

Functionality/Dependability Co-design in Real-Time Embedded Software

Elisabeth A. Strunk John C. Knight
Department of Computer Science
University of Virginia, USA
{strunk, knight}@cs.virginia.edu

Abstract

As embedded processors become more powerful, demand for quantity and complexity of real-time embedded function increases. This increase in function is in direct opposition to system dependability requirements, since dependability is harder to achieve in larger or more complex applications. While tradeoffs must be made in designing systems to achieve both these ends, system architectures have the potential to ameliorate the problem. In this paper, we advocate the use of survivability as a mechanism for maintaining dependability while attaining the functionality desired by users and discuss research directions needed to realize its benefits.

Keywords: functionality/dependability co-design, survivability, safety-critical systems.

1. Introduction

The more computing power that embedded processors provide, the more functionality application designers desire to include in embedded systems. This functionality, while offering many possibilities in terms of convenience and safety, can quickly become much more complex than what humans are able to comprehend. Lack of comprehension introduces opportunities for error, and in systems that must be dependable, such as those described as *safety-critical*, those errors could easily have unacceptable consequences. Furthermore, the resources required to ensure dependability of conceptually simple but extensive functionality might be more than a customer is willing to provide. This sort of tradeoff suggests that functionality/dependability co-design will become an issue of increasing importance.

Limiting the functionality included on a processor is an infeasible option. Not only does it seem a poor choice economically, in practice it is likely to be ignored. Introducing additional complex safety checks can add its own risks due to an increasing incomprehensibility of the overall design. We therefore need technologies that enable designers to include new functionality without compromising the critical dependability requirements of the system.

Sha has proposed the use of simplicity in dealing with complexity [7], and has shown how this works in control systems. In this paper we introduce the notion of applying *survivability* to embedded systems, extending Sha's concept to the more general framework of arbitrary embedded systems while combining it with principles from the field of critical networked information systems. The goal in designing a survivable embedded system is to develop the system in such a way that it provides crucial functionality during operation even if it is not able to provide non-crucial functionality. By doing so, different dependability requirements can be associated with different functional elements, and, provided the system is designed appropriately, crucial system properties, such as safety, can be maintained even if desirable though non-crucial functionality cannot.

In practice many safety-critical systems are built this way, although with an ad hoc approach. We propose a general, comprehensive approach based on a rigorous definition of survivability. This approach permits a tradeoff between the degree to which functionality is maintained and the cost of system development. Within an application, it also provides a feasible route to the ultradependable implementation of crucial services without demanding ultradependable implementation of the entire application, a goal that is often technically infeasible using more traditional methods.

The remainder of this paper is organized as follows. Section 2 discusses why survivability is a good strategy for addressing the functionality/dependability co-design issue. Section 3 gives background on survivability from other disciplines and defines it for embedded real-time systems. Section 4 gives a brief example of what this might mean in

terms of avionics system regulations. Sections 5 and 6 enumerate future research directions needed to realize the potential of this strategy, and Section 7 concludes the work.

2. Why Survivability?

Many techniques have been developed to support the engineering of dependable systems. In a broad sense, these techniques fall into three primary areas: fault avoidance, fault elimination, and fault tolerance. Combined with analysis techniques such as fault-tree analysis, event-tree analysis, and failure-modes-and-effects analysis, these approaches to dealing with faults permit useful dependability predictions to be made about specific designs.

The use of these techniques in various combinations can be extremely useful in building dependable systems. However, none of them effectively address the dependability problems arising from the growing complexity of embedded software. For example, showing that an entire modern avionics system designed for a commercial air transport meets the FAA's mandated safety goal (see below) is, in general, beyond the present state of the art.

In many cases, much of the functionality included in a system is not directed primarily at system safety. For example, the autopilot system on a commercial air transport could contribute to an accident, but while it is a significant part of the safety case for the aircraft, cessation of its function is unlikely to have catastrophic consequences. This suggests that such a system does not need to be *ultradependable* as much as it needs to be *fail-stop* [6]. Provided the autopilot either works correctly or stops and alerts the pilot, the aircraft is unlikely to come to harm because of it. The complete avionics system for the aircraft needs to be able to operate without the autopilot (and other similar subsystems) so that safety is not compromised even though an emergency landing might be required if the autopilot fails. Such a strategy also reflects the practice of *safe programming* as advocated by Anderson and Witty [1]. In an informal sense, such an avionics system is *survivable* rather than *ultradependable*.

