Building a Preliminary Safety Case: An Example from Aerospace

Tim Kelly, Iain Bate, John McDermid, Alan Burns

Rolls-Royce Systems and Software Engineering
University Technology Centre
Department of Computer Science
University of York
York YO1 5DD, U.K.

Email: tpk|ijb|jam|burns@cs.york.ac.uk

Abstract

The phased production of safety cases, in step with an evolving design, is an increasingly common approach to managing the potential risk associated with certification. The Preliminary Safety Case, the first safety case to be issued, is prepared during the initial stages of project development. An important part of the Preliminary Safety Case involves defining the safety argument approach that is being adopted for the system. Such an argument can make clear the principal safety objectives and constraints of the project, and outline how they will be interpreted and addressed. In this paper we describe the production of these 'Preliminary Safety Arguments'. In particular, we show how we have used the Goal Structuring Notation as the basis for presenting the Preliminary Safety Argument for a distributed computing platform for aero-engine control. Through such an approach, we argue that certification risk can be reduced by deriving safety objectives in advance of system development rather than 'discovering' them after significant functional design commitments have already been made.

1. Introduction

The safety case is the document, or set of documents, presenting the argument that a system is acceptably safe to operate in a given context. For safety-critical and related systems, an acceptable safety case must typically be presented to the appropriate regulatory authority prior to a system being allowed to enter service.

Due largely to the fact that system certification is seen as a final 'hurdle', historically the production of the safety case has often been left until towards the end of the project – following system development, analysis and test. However, the risk with such an approach is that when producing the safety argument a 'shortfall' is discovered, due to the design and/or the analysis, that then demands potentially large amounts of rework. Cullen [1] describes such problems when consideration of the safety case was left too late in the design process for the BNFL Sellafield Alpha Reduction Plant. Another drawback is that by leaving safety case production until 'all is said and done' on system development it can be difficult to recall and capture the safety rationale that underlies safety related design and analysis decisions. The '*why?*' behind how safety features are implemented and the *assumptions* that were made in analysis can be lost. To avoid these problems *phased production* of the safety case, in step with the evolving design, is increasingly being recommended, and in some cases mandated, by safety standards (such as the U.K. Ministry of Defence (MoD) Standard 00-56 [2]) and regulators.

2. Phased Safety Case Production

To reduce the risk of forced redesign in order to receive system certification, it is necessary for projects to consider from the earliest possible opportunity, "*How will we argue that this system is safe?*" The safety case, as the presentation of how safety requirements have been decomposed and addressed, provides an appropriate vehicle for this consideration. For this reason, development of the safety case should be initiated early on in a project at the requirements definition stage and carried through to commissioning and beyond. Safety standards are increasingly recommending such an approach, for example the U.K. Ministry of Defence (MoD) Standard 00-56 [2] states that:

> *"The Safety Case should be initiated at the earliest possible stage in the Safety Programme so that hazards are identified and dealt with while the opportunities for their exclusion exist"*

Equally, another of the U.K. MoD Standards – JSP 430 The Ship Safety Management Handbook [3]– requires that:

> *"The Safety Case is to be prepared in outline at presentation of the Staff Requirement and is to be updated at each major procurement milestone up to and including hand-over from the procurement to the maintenance authority … Ideally there should be a seamless development of the Safety Case from one phase to the next"*

A common approach to managing the gradual development of the safety case is to submit a safety case at various stages of project development. For example, the U.K. MoD Defence Standard 00-55 [4] talks of formally issuing at least three versions of the (Software) Safety Case:

- **Preliminary Safety Case** – after definition and review of the system requirements specification
- **Interim Safety Case** – after initial system design and preliminary validation activities
- **Operational Safety Case** – just prior to in-service use, including complete evidence of having satisfied the systems requirements

In practice there may be variation on the above for specific domains, e.g. an additional safety case to clear an aircraft for flight trials.

In this paper, we focus on the role and content of the Preliminary Safety Case. In particular, we describe the presentation of *Preliminary Safety Arguments* as a means of clearly communicating the key safety requirements, claims and forms of evidence that are expected to form the structure of the final safety case. As an illustration of the approach, we describe an example preliminary safety argument created during the development of a distributed computing architecture for aero-engine control.

