would truly be "just-in-time certification."

5. Conclusions and Suggestions For Future Research

New methods are urgently needed for effective—and cost-effective—certification of modern systems whose innovations in function, design, or construction present challenges to traditional methods. I have outlined a "scientific" approach to certification that integrates several ideas developed by others, and suggested how it can be extended to compositional certification of component-based systems. I have also advocated consideration of systems whose behavior is partially synthesized at runtime and argued that sound monitoring or synthesis from verified models can support "just-in-time" certification.

A large benefit of the just-in-time approach is that it meshes well with modern ideas of system extent: the idea that the system boundary does not end with the hardware, but extends into the human and societal fabric in which it is located. Many failures are now attributed to the whole system rather than its components [31] and to poorly engineered human interactions (at both individual and societal level), and a failure to anticipate how introduction of a new system changes the organizational context in which it is located [40]. A whole topic of "resilience" has emerged to focus on these topics [25].

Future work should explore these connections and opportunities (including the possibility of including explicit representations of human "mental models" [51] as part of the modeled environment). Industrial application of "scientific certification" requires powerful formal analysis tools integrated into engineering environments such as Matlab and AADL, while compositional certification requires some worked examples and powerful automation for compositional verification over hybrid systems. Runtime synthesis and just-in-time certification require more scalable methods for controller synthesis. Almost all applications of automated formal methods will benefit from more effective techniques for invariant generation.

Acknowledgments. I am grateful to Robin Bloomfield and Bev Littlewood and their colleagues for introducing me to several of the topics discussed here during a visit to CSR at City University in November 2006. Discussions with my colleagues Rance DeLong, Shankar, and Brian Williams were very influential, as were several stimulating meetings organized by the High Confidence Systems and Software working group of the NITRD. Suggestions by César Muñoz of NIA and the anonymous referees helped improve the presentation.

References

[1] A. Abate and A. Tiwari. Box invariance of hybrid and switched systems. In *2nd IFAC Conf. on Analysis and De-*

sign of Hybrid Systems, ADHS, pages 359–364, 2006.

[2] P. Amey and A. J. Hilton. Practical experiences of safety- and security-critical technologies. *Ada User Journal*, 22(1), Mar. 2001.

[3] T. S. Ankrum and A. H. Kromholz. Structured assurance cases: Three common standards. In *High-Assurance Systems Engineering Symposium (HASE'05)*, Heidelberg, Germany, Mar. 2005. IEEE Computer Society.

[4] Australian Transport Safety Bureau. *In-flight upset event, 240 km north-west of Perth, WA, Boeing Company 777-200, 9M-MRG, 1 August 2005*, Mar. 2007. Reference number Mar2007/DOTARS 50165, available at `http://www.atsb.gov.au/publications/investigation_reports/2005/AAIR/aair200503722.aspx`.

[5] R. Bloomfield and B. Littlewood. Multi-legged arguments: The impact of diversity upon confidence in dependability arguments. In *The International Conference on Dependable Systems and Networks*, pages 25–34, San Francisco, CA, June 2003. IEEE Computer Society.

[6] M. Bozzano and A. Villafiorita. Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform. In SAFECOMP *2003*, number 2788 in Lecture Notes in Computer Science, pages 49–62, Edinburgh, Scotland, Sept. 2003.

[7] V. Carreño and C. Muñoz. Implicit intent information for conflict detection and alerting. In *23rd Digital Avionics Systems Conference*, Salt Lake City, UT, 2004.

[8] *Common Criteria for Information Technology Security Evaluation*, Jan. 2004. Version 2.2, CCIMB-2004-01-001, 002, 003.

[9] P. Conmy. *Safety Analysis of Computer Resource Management Software*. PhD thesis, Department of Computer Science, University of York, York, UK, 2005.

[10] L. de Alfaro and T. A. Henzinger. Interface automata. In *9th Annual Symposium on Foundations of Software Engineering (FSE)*, pages 109–120. ACM Press, 2001.

[11] W.-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference (Revised lectures from International Symposium COMPOS'97)*, volume 1536 of *Lecture Notes in Computer Science*, Bad Malente, Germany, Sept. 1997.