The notion of survivability has been discussed extensively in the area of networked information systems, and numerous informal definitions of the term have appeared. Knight, Strunk, and Sullivan have presented a more rigorous definition based on the idea that a survivable system is one that complies with its *survivability specification*, a structure that defines the dependability requirements that must be met for different sets of system functionality [3]. We claim that this rigorous definition can be applied in a straightforward manner to the domain of real-time, embedded systems with significant benefits, including the provision of a precise framework for functionality/dependability co-design.

Defining survivability in terms of a specification offers significant advantages in systems engineering, in comprehensibility of the necessary dependability properties of a system, and in demonstration of those dependability properties. In terms of systems engineering, the specification permits domain experts to define precisely what functionality is crucial to system dependability before the software is designed, thereby enabling designers to make appropriate tradeoffs. Specifications allow experts to see software at a high level of abstraction, aiding them in understanding what the system as a whole is required to accomplish. The specification can require overall dependability properties of the system as well, defining fault conditions under which those properties must hold. Some formal systems offer considerable benefits through design verification of such dependability properties. Finally, being able to provide crucial functionality in the presence of certain classes of faults means that those faults do not have to be tolerated by the entire system, and demonstrating that only a part of the system tolerates those faults is not only less expensive but also a much more tractable problem.

3. Definition Of Survivability For Embedded Systems

3.1. Survivability in Critical Information Systems

Survivability is an established research discipline in the realm of critical information systems. The general idea of survivability is that a system will "survive" (i.e., continue some operation), even in the event of damage. The operation it maintains may not be its complete functionality, or it may have reduced dependability properties, but it will be some useful functionality that provides value to the users of the system including possibly the prevention of catastrophic results due to the system's failure.

Like many terms used in technologies that have not yet matured, however, survivability is not defined with the rigor we would like to see. It has roots in other disciplines that begin to indicate what it should mean in our field; for instance, the telecommunications industry defines survivability as:

Survivability: A property of a system, subsystem, equipment, process, or procedure that provides a defined degree of assurance that the named entity will continue to function during and after a natural or man-made disturbance; e.g., nuclear burst. Note: For a given application, survivability must be qualified by specifying the range of conditions over which the entity will survive, the minimum acceptable level or post-disturbance functionality, and the maximum acceptable outage duration [6].

The network survivability community has attempted to come up with a more directly applicable description, resulting in definitions such as Ellison's:

Survivability: The ability of a network computing system to provide essential services in the presence of attacks and failures, and recover full services in a timely manner [3].

The sundry definitions of survivability vary considerably in their details, but they share certain essential characteristics. One of these is the concept of service that is essential to the system. Another is the idea of damage that can occur, and responding to that damage by reducing delivered function.

Definitions such as these are inadequate because they do not give system developers criteria for determining whether a system is survivable. One cannot determine whether a system is survivable if one is unsure exactly what survivability means. Also, knowing exactly what survivability means in general does not ensure that a straightforward implementation of survivability exists for a particular system. A precise definition is necessary in order to make survivability a meaningful system property.

Knight *et al.* give a definition based on specification: "A system is survivable if it complies with its survivability specification" [3]. They draw on the properties mentioned above and present a specification structure that tells developers what survivability means in an exact and testable way. When followed, this structure will cause them to document what it means for their system to be survivable. It is this perspective we take when defining survivability in embedded systems.

3.2. Requirements of a Survivability Specification

Embedded real-time software has certain similarities to and differences from large networked systems. It has a certain level of intellectual manageability stemming from its less distributed nature. However, it still rarely possesses the qualities of what could be considered a stand-alone application. Embedded systems generally receive input from and send output to other devices; this is the purpose for which such systems are built. These devices can fail just as network nodes can, and such failures must be considered in order to build software that is safe.

