Provably Dependable Software Architectures

Victoria Stavridou and R. A. Riemenschneider

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
Email: {victoria,rar}@csl.sri.com

Abstract

Dependable architectures demonstrably possess properties such as safety, security, and fault tolerance. We are interested in developing methods allowing formal demonstrations through proof that an architecture does indeed possess the desired dependability properties. We focus on architecture hierarchies as a means of enabling such demonstrations. We pose a challenge problem for dependable software architectures and we propose a research agenda for solving it.

1 What are dependable software architectures?

Software architectures describing software products that are used to implement critical functions must be trustworthy. *Dependability* is the property of a computing system which allows reliance to be justifiably placed on the service it delivers. The service delivered by a system is its behavior as it is perceived by its users. Dependability is a qualitative judgment about a system.

The software architecture community has made great strides toward characterizing and capturing system descriptions appropriately and toward providing linguistic support for defining families of software products, but current ADLs and their associated methodologies do not adequately address dependability. When software products are deployed in a high-integrity system, their dependability profile is key to the survivability of the system. It is desirable to have developers of critical systems also benefit from software architecture technology. For example, an autopilot software producer would like to be able to derive, from a single abstract specification of a dependable software architecture, implementations for a number of aircraft variants. The derivation process would be supported by tools for applying dependability-preserving transformations that make abstract specifications more concrete. So, these implementations would be known to share dependability attributes as well as functionality. Such dependability-preserving transformations are key not only for effective and timely resource development, but also for assurance and certification. Enriching software architecture descriptions by including dependability attributes will enable and facilitate the reuse of not only software components but certification data *as well*. In critical systems, the cost of assurance and certification is comparable with the development cost.

Dependable architectures demonstrably possess properties such as safety, security, and fault tolerance. It is our objective to produce methods allowing formal demonstrations through proof that an architecture does indeed possess the specified dependability properties.

One important limitation on the utility of applying typical formal methods to reasoning about architectures is the purely informal connection between the mathematical models that are analyzed and the system being modeled. If formal analysis of a model reveals the presence of a flaw, it is generally easy to determine whether that flaw is present in the implemented system. But it is highly desirable to use formal methods to establish that the software is free of certain types of flaws, such as failing to meet safety, security, or fault tolerance objectives. Security properties typically state that certain sorts of error cannot occur. For example, a security property might state that certain sorts of communication within the system — such as flow of restricted information to a component without adequate clearances — cannot occur. Tools supporting various formal methods can be used to prove that any correct implementation of the abstract mathematical model has these properties. However, there is no guarantee that the actual software correctly implements the model. The problem of gaining confidence in the correctness of the implementation is especially acute in the case of dynamic, dependable architectures, where exhaustive testing of architectural configurations is frequently prohibitively expensive, when not theoretically infeasible.

2 IMA: a challenge problem for dependable software architectures

A typical example of a system architecture that needs to be ultra-dependable, is that of *integrated avionics architectures* such as the IMA (Integrated Modular Avionics) concept [4] in civil aviation and the JIAWG Joint Integrated Architecture Working Group architecture [1] in the military. The IMA platform, shown in Fig. 1, houses microprocessors (possibly together with a few ASICs) in a cabinet forming a subsystem with a common chassis design, common fault tolerant processing, redundant power supplies, and flexible aircraft interfaces. Several of these cabinets are interconnected with special-purpose data buses which also connect avionics hardware and other peripherals out-