3. The Preliminary Safety Case

The Preliminary Safety Case will typically be prepared in a project after the following activities have been performed:

- **Production of Safety Plan -** Definition of the key safety processes, roles and responsibilities to be enacted during system development.
- **Identification of Required Safety Properties -** Including identification of applicable safety standards, the requirements from these standards that apply to the system under development and customer-desired safety properties.
- **Preliminary Hazard Analysis (PHA) -** Identification of potential system hazards through systematic review of the initial, top-level, system design – e.g. for a software system, through performing Software HAZOPS [5] over a high-level data flow diagram that defines the key processes and data flows.
- **Risk Estimation -** Estimation of the level of risk posed by each of the identified system hazards – e.g. through qualitative description of both the severity and likelihood of hazard consequences and use of a Hazard Risk Index (HRI) Matrix.
- **Identification of Failure Rate and Integrity Level Requirements -** Predominantly, identifying the principal requirements implied by the risks identified in the Risk Estimation exercise – e.g. tolerable failure rate targets for each risk category.

Notably, however, the Preliminary Safety Case will usually be prepared prior to there being any ***detailed*** *system design or specification*, and therefore before any detailed system safety analysis or testing is possible. Given the absence of design detail, one might question the value of producing a safety case at this stage in the project. However, the document can fulfil the following objectives:

- Defining the **scope** of consideration for the (final) safety case
- Declaring what have been identified as the **key safety issues and objectives** associated with the system - the principal System Hazards, Safety Requirements and Applicable Standards
- Defining the **approach** that is being adopted in arguing safety – including the key techniques and sources of **supporting evidence** to be employed
- Defining (safety-relevant) development **procedures** that will be enacted during system development, e.g. the languages, methods and tools to be used for each Software Integrity Level

Having fulfilled these objectives, submitting the Preliminary Safety Case to the customer (regulator) provides an early opportunity to get agreement, even if only informally, on the certification approach being adopted. In addition, the Preliminary Safety Case helps the developer to clearly set out the safety context within which the project must be executed. Through the document, safety objectives to be achieved are flagged in advance of system development – reducing the extent to which requirements will be 'discovered' after significant functional design commitments have already been made.

4. Preliminary Safety Arguments

As with both the Interim and final Operational Safety Cases, the Preliminary Safety Case will typically present information under the following headings: (Under each heading we describe the contents that could be expected at the time of producing the Preliminary Safety Case.)

- **Scope -** Boundary of concern, standards to be addressed, relationship to other systems / extant safety cases
- **System Description** - High-Level (Preliminary) Overview of the System: Key functions + Outline of Physical Elements
- **System Hazards** - Results of Preliminary Hazard Analysis - Key Credible Hazards. (These may well change for later submissions of the safety case.)
- **Safety Requirements -** Description of Top-level safety requirements (emerging from study of the standards, and the Preliminary Hazard Analysis), e.g. Failure Rate in particular Failure Modes.
- **Risk Assessment** - Results of Risk Estimation exercise, Accident Sequences, HRI used and the resulting Risk Classes for all identified Hazards. (As with the System Hazards – the assigned Risk Classes may change for later submissions of the safety case.)
- **Hazard Control / Risk Reduction Measures -** At this stage - how the project plans to tackle each identified risk - design measures, protection systems, redundancy etc.
- **Safety Analysis / Test** - At this stage - how the project intends to provide evidence of successful deployment of risk reduction measures, meeting failure rate targets, demonstrating correctness, etc.
- **Safety Management System -** Reference to contents of Safety Plan for roles, responsibilities, procedures. (This will be fairly stable for later safety cases.)
- **Development Process Justification -** An outline of the development procedures, design methodologies to be used, coding standards, change control procedures etc. and how these will be shown to meet integrity level, or development assurance level, requirements.
- **Conclusions -** At this stage - the key reasons why the project believes that the system will be safe to deploy the system, what will be concluded from analysis and test evidence etc.

Although each of the elements listed above forms a necessary part of the Preliminary Safety Case, as stated earlier one of the principal objectives is to obtain general agreement with the customer as to the argument approach being adopted on the project. To do this, it can be useful to explicitly present a *Preliminary Safety Argument* that describes the emergent safety requirements, the interpretation of these requirements and points forward to the claims that will be made about the system and the evidence that will be used in support of these claims.

In the example described in the next section, we show how we have used the Goal Structuring Notation (GSN) [6] – a graphical notation developed at York for the presentation of safety arguments – to outline a preliminary safety argument. The principal symbols in the notation are shown in Figure 1 (with example instances of each concept).