Adding to the problem is the inherent functional complexity of many embedded systems. Networked survivable systems are designed to deal with the failure of software on individual nodes; but, when dealing with embedded systems, that software may be in the logical central node. Input and output devices generally are not designed to compensate for failure of the embedded software, and so the software must be designed to survive internal failures. Furthermore, many embedded systems require some minimal level of function to ensure safety. For example, software controlling aircraft flight cannot simply terminate in midair; there must be some basic level of operation that it is guaranteed to maintain.

Functionality/dependability co-design decisions, then, must make some compromise between the functionality a user desires to see in a system and the minimal functionality a system must maintain to be considered dependable. We are proposing essentially a framework where the former is the primary function and the latter the backup. This is an incomplete view, however, for three reasons:

- *User expectation.* The user is likely to expect some minimum probability that the full function is provided. Operating exclusively in backup mode is almost certain to be unacceptable.
- *Multiple functionalities.* Usually, there will be more than two major classes of function. If the system must degrade its services, some services are likely to be more valuable than others even if they are not essential for dependable operation, and the system should continue to provide those services if possible.
- *Value as a function of state.* What is essential for dependable function usually depends on prevailing conditions. In other words, the functionality that is determined to be crucial by domain experts will usually depend upon operating circumstances. As an example, consider an automatic landing system. It could halt and simply alert pilots of its failure if it were not in use (i.e., in standby mode), but if it were controlling an aircraft it would have to ensure that pilots had time to gain control of the situation before halting.

These concepts are used extensively in industrial software development, but the survivability framework puts them on a rigorous footing. This enables them to be analyzed to determine whether they do in fact satisfy the user's needs.

3.3. Defining Survivability

The criteria above are vague, and using them informally will not enable developers to determine whether they have constructed a system that meets them. We therefore must define what we mean by a survivability specification in some rigorous but general way so that, when a specification is built using the framework, it can be analyzed to determine whether it possesses all the necessary information. The form of survivability specification we will use for embedded systems has six elements:

S: the set of functional specifications of the system. This set includes the preferred specification defining full functionality. It also includes alternative specifications representing forms of service that are acceptable under certain adverse circumstances (such as failure of one or more system components). Each member of *S* is a full specification, including dependability requirements such as availability and reliability for that specification.

E: the set of characteristics of the operating environment that are not direct inputs to the system, but affect which form of service (member of *S*) will provide the most value to the user. For example, when developing an aircraft automatic landing system, whether the aircraft has reached decision height (the height below which it is committed to land) might be a member of *E*. Above decision height it might be safer for the system to pull the plane up before relinquishing control, while below decision height it would leave the aircraft on the course to land. Each characteristic in *E* will have a range or set of possible values, in this case *above decision height* or *below decision height*. These values also must be listed.

D: the set of assignments of value to members of *E* that the system might encounter. This is essentially the set of all modes (i.e., collection of states) the environment can be in at any particular time. Each element of *D* is some predicate on the environment. *D* will not necessarily be equal to the set of combinations of all values of elements in *E*; for example, if *stage of flight* were also a member of *E*, then {*below decision height, enroute*} could not be a member of *D* because it is an unreachable state of the environment.

V: matrix of relative values each specification provides to the user. Each value will be affected both by the functionality contained in the specification and the environmental conditions for which that specification is appropriate. For example, the primary specification might have value 5 under all members of *D*, the alternative of pulling up value 2 when above decision height and 0 below decision height, and the alternative of continuing on current course value 2 when below decision height and 0 above decision height. Quantifying these values is impossible, but using relative values (as is done in economic utility theory) gives the ordering a developer needs to implement the system.

T: the valid transitions from one functional specification to another. Each member of *T* represents a transition from one specification to another. It includes the specification from which the transition originates (source specification), the specification in which the transition ends (target specification), and a member of *D* defining the environmental conditions under which that transition may occur (the transition guard). The guard enables a specifier to define which transitions are valid under certain circumstances, and the developer can then use *V* to decide which target specification is most valuable under those conditions.

P: the set of probabilities on combinations of specifications. Each member *P* will be a set of specifications mapped to a probability. The set of specifications is the specifications that provide approximately the same level of functionality, under different environmental conditions. The probability is the probability of a failure occurring in the system when the system is in compliance with one of those specifications (or the single specification, if there is only one in the set for that probability). The probabilities serve to provide a lower-bound guarantee of system operation.