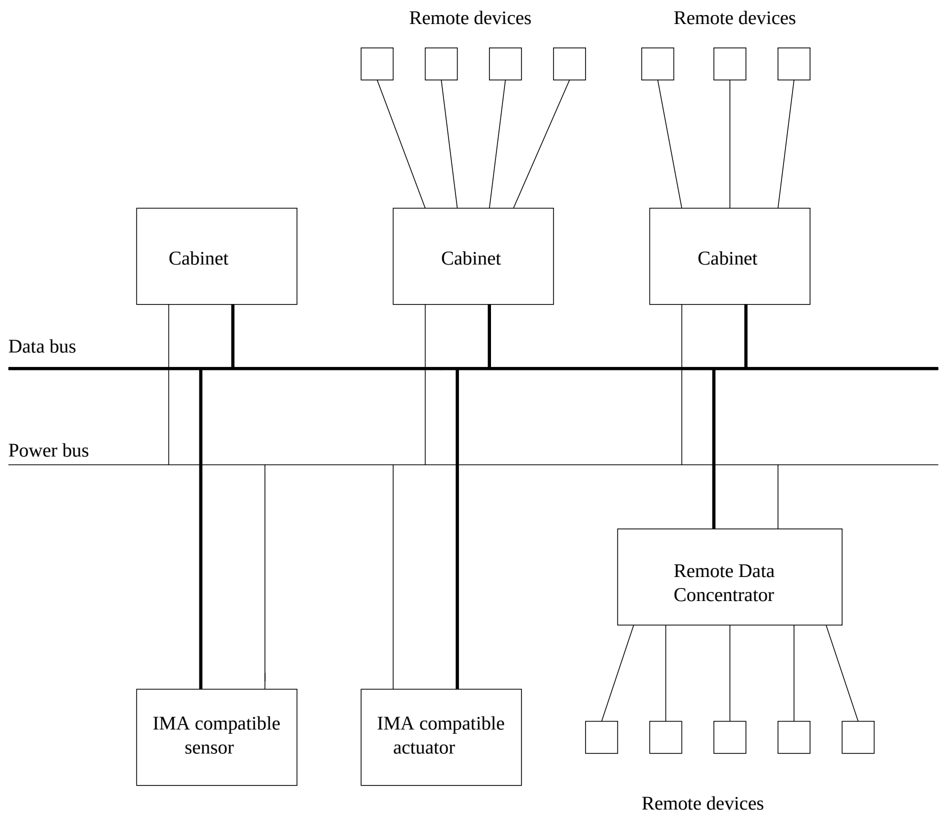

Figure 1: Integrated Modular Avionics (IMA)

side the cabinet to form an integrated system for performing all the avionics functions on the aircraft. Thus, as well as reduced ownership cost, integrated avionics architectures offer unprecedented generality and on-board computing resource utilization to the degree that dynamic, in-flight software reconfiguration across the architecture to optimize safety and mission characteristics becomes a distinct, if distant, vision. The most formidable obstacle to realizing this vision is the need to demonstrate that a centralized architecture failure cannot bring about simultaneous loss of functions utilizing shared resources. Fault tolerance is applied to overcome this problem, primarily through the provision of redundancy and high-integrity monitoring. A more subtle issue, and one that is a significant impediment to the certification and deployment of integrated avionics, is the fact that mixed criticality functions are processed in one cabinet. This allows potentially undesirable interactions between separate avionics functions. The current lack of technology allowing convincing demonstrations of safety, fault tolerance, and noninterference is delaying integrated avionics deployment by 15-20 years in military aviation and undoubtedly many more years in the civil sector.

3 Architecture hierarchies

It is very common to describe a single architecture, or related class of architectures, at multiple levels of abstraction and from a variety of perspectives. The advantage of doing so is obvious. Proving that an architectural property of interest holds at an abstract level is much easier than proving that it holds at a more concrete level. A property that is easy to prove from a data-oriented description may be difficult or impossible to prove from a function-oriented description at a similar level of abstraction.

Another motivation for having several architectural descriptions, rather than just one, is to help fill the conceptual gap between a very abstract description of the architecture and its fully concrete implementation. Often, it can be difficult to determine whether an abstract architectural description is accurately describing the implemented architecture. Unless we have confidence that the description is accurate, there is no reason to believe that properties of the description will be true of the system. More concrete architectural descriptions also provide more guidance to system implementors and maintainers.

For those reasons, we have developed an architecture description language, called SADL [3], that represents collections of architectural descriptions as *hierarchies*, with each description linked to others by interpretation mappings that have been shown to guarantee consistency of the collection, as shown in Fig. 2.

