Balancing reliability and availability

Chris Hobbs, Senior Developer, Safe Systems, QNX Software Systems Limited

On 14 September 1993, Lufthansa Flight 2904 overran a runway in Warsaw because the reverse thrust deployment system operated exactly to specification.[1] Unfortunately, the Airbus designers had not anticipated conditions during a cross-wind landing. In an analogous incident, on 11 July 2011, a Victoria underground train in London moved off with the doors open, not because of any system failure, but because the system designer had not anticipated how the system would be used.[2]

The interaction of a system with its environment is two-way: The environment places unanticipated requirements on the system, and the system places stress on the environment. This article explores a few of those interactions by considering the adverse impact of system reliability over availability.

Dependability
For a software-based system, *dependability* is a combination of *availability* (how often the system responds to a stimulus in a timely manner) and *reliability* (how often the response is correct). For some systems, reliability is more important than availability—no action is better than the wrong action; For other systems, availability is more important—any reasonable response is better than no answer.

Design safe states
Whether our system is more sensitive to availability or reliability, it must take defined actions when a dangerous condition arises. Usually, it must move into its *Design Safe State*. Too often, designers fail to consider the impact of this move on the wider environment in which the system operates.

Most complex systems are built by integrating subsystems, each developed without an understanding of the larger systems into which it will be integrated. Databases, operating systems, communications stacks and similar components are designed to operate in any environment meeting the conditions defined in their Safety Manuals; they are not developed for a specific system. In IEC 62304 compliant medical devices, such components are termed SOUP (Software of Unknown Provenance); in ISO 26262, automobile systems they are known as SEooCs (Safety Elements out of Context).

At the Safety-Critical Systems Symposium in 2012, Martyn Thomas drew attention to *accidental* systems—systems where the designers are unaware of the hidden dependencies between components from different sources.[3] The behavior of the overall system when an unanticipated condition occurs is, therefore, very difficult to compute, because it depends on multiple, largely independent, components moving to their Design Safe States. Even if our complete system moves cleanly to its Design Safe State, it is likely to be part of a larger system which in turn will experience increased stress.

The environment

A system is an arbitrary grouping of components; my system may be your component. For example, Manufacturer A assembles components into a system: a Doppler radar for measuring the speed of a train. Manufacturer B considers this system a component of a train control system, braking the train when its authority limit is reached.

If the train's Doppler radar encounters conditions it wasn't designed to handle, it will move to its Design Safe State, sending an error signal to the control system. The control system will have been programmed to handle this and may switch to a GPS source for speed information. This switch causes extra stress, because the system must execute paths that have been less well-exercised.

If this train control system were an "accidental" system, the Doppler radar might be internally dependent on GPS signals for timing information. Its failure might be caused, therefore, by the GPS signals being jammed. In this case, switching to a GPS source wouldn't be useful. Two failures that the system designer considered as independent during failure calculations have turned out to be strongly correlated.

If the control system itself fails, it will move to its Design Safe State, probably applying the train's brakes. Although this will make the train safe, it puts additional stress on the railway signaling system—a train unexpectedly stationary on a high-speed track must have been considered in the signaling design but, again, dealing with this event will use an execution path that is infrequently exercised in the field.

Reducing environmental stress

Our system affects its environment, which in turn generates unanticipated stimuli. The Victoria underground train and the Airbus at Warsaw incidents both illustrate the dangers associated with such stimuli.

More frequently, problems occur when a system puts additional stress on the larger system within which it operates. The Victoria underground line provides further insight here: There are usually more trains on the line than there are stations.[4] The Design Safe State for each train might be to stop at the next station, but this isn't possible at the system level: There are insufficient stations.

To map our system's behavior onto its environment, we have to consider the four states that our system may be in at any time (Table 1).