[12] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[13] Federal Aviation Administration. *System Design and Analysis*, June 21, 1988. Advisory Circular 25.1309-1A.

[14] Federal Aviation Administration. *Order 8040.4: Safety Risk Management*, June 1998. Available at `http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/media/app_g_1200.PDF`.

[15] Federal Aviation Administration. *Reusable Software Components*, Dec. 7, 2004. Advisory Circular 20-148.

[16] N. Fenton and M. Neil. The jury observation fallacy and the use of Bayesian networks to present probabilistic legal arguments. *Mathematics Today (Bulletin of the IMA)*, 36(6):180–187, June 2000. Available at `http://www.dcs.qmul.ac.uk/~norman/papers/jury_fallacy.pdf`.

[17] G. Frehse, Z. Han, and B. Krogh. Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In *43rd IEEE Conference on Decision and Control (CDC 2004)*, volume 1, pages 479–484, Atlantic, Bahamas, Dec. 2004.

[18] A. Galloway, R. F. Paige, N. J. Tudor, R. A. Weaver, I. Toyn, and J. McDermid. Proof vs. testing in the context of safety standards. In *24th AIAA/IEEE Digital Avionics Systems Conference*, volume 2, Washington, DC, Oct. 2005.

[19] A. German. Software static code analysis lessons learned. *Crosstalk*, Nov. 2003. Available at `http://www.stsc.hill.af.mil/crosstalk/2003/11/0311German.html`.

[20] D. Giannakopoulou, C. Pasareanu, and H. Barringer. Assumption generation for software component verification. In *Automated Software Engineering*, 2002.

[21] B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani. Synergy: A new algorithm for property checking. In *14th Annual Symposium on Foundations of Software Engineering (FSE)*, pages 117–127, Portland, OR, Nov. 2006. ACM Press.

[22] G. Hamon, L. de Moura, and J. Rushby. Generating efficient test sets with a model checker. In *2nd International Conference on Software Engineering and Formal Methods (SEFM)*, pages 261–270, Beijing, China, Sept. 2004. IEEE Computer Society.

[23] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson. A practical tutorial on modified condition/decision coverage. NASA Technical Memorandum TM-2001-210876, NASA Langley Research Center, Hampton, VA, May 2001. Available at `http://www.faa.gov/certification/aircraft/av-info/software/Research/MCDC\%20Tutorial.pdf`.

[24] T. A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *International Conference on Concurrency Theory*, pages 320–335, 1999.

[25] E. Hollnagel, D. D. Woods, and N. Leveson, editors. *Resilience Engineering*. Ashgate, 2005.

[26] Information Assurance Directorate, National Security Agency, Fort George G. Meade, MD 20755-6000. *U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness*, July 2004. Version 0.621.

[27] F. Ingrand and F. Py. Online execution control checking for autonomous systems. In *The 7th International Conference on Intelligent Autonomous Systems (IAS-7)*, Marina del Rey, CA, March 2002.

[28] International Electrotechnical Commission,, Geneva, Switzerland. *IEC 61508—Functional Safety of Electrical/Electronic/Programmable Electronic safety-related systems*, Mar. 2004. Seven volumes; see `http://www.iec.ch/zone/fsafety/fsafety_entry.htm`.

[29] A. Joshi, S. Miller, M. Whalen, and M. Heimdahl. A proposal for model-based safety analysis. In *24th AIAA/IEEE Digital Avionics Systems Conference*, volume 2, Washington, DC, Oct. 2005.

[30] J. C. Knight and N. G. Leveson. An empirical study of failure probabilities in multi-version software. In *Fault Tolerant Computing Symposium 16*, pages 165–170, Vienna, Austria, July 1986. IEEE Computer Society.

[31] N. Leveson. A new accident model for engineering safer systems. *Safety Science*, 42(4):237–270, Apr. 2004.

[32] B. Littlewood. The use of proof in diversity arguments. *IEEE Transactions on Software Engineering*, 26(10):1022–1023, Oct. 2000.

[33] B. Littlewood and D. Wright. The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN analysis of an idealised example. *IEEE Transactions on Software Engineering*, 33(5):347–365, May 2007.