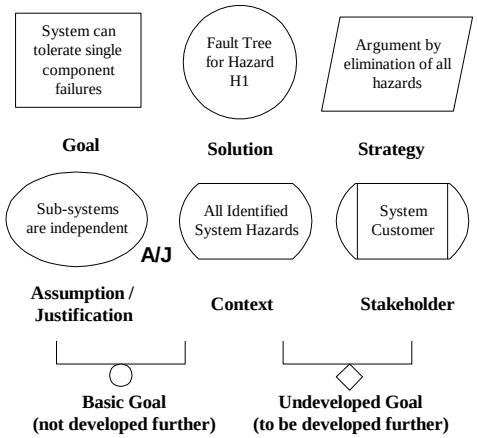

Figure 1 - Principal Elements of the Goal Structuring Notation

The purpose of a goal structure is to show how **goals** are broken down into sub-goals, and eventually supported by evidence (**solutions**) whilst making clear the **strategies** adopted, the rationale for the approach (**assumptions, justifications**) and the **context** in which goals are stated. For further details on GSN see [6].

5. Example: Preliminary Safety Argument for a Distributed Aero-Engine Controller Platform

This section describes the preliminary safety argument for a distributed aero-engine controller architecture. Traditionally engine controllers have been based on a centralised uni-processor approach, with direct-wired electrical cabling to all engine sensors and actuators. There are potential cost and weight savings that can be achieved through adopting a distributed approach – by using 'smart' sensors and actuators and a common databus rather than many individual cables. In addition,

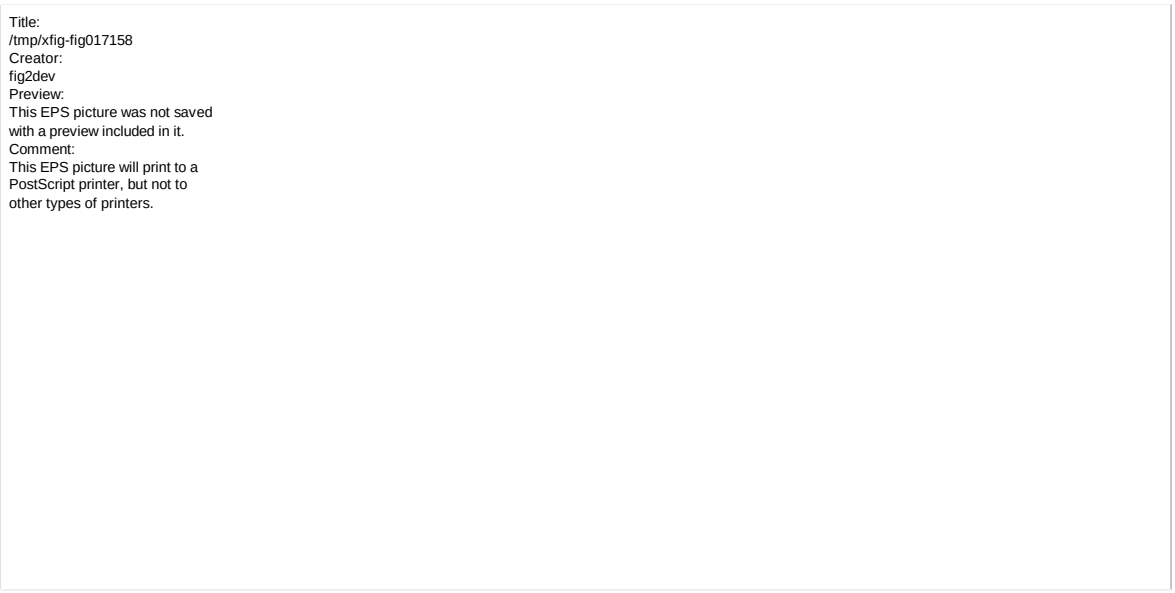

Figure 2 - Subsystem Structure

distributed processing units can provide additional flexibility and scalability in implementing the core controller functions. However, the distributed approach is *new* and therefore attracts particular scrutiny in certification (as is commonly the case for novel concepts in the aerospace sector). For this reason, it has been especially important to have clearly defined, from the earliest possible opportunity, how we would argue the safety of this platform.