Often, it is useful to introduce a branching in the hierarchy and explore alternative implementations. Branching hierarchies can be thought of as defining a family of architectural variants. Defining variants that differ with respect to dependability properties is particularly useful. Achieving higher levels of dependability typically requires paying a price — such as increased development cost or decreased system speed — and the price that is perfectly appropriate for one application may be too high for another. Rather than making every user of an architecture pay the cost of dependability, variants that provide appropriate levels of security, fault tolerance, safety, and so on, can be developed

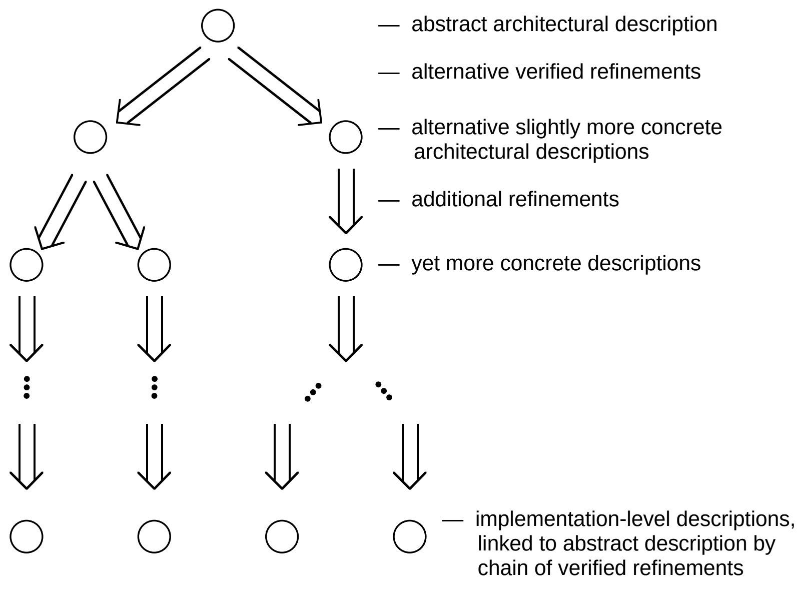
— abstract architectural description
— alternative verified refinements
— alternative slightly more concrete architectural descriptions
— additional refinements
— yet more concrete descriptions
— implementation-level descriptions, linked to abstract description by chain of verified refinements

Figure 2: A correct architecture hierarchy

and used where required.

An example of a family of architectures having varying dependability properties can be found in our formalization of secure variants of the X/Open Distributed Transaction Processing (DTP) standard architecture [2]. At the most abstract level of the hierarchy is a dataflow description of the X/Open standard [5] that also correctly describes the secure variants. Below this level, the hierarchy splits into secure and nonsecure branches. The nonsecure branch leads to a variety of valid implementations of the original X/Open standard. The secure branch adds certain general constraints on dataflow security, and then further branches to create variants suited to different applications. One branch describes architectures suited to applications where all components are multilevel secure, another branch describes architectures for the case where the components are single-level secure and the level of every component is the same, and yet another branch describes architectures for the case where single-level secure components vary in level.

So, current SADL technology provides a solution to the problem of relating abstract security models to concrete system implementations. A key objective in future research is to deal with other dependability properties — fault tolerance, safety, timeliness, and so on — in the same fashion. Overall system functionality and performance measures can vary radically as dynamic system architectures evolve. One sort of change is alteration or reallocation of functionality among components and connectors, with no change to connections as for example happens during a reconfiguration of an integrated avionics architecture. In this case, the structure of the architecture remains the same, while the allocation of function to structure changes. Alteration in architectural structure — adding or deleting components and connectors — is another sort of change.

4 Reasoning about dependability properties

SADL architecture description hierarchies provide a way of bridging the gap between abstract architectural models and their concrete implementations. If a hierarchy is developed using only refinement patterns that have been proven to preserve certain dependability properties, then the most concrete description in the hierarchy must have all the dependability properties of the more abstract descriptions. If it can be shown that some abstract description has a desirable dependability property, the most concrete description must also have that property. Since it is easy to verify that the most concrete description matches the actual implementation — because the most concrete description is based on the architectural constructs employed in the implementation — confidence that the property holds of the implementation is easy to obtain.