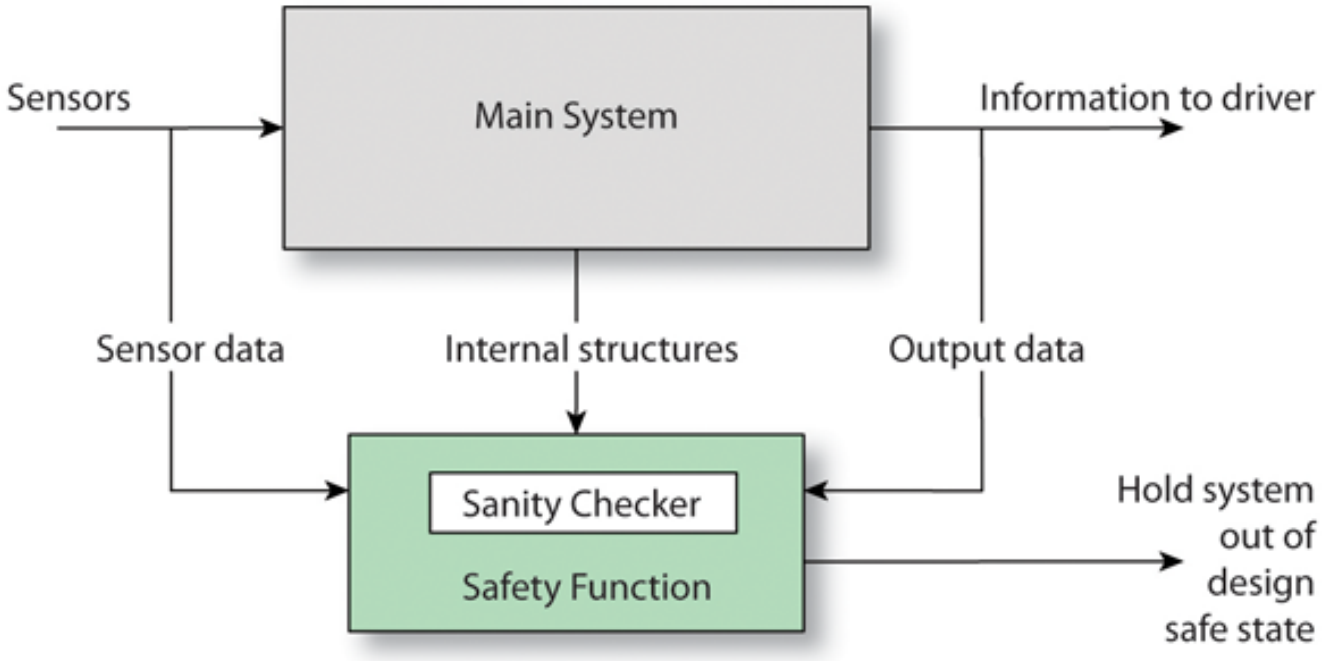
State		Device safe?	Device useful?	Stress on the environment?
I	Operating, no dangerous condition	Yes	Yes	No
II	Safe state, no dangerous condition	Yes	No	Yes
III	Operating, dangerous condition	No	No	Yes
IV	Safe state, dangerous condition	Yes	No	Yes

In state I, the system is operating correctly, no dangerous condition has occurred and no additional stress is being caused. In state IV, a genuinely dangerous situation has occurred and the system has, correctly, moved to its Design Safe State. This has put stress on the environment, but the system had no choice. State III, where the system has failed to detect a dangerous condition but continues to operate, is unsafe. Our design aims to reduce the probability of this state occurring to below an acceptable level (e.g., probability of less than 10^{-7} per hour of operation).

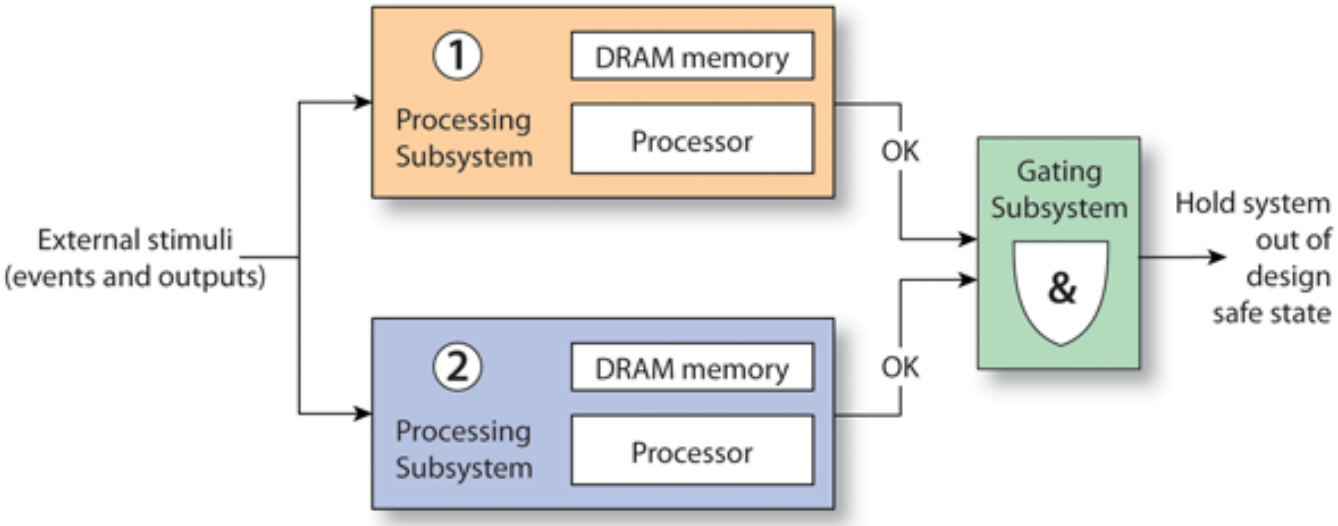
State II is the state which perhaps receives the least attention. Here, the system has incorrectly detected a dangerous condition that doesn't actually exist. It moves unnecessarily to its Design Safe State, increasing stress on its environment.