In this paper, we purposefully provide a simplified overview of the distributed approach, as our principal aim is to discuss the safety case. There are many complex issues, such as vote synchronisation and processor state recovery, that are outside the scope of this paper. In describing the architecture, the following two terms are used:

- **Component** – a device that performs some function
- **Subsystem** – a configuration of replicated components performing identical functions (so that faults may be tolerated)

The proposed architecture consists of a number of subsystems that together would execute the software found on a conventional electronic engine controller. Figure 2 shows the top-level design of a single subsystem.

Each subsystem consists of the following elements:

1. **Voter** - An exact consensus voter that compares the output values of three or more replicated components and can identifies failures value differences are present. In the event of an identified failure a reset signal is sent to the corresponding component. The component will restart, but will be taken out of service if several resets are required in quick succession. When a component is recognised as being out of service then the voter will no longer use its output.
2. **Processor** - The architecture places minimal restrictions on the specific microprocessors to be used (in order to support 'technology transparency' [7]). Provided the processors have comparable throughput they may be used within a single subsystem (as the voting logic and scheduling approach facilitates lock stepping). Processor tasks are scheduled using the fixed priority approach [8].
3. **Timing Watchdog** - A countdown timer that will detect processor timing overruns.
4. **Local Memory** - Dedicated memory for each processor to provide a greater degree of isolation between replicated components.
5. **Local Clock** – Dedicated real-time clock for each processor that can be read and updated.

A Controller Area Network (CAN) [9] databus is used for carrying messages between subsystems and the smart engine sensors and actuators. Messages are scheduled using the fixed priority scheduling. In addition, at least one processor unit has a TDMA (Time Division Multiple Access) link to allow communication with the airframe. A global time base is maintained for all subsystems through synchronisation of local clocks across the databus [10].

For an aeroplane engine, the top-level hazards (such as 'deployment of thrust-reversers in cruise') are well understood within the industry. At the level of the architecture, we are concerned with those classes of

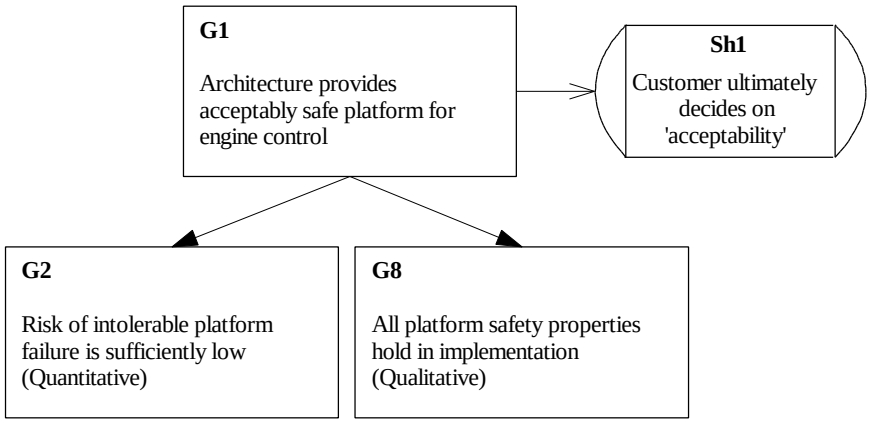

Figure 3 - Argument for Acceptable Platform Safety

failure mode that can give rise to hazards. To illustrate the principles of preliminary safety cases, we focus on these specific architectural level failure modes. (We believe it is possible to produce generic, reusable safety cases for such architectures, but discussion of such issues is outside the scope of this paper.) The classes of failure mode are:

- **Random Failures** – Even with the redundancy provided by replicated components, there remains a risk that random failures, originating from ageing or breakdown, may cause a system hazard.
- **Systematic Failure**s – Replication of identically implemented functionality will not protect against the following two forms of design error:
  - **Timing Failures** – Failure to meet hard real-time requirements and/or preserve functional ordering could result in a system hazard.
  - **Functional Failures** – Both transient and permanent errors in the control output of the subsystems, dependent on the situation, can result in system hazards. A transient failure, such as inadvertent thrust reverser actuation on an engine in flight, can have catastrophic consequences – as shown in the Lauda Air 767 disaster. The same error can have different consequences dependent on whether they are transient or permanent errors. For example, a *transient* error in fuel demand output is unlikely to cause a system hazard, namely engine overheat, due to the thermal mass of the engine. However, if the same error were *permanent* – the hazard could occur.