So, if an architectural description hierarchy is developed using verified refinement patterns, the proofs of the patterns guarantee that a certain class of properties is preserved. If it can be shown that one of the properties in that class holds at any level of the hierarchy, it must hold at every lower level, right down to the implementation level, as well. Thus, given a property of interest, the question naturally arises: *Does this property follow from the description at some abstract level in the hierarchy?* For some properties, like the security properties in the X/Open example mentioned above, the answer can be determined immediately by examining the abstract description. In other cases (including fault tolerance, safety and timeliness properties), complicated, error-prone reasoning may be required.

It is in these latter cases that an interface between SADL and theorem proving tools would be invaluable. An attempt to produce a formal derivation of the desired property from a description often reveals subtle errors in the argument, and in the specification itself. (Examples of incorrect informal proofs of properties of real-time system behavior revealed by the application of formal methods abound in the literature.) If the attempt is ultimately successful, confidence that the description in fact entails the property is substantially increased. For this reason, we are interfacing SADL with the PVS verification system.

5 The way forward

Our research agenda is to develop practical technology that can be used to prove the dependability properties of software architectures. We are particularly interested in timeliness, safety, fault tolerance, performance, and noninterference. We believe that the focus needs to be on reasoning about dependability-property-preserving transformations. Developing such transformations can be facilitated by focusing on a realistic, complex, ultra–high dependability application such as commercial or military avionics.

Under sponsorship from DARPA, the UK Ministry of Defence, and the UK Engineering and Physical Sciences Research Council, we have developed complementary methods and languages for reasoning about dependability properties and for specifying detailed architectural blueprints. We need to add several innovations to this background technology to facilitate definition and analysis of families of architectures with varying dependability characteristics. Specifically, we need to address these critical questions:

- *How do we add the dependability dimension to architecture definition and transformation?* ADLs need to be extended with appropriate linguistic support for expressing dependability constraints. They also need to be furnished with an appropriate semantics, to enable formal verification of architectural properties. The dependability constraints should be traceable through families of architectures and their respective hierarchies. Dependability-preserving transformations also need to be developed. Such transformations will typically involve expanding high-level control and mode functions, elaborating interfaces as well as synchronization and communication protocols.

- *How do we reason formally about architectural descriptions?* We need to develop a methodology and the associated infrastructure to support intuitive and highly automated formal reasoning about architectures by augmenting ADLs with interfaces to existing formal reasoning systems.

- *How can we enable synergy between reasoning tools?* The current state of the art means that theorem provers are inadequate, individually, to reason about all aspects of complex applications. We need to find synergies between the best currently available verification systems by applying them to different facets of a real-world problem. This can leverage their respective powers, thus enabling reasoning about systems beyond the reach of current technology.

- *How do we put this technology to use so as to enable building otherwise infeasible systems?* As a community we need to find ways of getting our technology out there in the field. Toy examples simply do not cut it. We need a key example, such as integrated modular avionics, which is intractable by current technology, to exercise and refine our methods, languages, and tools.

References

[1] *JIAWG Advanced Avionics Architecture (A^3) Standard, JIAWG Document No. J87-01*, December 1991.

[2] M. Moriconi, X. Qian, R. A. Riemenschneider, and L. Gong. Secure software architectures. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 84–93, May 1997. Available at `http://www.csl.sri.com/sadl/sp97.ps.gz`.

[3] M. Moriconi and R. A. Riemenschneider. Introduction to SADL 1.0: A language for specifying software architecture hierarchies. Technical Report SRI-CSL-97-01, Computer Science Laboratory, SRI International, March 1997. Available at `http://www.csl.sri.com/sadl/sadl-intro.ps.gz`.

[4] A. Nadesakumar, R. M. Crowder, and C. J. Harris. Advanced system concepts for future civil aircraft — an overview of avionics architectures. *Proceedings of Institution of Mechanical Engineers*, 209:265–272, 1995.

[5] X/Open Company, Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K. *Distributed Transaction Processing: Reference Model*, November 1993.