4. An Avionics Example

As an example of how survivability can be applied to embedded systems, we show how it might be used with the current dependability requirements for U.S. commercial avionics systems put forth by the US Federal Aviation Administration (FAA). The FAA categorizes aircraft functionality into three major levels of criticality according to the potential severity of its failure conditions [4]:

Minor: failure conditions which would not significantly reduce airplane safety, and which involve crew actions that are well within their capabilities...

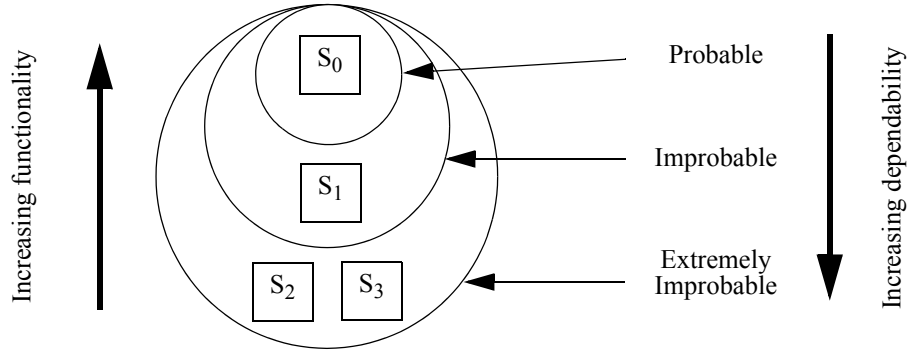


Figure 1. Function and dependability in a survivability specification

Major: Failure conditions which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, (i) A significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or some discomfort to occupants; or (ii) In more severe cases, a large reduction in safety margins or functional capabilities, higher workload or physical distress such that the crew could not be relied on to perform its tasks accurately or completely, or adverse effects on occupants.

Catastrophic: Failure conditions which would prevent continued safe flight and landing.

Failure conditions must have probabilities of not occurring proportional to the potential consequences of their occurrence. “(1) Minor failure conditions may be probable. (2) Major failure conditions must be improbable. (3) Catastrophic failure conditions must be extremely improbable” [4]. “Probable” is defined as “anticipated to occur one or more times during the entire operational life of each airplane”; “improbable” as “not anticipated to occur during the entire operational life of a single random airplane”; and “extremely improbable” as “so unlikely that [the failure condition is] not anticipated to occur during the entire operational life of all airplanes of one type” [4]. Quantifying these definitions leads to probabilities that can be extremely small. “Extremely improbable”, for example, corresponds to a failure rate of 10^{-9} per hour of operation.

In our automatic landing system example, we will assume four functional specifications, as shown in Figure 1. The first, *primary*, specification (S_0) will have all of the functionality the user desires for the system. The consequences of any failures will be minor because, if they have the potential to be more severe, the system can transition to one of the other three specifications. Therefore, any failure in the primary specification may be “probable”.

The first alternative specification (S_1) will have much of the functionality desired by the user, but some desirable yet unnecessary functionality removed. For example, the system might have to follow the step-down altitude clearances for the runway to descend at the proper rate rather than using the glideslope. All failures in this specification must be “improbable”; its functionality is important enough that frequent interruptions could have adverse consequences. However, none of it need be “extremely improbable” because any failures with potentially catastrophic consequences will cause a transition to a different alternative specification (S_2 or S_3).

S_2 and S_3 are the specifications that have very high dependability requirements. We will let S_2 be the specification requiring the aircraft to pull up and alert the pilot on system failure and S_3 be the specification requiring that the system continue on its current course when alerting the pilot if the system fails. They contain the minimum functionality necessary to maintain safe operation of the system. Any non-masked failure of either of these specifications—such as failure to alert the pilot that the system has malfunctioned and the pilot is now in control—must be “extremely improbable”, as they are designed to include only the system functionality whose failure could have catastrophic consequences.