Random and timing failures are essentially 'architectural' issues. Functional errors, however, are predominantly defined by the application. Working at the architecture level, we were therefore only able to consider the overall function of fault-tolerance implemented within the elements of the architecture.

The top level of the safety argument (supporting the claim of acceptable safety) is represented in Figure 3. (In the goal structures presented, lines with filled arrowheads indicate the central chain of reasoning in the argument, lines with open arrowheads indicate tangential references to contextual information). The goal structure first indicates that it is the Customer who 'owns' the top level ('acceptably safe') goal. It is they who will ultimately decide on whether the goal has been achieved. The argument is then broken down into two parts: a qualitative and quantitative part. The quantitative part argues that the risk of failure is acceptably low, represented in Figure 4. The qualitative part addresses whether the implementation of the architecture successfully meets the necessary safety properties, expressed in Figure 5.

The quantitative argument is shown in Figure 4. The overall failure rate requirement for aircraft loss due to engine failures is approximately $1x10^{-5}$ per flight hour. Of which, a budget of $1x10^{-6}$ failures per flight hour is allocated to the engine control system. To ensure the introduction of systematic errors is appropriately minimised the system will be developed to Development Assurance Level A (defined by the civil aerospace standard DO-178B [11]).

The qualitative argument that the safety properties of the system hold is more complex. There are two aspects to the argument, shown in Figure 5, which address the functional and non-functional safety properties of the system. The non-functional safety properties of the system concern the timing and resource behaviour. (Resource exhaustion was identified as a potential cause of both timing and application function failures.) Experience shows the correct resource requirements are difficult to predict, which frequently leads to reworks being carried out to increase resource capability, or to optimise the use of resources. Our technique for addressing this problem is to make the architecture scalable, allowing extra subsystems to be added with the minimum of effort.