The "hair trigger" system

Figure 1 illustrates a common design for safety-critical systems. The main system accepts sensor values and performs the system-dependent calculation to determine what action to take. Its operation is monitored by a Safety Function (in IEC 61508 terms), performing a much simpler (and therefore less error-prone) computation to decide whether the main system is sane (Figure 2). To increase the reliability of the Safety Function, we design two monitors working in parallel—if either monitor believes that the main system is not sane, it indicates this finding, and the system is no longer held out of its safe state. Hence, this is a 1 out of 2 (1oo2) system: Either monitor can cause the system to move to its design safe state.



A key phrase in the previous paragraph is “To increase the reliability….'' The 1oo2 system does increase reliability, but at a serious cost to availability. If either monitor is corrupted, for example by a memory error induced by the train's driver placing a cellular telephone close to the device, then the system immediately enters state II, reducing the system’s availability and placing unnecessary stress on the environment.



Getting a second opinion

We can expect that Heisenbugs cause most failures in our system, because Bohrbugs will almost certainly have been detected and removed during testing. A Heisenbug may arisen from the software itself (e.g. a subtle and infrequent race condition) or from a soft memory error (e.g. caused by the driver's cellular telephone). Whatever the cause, the characteristics of a Heisenbug mean that rerunning the computation will give a different, presumably correct, result.

Is there time to rerun the computation? That depends on the system. A high-speed

train can travel about 100 meters per second. Is it acceptable when the two monitors disagree to hold off the move to the Design Safe State to allow the computation to be rerun? Remember, if the two monitors agree to move to the Design Safe State, then it will occur; we are only talking hereabout the condition when they disagree.

The answer, of course, comes from the failure analysis of the system, but in many cases incorporating this type of “second opinion” design can increase the availability of a system significantly without seriously degrading reliability. And when the failure analysis extends to incorporate the larger environment, increasing system availability at the cost of reliability can often improve overall safety.

Thinking outwards and inwards

Systems used in applications affecting safety must be dependable. But dependability consists of reliability and availability, which are often antagonistic—we enhance one at the expense of the other.

When designing a system it is tempting to consider it in isolation, stating that “our system is safe”, without considering the effect of its non-availability on the larger system in which it will run. Unfortunately, when using a complex component it is all too easy to incorporate hidden failure dependencies into our system (building an accidental system) that invalidate our failure calculations.

As designers of embedded software we naturally focus on developing and validating our software to build its safety case. Often, we can do little more than make our environmental assumptions explicit, because we do not have control (or even knowledge) of how and where our software will be used. When we do know the environment, it is natural for us to think outwards from the software to its environment; this is essential because of the stress we can place on that environment. We may build safer systems, however, if we also think inwards from the environment back to our software.

[1] Main Commission Aircraft Accident Investigation Warsaw,Report on the Accident to Airbus A320-211 Aircraft in Warsaw on 14 September 1993. <www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Warsaw/warsaw-report.html [1]>

[2] Rail Accident Investigation Branch, Department for Transport, Rail Accident Report, Train departed with doors open, Warren Street, Victoria Line, London Underground, 11 July 2011. Report 13/2012, July 2012. <http://www.raib.gov.uk/cms_resources/120705_R132012_Warren_Street.pdf [2]>

[3] Martyn Thomas, “Accidental Systems, Hidden Assumptions and Safety Assurance”.<tv.theiet.org/technology/manu/12667.cfm [3]>

[4] See Jonathan Storey, “Safety Case Approach for the Victoria Line Resignalling Project”, SCSS 2013.<http://tv.theiet.org/technology/manu/16018.cfm [4]>

Source URL (retrieved on *09/18/2014 - 3:30pm*):
http://www.ecnmag.com/articles/2013/03/balancing-reliability-and-availability

Links:
[1] http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Warsaw/warsaw-report.html
[2] http://www.raib.gov.uk/cms_resources/120705_R132012_Warren_Street.pdf
[3] http://tv.theiet.org/technology/manu/12667.cfm
[4] http://tv.theiet.org/technology/manu/16018.cfm