A major factor that the FAA guidelines for flight systems does not address is changes in what system dependability requirements might be based on environmental circumstances. Whether the system transitions to S_2 or S_3 on a failure of S_1 depends on whether the aircraft is above or below decision height at the time of the transition. The new probability requirement, then, would be that a failure of S_2 above decision height is “extremely improbable”, and a

failure of S_3 below decision height is “extremely improbable”. In some cases the environmental conditions might change, and a transition between specifications appropriate to different conditions must occur in order to keep the system operating with the optimal functionality.

Finally, it is possible that the system could recover from a failure that forced it to transition to a lower level of functionality. For instance, the aircraft might operate under specification S_1 while the glideslope transmitter is reset to recover from a transitory error, then transition back to specification S_0 . A specification structure for a survivable system should provide for this as well.

In considering this example, it is important to note the difference between this discussion and the normal approach that is taken by regulating agencies such as the FAA. The example illustrates the use of survivability and the association of different dependability requirements with different functionalities within the survivability specification. This contrasts with the current situation in which the entire system would be assigned a certification level by the FAA even though much of the functionality would not require that level of dependability.

5. Research Challenges in Functionality/Dependability Co-design

The notion of functionality/dependability co-design is complex yet potentially very fruitful. There are numerous remaining issues that need to be resolved before the concept as manifested in the notion of survivability applied to embedded systems can be used routinely. In this section, we present some of the research issues.

- *Defining confidence for different software dependability levels.* Current software practice defines measurements for determining confidence in software, but these measurements tend to be based on intuition and can vary widely across application domains and regulatory agencies. The research community has argued that engineers cannot be confident that current software will function at ultradependable levels. The aim of functionality/dependability co-design of software is to make this a tractable problem. Powerful but potentially expensive mathematical analysis such as model checking and putative theorem proving can be applied to critical parts of a system while leaving most of a company’s standard development processes intact. Testing might be able to show statistically that certain functionality meets its less stringent dependability requirements. Determining whether ultradependable functionality meets its requirements still is not, however, a statistical problem [2]. The computing research community is now presented with the challenge of deciding what metrics or processes give us confidence that software conforming to them indeed achieves stated dependability goals.
- *Validation of survivability specifications.* Potentially, the most valuable benefit from building survivable specifications is a significant reduction in complexity for the most dependable portions of a software system. This reduced complexity can facilitate inspection and validation of the system. It can also help application domain experts determine what properties are required in order to ensure the system is dependable in the common sense, i.e., that its users can depend on its performance. The composite survivability structure, however, is more complex than any individual specification. Inspection and other validation methods are needed to ensure experts’ understanding of the overall function of the system.
- *Analysis of transitions between specifications.* The powerful advantages gained through applying survivability require that certain real-time and other properties of transitions between specifications be guaranteed. The specifier must be able to show that the system is able to transition to an alternative specification without violating the system’s overall dependability requirements if he is to claim that the alternative specification can stand in for the primary specification in terms of dependability analysis. How this is to be shown is still a research issue.
- *Determining criteria for violation of dependability requirements.* In typical dependable systems, violation of dependability requirements is strictly disallowed and so determining when a violation occurs is unnecessary. Because survivability permits some leeway in this aspect, decidable criteria for these properties must be defined. Availability and reliability, for example, are defined probabilistically, but their probabilities are defined over some period of time that in the current framework is unbounded. In practice, real-time requirements impose a bound whose interaction with other requirements must be assessed in order to determine when dependability properties are violated.
- *Establishing composite system properties.* A survivable system is broken into separate survivability specifications with separate probabilities, but these specifications come together to form the overall system specification. It is the properties of this overall specification in which the user is interested for purposes of determining deliv-

ered value. For example, proving that a failure of the automatic landing system to alert pilots when it malfunctions is extremely improbable does not show that the autopilot will function as desired while enroute. It is these properties that must be studied to optimize decisions in functionality/dependability co-design.

6. Research Challenges in Hardware/Software Co-design of Survivable Systems

Survivability as a system concept has impact beyond functionality/dependability co-design. It is quite possible for it to be exploited to assist in the process of hardware/software co-design because it provides a much more flexible software architecture than is found in current non-survivable designs. Also, effective hardware/software co-design can magnify the benefits of survivable systems. We discuss in this section several challenges in the area of hardware/software co-design as it relates to survivability.