[34] E. Lloyd and W. Tye. *Systematic Safety: Safety Assessment of Aircraft Systems*. Civil Aviation Authority, London, England, 1982. Reprinted 1992.

[35] J. Lygeros, C. Tomlin, and S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3), March 1999.

[36] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 6(1):68–93, 1984.

[37] K. Marzullo. Tolerating failures of continuous-valued sensors. *ACM Trans. Comput. Syst.*, 8(4):284–304, Nov. 1990.

[38] S. P. Miller, A. C. Tribble, and M. P. E. Heimdahl. Proving the shalls. In *International Symposium of Formal Methods Europe, FME 2003*, volume 2805 of *Lecture Notes in Computer Science*, pages 75–93, Pisa, Italy, Mar. 2001.

[39] F. Ortmeier, W. Reif, and G. Schellhorn. Formal safety analysis of a radio-based railroad crossing using deductive cause-consequence analysis (DCCA). In *5th European Dependable Computing Conference (EDDC)*, number 3463 in Lecture Notes in Computer Science, pages 210–224, Budapest, Hungary, 2005.

[40] C. Perrow. *Normal Accidents: Living with High Risk Technologies*. Basic Books, New York, NY, 1984.

[41] A. Pnueli, E. Asarin, O. Maler, and J. Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control*. Elsevier, 1998.

[42] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL)*, pages 179–190, New York, NY, USA, 1989.

[43] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, Jan. 1989.

[44] Requirements and Technical Concepts for Aviation, Washington, DC. *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, Dec. 1992. This document is known as EUROCAE ED-12B in Europe.

[45] Requirements and Technical Concepts for Aviation, Washington, DC. *DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*, Nov. 2005.

[46] J. Rushby. Critical system properties: Survey and taxonomy. *Reliability Engineering and System Safety*, 43(2):189–219, 1994.

[47] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Available at `http://www.csl.sri.com/~rushby/abstracts/partitioning`.

[48] J. Rushby. Bus architectures for safety-critical embedded systems. In *EMSOFT 2001: Proceedings of the First Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323, Lake Tahoe, CA, Oct. 2001.

[49] 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`.

[50] J. Rushby. An overview of formal verification for the time-triggered architecture. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 2469 of *Lecture Notes in Computer Science*, pages 83–105, Oldenburg, Germany, Sept. 2002.

[51] J. Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, Feb. 2002. Available at `http://www.csl.sri.com/users/rushby/abstracts/ress02`.

[52] J. Rushby. Harnessing disruptive innovation in formal verification. In *4th International Conference on Software Engineering and Formal Methods (SEFM)*, pages 21–28, Pune, India, Sept. 2006. IEEE Computer Society.

[53] J. Rushby. What use is verified software? In *12th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, pages 270–276, Auckland, New Zealand, July 2007. IEEE Computer Society. Available at `http://www.csl.sri.com/~rushby/abstracts/iceccs07-vsi`.

[54] J. Rushby and R. DeLong. *Toward an Integration Protection Profile for MILS*. Computer Science Laboratory, SRI International, Menlo Park, CA, 2007. To appear.

[55] U. Schmid and K. Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing*, 14(2):101–111, May 2001.

[56] A. Tiwari. Abstractions for hybrid systems. *Formal Methods in Systems Design*, 2007. To appear, available at `http://www.csl.sri.com/~tiwari/new.pdf`.

[57] W. Tsai. Service-oriented system engineering: a new paradigm. In *IEEE International Workshop on Service-Oriented Systems Engineering (SOSE) 2005*, pages 3–6, Oct. 2005.

[58] UK Air Investigations Branch. *AAIB Special Bulletin S1/2005: Airbus A340-642, G-VATL*, 2005. Available at `http://www.aaib.dft.gov.uk/cms_resources/G-VATL_Special_Bulletin1.pdf`.

[59] UK Ministry of Defence. *Interim Defence Standard 00-56, Issue 3: Safety Management Requirements for Defence Systems. Part 2: Guidance on Establishing a Means of Complying with Part 1*, Dec. 2004. Available at `http://www.dstan.mod.uk/data/00/056/02000300.pdf`.