Safety Case strategy for COTS

Nicholas Mc Guire
Distributed & Embedded Systems Lab
Lanzhou, China *safety@osadl.org*, *mcguire@lzu.edu.cn*

Software Safety Case

- Problem:
  - can't quantify failure rates
  - no quantitative risks associated with software modules
- Mitigation
  - qualitative methods (i.e SIL assignment 61508-5)
  - use of best practice techniques
  - software safety case: "confidence by arguments"

Justification is a matter of convincing the regulatory authority that you know what you are doing...

Source of Confidence in Safety Cases

Basically one can distinguish two types of safety cases.

- procedure based safety case
  - review driven (61508, CENELEC, DO178B)
  - evidence of meeting spec. is based on a formal set of documents and there review
- evidence based safety case
  - goal driven (CAP 670)
  - evidence of meeting spec. is derived from field service, testing and analysis

Q: evidence based safety case squeezed into 61508 and derivatives ?

Types of Safety Cases

- Monolitic - the safety case is based on a well defined sysem level context
- Modular - the safety case is based on a local context and contracts for interaction with "others" for a class of equivalent systems.

COTS components in monolitic safety cases void the whole pupose of using COTS - but not even 61508 is strictly monolitic and left sufficient space for arguments diverting from the standard.

"...has been conceived with a rapidly developing technology in mind..." [IEC 61508-1 Introduction]

COTS Evidence base Safety Case

- Limitation of COTS/OSS - not all methods listed in CENELEC are applicable to OSS/COTS.
- CENELEC allows not using the Highly recommended methods if they can be argued away - can we argue all of them away ? (50129 Appendix A).
- COTS/OSS safety cases will require categories of evidence that are independant of the software developmen process.
- *..unless justified in the functional safety planing...* [61508-1 5.2.4]

Modular safety cases in Avionics

Avionics industry has been moving towards modular hardware and software for the past years.

- IMA/IMS driving force
- Modularity not only in hardware but also in Mission-SW
- Safety case Maintenance/Upgrade issues
- Safety case reuse issues
- Modularity is a prerequisite to Encapsulation of variable software elements

To utilize the benefit of COTS (especially software) a modular safety case seems to be mandatory.

Basic concept of a safety case including COTS

- Decomposition of the safety case
- COTS as black/gray-box
- reverse process: derived requirements for COTS
- search for best match [IEC 62061 7.8]
- contract based justificaiton of safety modules

It is assumed that a 100% match is NOT made - thus pending arguments must be satisfied "exernally" to the COTS module.

safety case structures for 61508 and derivatives

- Generic Product Safety Case
- Generic Application Safety Case
- Specific Application Safety Case

In this taxonomy the Linux kernel would fit into:

- Generic Product Safety Case as "POSIX OS Layer"
- Generic Application Safety Case as "Mostly POSIX Linux kenrel"
- Specific Application Safety Case as "linux-2.6.22-config3 PPC440"

safety case elements

- suitable safety process
- technical suitability - product provides necessary level of safety
- demonstrate: product + process = necessary level of evidence

Furthermore a safety case must show where indentified hazards are handled.

safety elements

- safety functions
- integration procedures
- safety applicatoin conditions (handling of open hazards)
- operation and maintenance (post comissioning safety)

For Linux ideally there are no safety functions (in the kernel) though there will be constraints through operational procedures and safety application conditions to constraint the OS-usage (i.e. limiting it to POSIX). The issue of maintenance of Linux will most likely mandate a "kernel update procedure".

safety justification: Procedures

Safety justification begins with a set of documents detailing:

- Kernel / OS Selection criteria and procedures
- kernel SW-lifecylce: coding style, feature lifecylce, commit window, etc
- Tools selection: suitability and reliability [61508-3 Table A3]
- Validatio tool chain: selection, evidence of suitability and reliability
- Testing: test-suites, standard conformance
- Maintenance: Tracability (git), bug-tracking, pier review (LKML, OLS, publications)

Naive proven-in-use "there are millions of linux systems..." is no more than a supportive argument - dont stressing it too much.

Strategic Options 61508 and derivatives

Options we see posible:

- proven-in-use
- divert from 61508 and justify
- bespoke nano-kernel
- application level safety
- total diversity
- argue the Linux SW-lifecycle as equivalen with respect to objectives

this is quite speculative - there is no one strategy that will "guaranee" 61508 compliance in all situations.

Strategic Options: reference 50128

Build a CENELEC complient COTS argument, which would be heavaly based on proven-in-use (refere to clause 3.4 - definition of COTS, which clearly points towards Proven-in-Use), and section 9.4.5 subclause i,ii and iii.

Justify this argument in 61508 context based on EN 50128 - provided the application fits the application sector constraints introduced in EN 50128 (rail signaling) sufficiently well.

Strategic Options: Evidence Based

Build a evidence based safety case which is clearly NON-61508 complient and argue the divergence from 61508 which is entirely procedure based.

- Compliance means that the **objectives are met:** ***"... and therefor for each claus or subclause, all the objectives have been met.***
- **Evidence base approaches must be justified:** ***"...exeption from compliance with such requirements is acceptable provided it is justified"***

Strategic Options: Virtualization

Build a CENELEC complient procedural safety case for a nano-kernel/microkernel that runs GNU/Linux as one of its (user-space) tasks running safety critical apps (or atleast the safety responsible components like voters) under direct control of the nano-kernel. The nano-kernel in this construction is a **SIL#** software entity not COTS !

This strategy seems resonable if GNU/Linux is to be utilized for SIL0 services (i.e. status monitoring, maintenance, etc.)

Strategic Options: SIL0 for COTS

Put the safety responsibility completly into the application and argue GNU/Linux as gray-channel by implementing:

- end-to-end checks not influenced by the OS
- providing N-version user-space safety critical application
- include external safety-bags (i.e. per-task HW-watchdog)
- coded monoprocess at the safety-related application level

Strategic Options: 61508-6 Annex E

Build a completly diverse system based on COTS and argue that the goal of 61508 of preventing systematic software errors is covered by total diversity. This is fundamentally simply a go at 61508-6 Appendix E.

X86	PPC
L4/Fiasco	XtratuM
L4Linux	paravirt GNU/Linux
safety-domain	POSIX safety-domain

Strategic Options: Justify GNU/Linux

Document the Linux development life cycle (the kernel that is) in a suitable way and argue that it provides comparable if not superior quality even though it does not follow the procedural requiremetns. In fact the stability of Linux-2.6 can in our opinion be argued in this way - the main issue really is if this is acceptable to the safeties.