- *Distribution of survivable software over available hardware.* Faults in the software in a survivable system can originate in damage to its underlying hardware, and reconfiguration can be initiated by hardware failure. We have presented a specification framework aimed at aiding functionality/dependability co-design of the software portion of the system. Further research in determining where critical pieces of function should reside, or how best to overlay the software on available hardware, could increase realized benefits from this architecture.
- *Analysis of tradeoffs between hardware cost and software function in survivability levels.* Survivability can be employed to avoid certain hardware-based solutions to dependability issues. Since convenient but noncritical functionality does not have to be dependable, a simpler but more fragile hardware implementation of it could be built. This would reduce development cost of the system, but at the expense of user convenience or satisfaction. Deciding what function belongs in each specification, which then dictates the dependability level at which that function must be implemented, is an area that holds great potential for economic research in system development.
- *Refining software's role in system risk analyses.* Attempting a more rigorous definition of what confidence means for software at certain dependability levels implies that a new definition of what confidence in a system means is also needed. For instance, if testing is to give a certain amount of confidence in software, one might question what hardware configuration is necessary to carry out that testing. Alternatively, if certain classes of software function are claimed to benefit from design diversity, hardware might play a practical role in this by providing options such as writing different versions of software for different hardware platforms. Finally, software might be implemented on hardware in such a way that the hardware prevents certain interactions between components. How to construct a risk analysis of the overall hardware/software system and what design decisions might facilitate that analysis are further promising future research areas.
- *Implementation feasibility.* Software specification languages can express functions not implementable in any programming language, and dependability levels can be required that are impossible to achieve with finite hardware. For example, it is possible to specify an oracle to the halting problem that will succeed with a probability of 1. The question of whether a feasible implementation of a specification exists must be answered for any system, but survivable systems can encompass a broader range of functionality and dependability requirements since they allow the tradeoff to be made in a gradual manner. Research in efficient iteration between specification and design stages of hardware and software development could greatly increase the efficiency of processes for building survivable systems.

7. Conclusion

As embedded real-time systems become larger and more complex, issues in functionality/dependability co-design will become increasingly important. Addressing these issues becomes much easier with the observation that much of the desired functionality is not essential for dependable system operation. Separating safety-critical function and designing it for ultradependability while implementing the residual function in a less dependable way is a promising solution. This strategy is currently employed in the domain of networked information systems under the name *survivability*. We have discussed several definitions of survivability, outlined a rigorous definition as applied to embedded real-time systems, and presented future directions for the research community to explore in realizing its benefits.

Acknowledgments

We thank William Greenwell for his assistance in constructing our avionics examples. This work was funded in part by NASA Langley Research Center under grants numbered NAG-1-2290 and NAG-1-02103, in part by the Defense Advanced Research Projects Agency under grant N66001-00-8945 (SPAWAR), in part by the U.S. Air Force Research Laboratory under grant F30602-01-1-0503, and in part by Microsoft. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the U.S. Air Force, or the U.S. Government.

References

- [1] Anderson, T., and R. W. Witty. "Safe programming." *BIT*, 18:1-8, 1978.
- [2] Butler, R. W., and G. B. Finelli. "The Infeasibility of Experimental Quantification of Life-Critical Software Reliability." ACM SIGSOFT '91 Conference on Software for Critical Systems, New Orleans, LA, December 1991.
- [3] Ellison, B., D. Fisher, R. Linger, H. Lipson, T. Longstaff, and N. Mead. "Survivable Network Systems: An Emerging Discipline." Technical Report CMU/SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University, November 1997.
- [4] Federal Aviation Administration Advisory Circular 25.1309-1A, "System Design and Analysis."
- [5] Knight, J. C., E. A. Strunk and K. J. Sullivan. "Towards a Rigorous Definition of Information System Survivability." DISCEX 2003, Washington, DC, April 2003.
- [6] Schlichting, R. D., and F. B. Schneider. "Fail-stop processors: An approach to designing fault-tolerant computing systems." *TOCS* 1(3):222-238.
- [7] Sha, Lui. "Using Simplicity to Control Complexity." *IEEE Software* 18(4):20-28.
- [8] U.S. Department of Commerce, National Telecommunications and Information Administration, Institute for Telecommunications Services, Federal Standard 1037C.