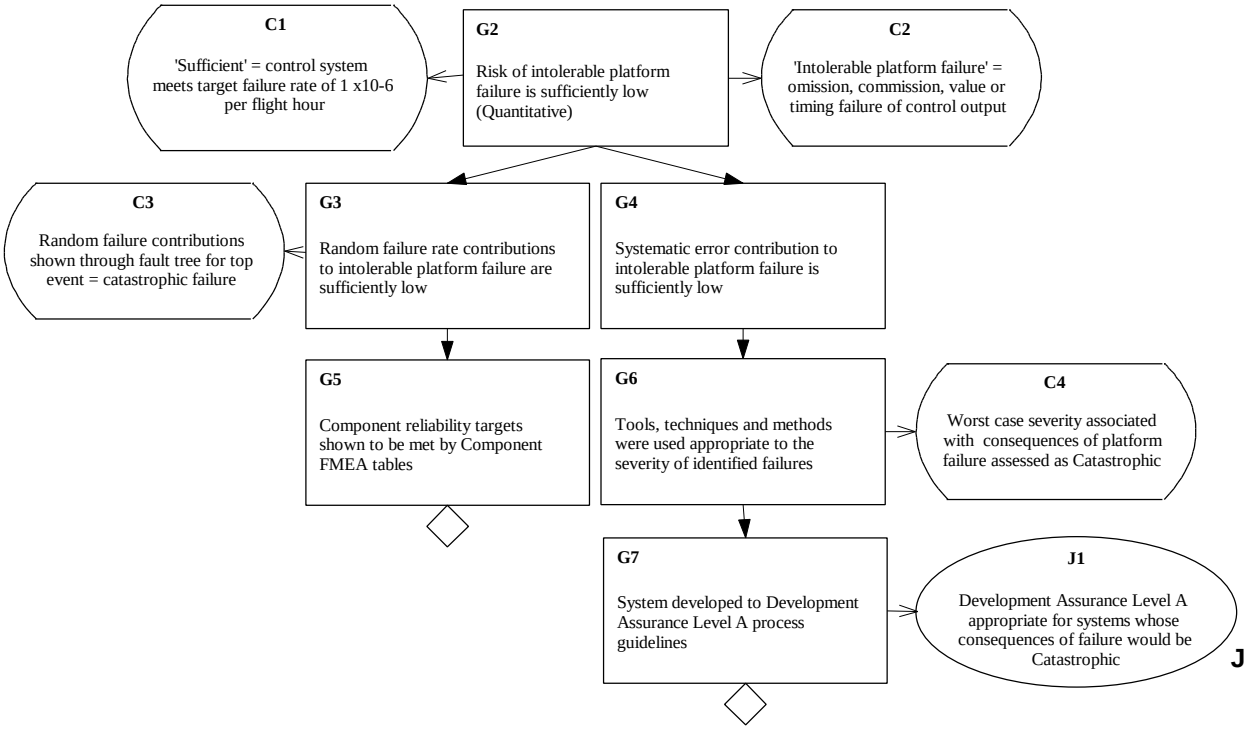

Figure 4 - Argument for Sufficiently Low Risk

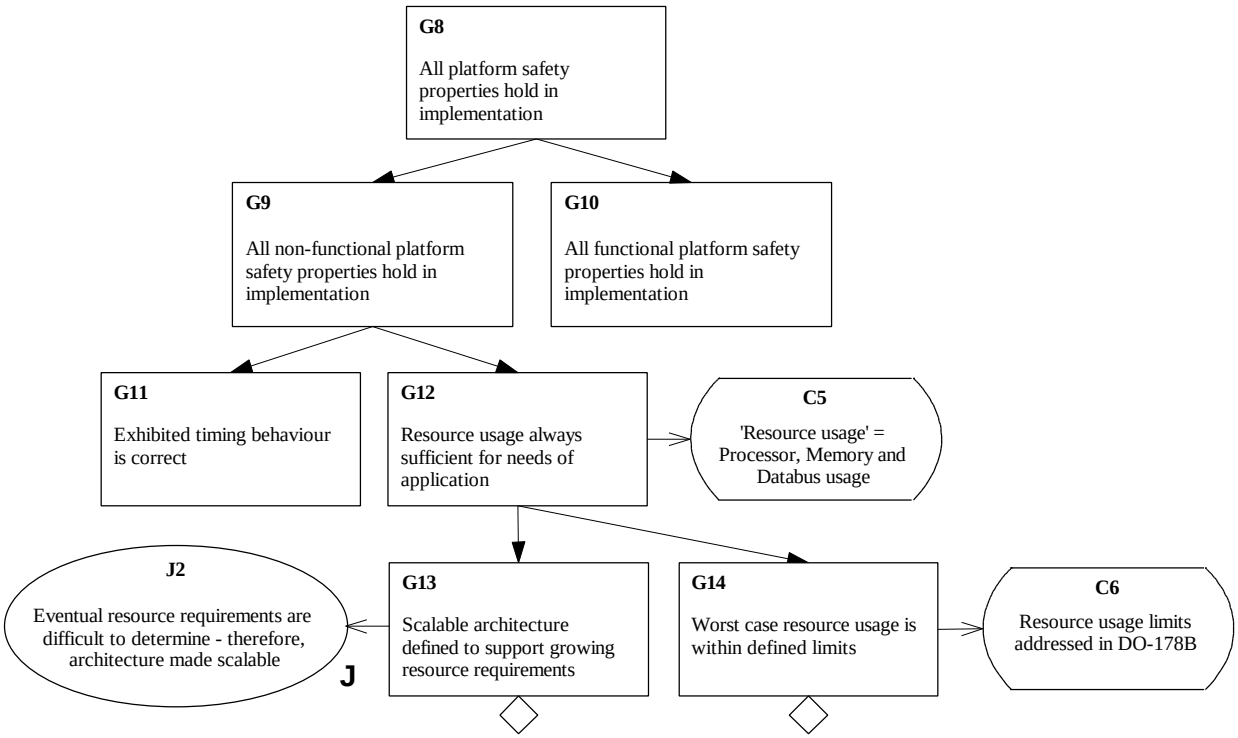

Figure 5 - Argument for Platform Safety Properties

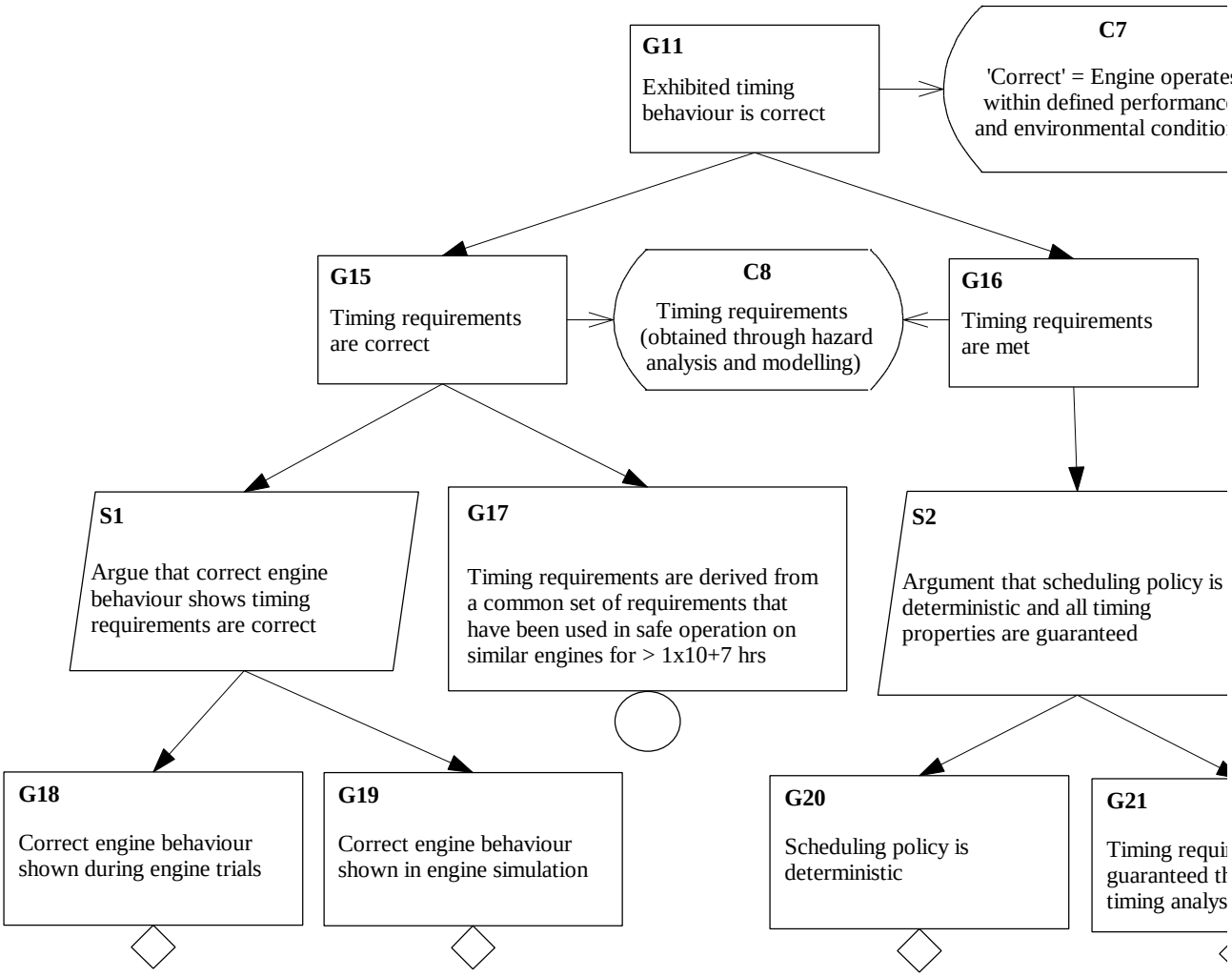

Figure 6 - Argument for Correct Timing Behaviour

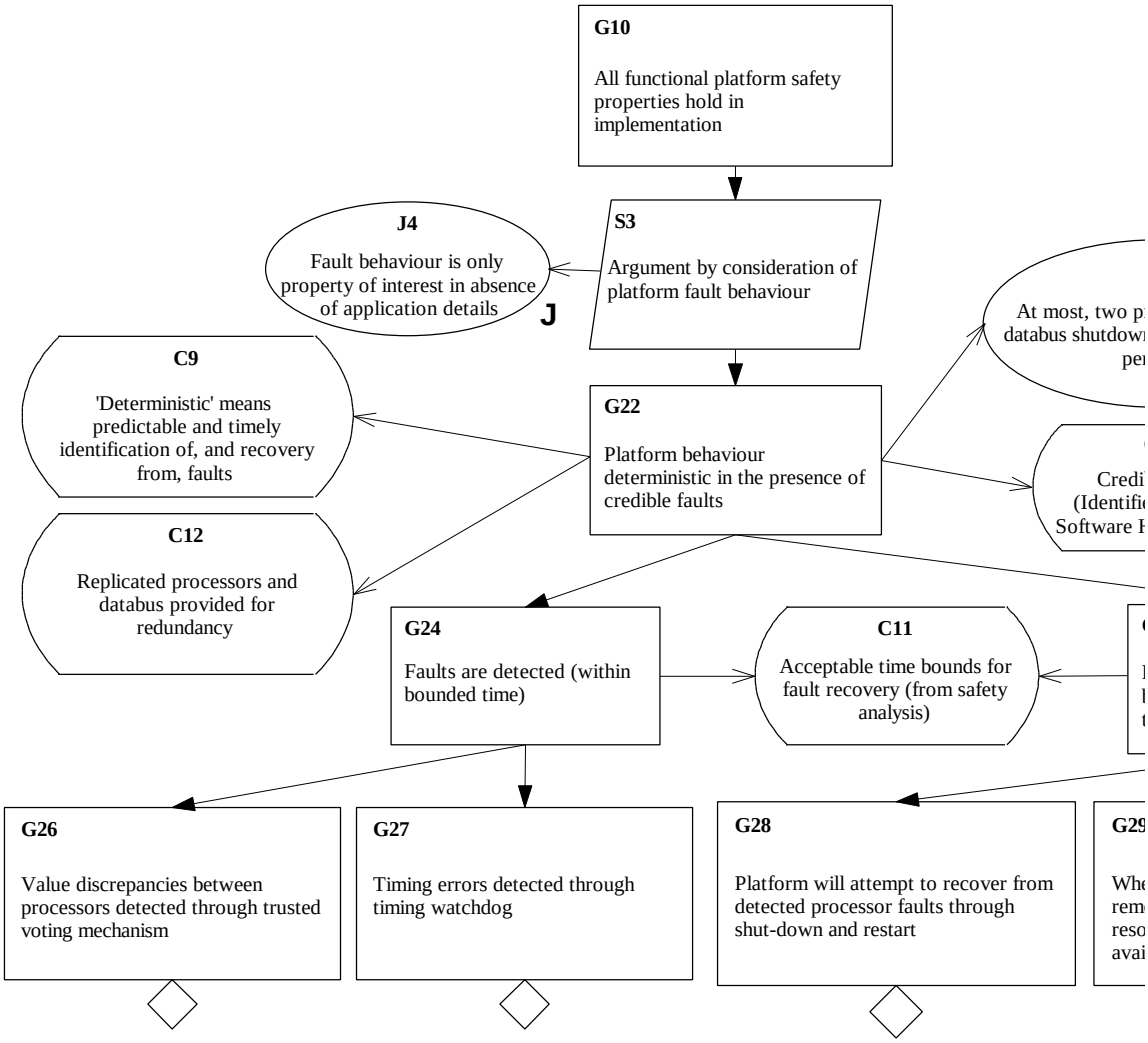

Figure 7 - Argument for Functional Platform Safety Properties

The argument of timing behaviour correctness (shown in Figure 6) consists of two parts: whether the timing requirements are correct and whether the requirements are satisfied. The timing requirements come from two sources, most are historical values related to the control loops of the engine which are known to provide stable and effective performance. The relevancy of the control timing requirements to this particular project is first checked using extensive simulation of an engine model, and later through large amounts of engine trials in all operation envelopes and conditions. In addition, there are design-derived requirements obtained through the hazard analysis process. An example of this type of requirement is shown in Figure 7 through goals G24 and G25, where a time bound for fault recovery is defined to reduce the period the architecture is at risk to additional errors. To verify that the timing requirements are met requires a deterministic scheduling policy, which allows appropriate analysis to be performed. Our solution is to use non pre-emptive fixed priority scheduling, which has a firm mathematical basis and determinable control flow.

The final part of our argument is shown in Figure 7, and predominantly concerns the fault-tolerance behaviour of the platform. The aim is to have a platform that operates deterministically even in the presence of faults. The faults are to be identified and recovered, where possible, within a bounded amount of time (in order that overall timing requirements can be guaranteed). Value and timing errors are identified in separate ways, but handled in the same manner. Value errors are identified using the trusted voting mechanism. A triplex processor architecture has been initially proposed as this will allow the voter to additionally identify the *source* of detected errors. For commercial reasons, related to the weight of cabling, only two databuses will be provided. However, the CAN databus is considered to be highly fault tolerant in its own right with the ability to withstand a wide variety of single and multiple errors. Timing errors are identified using the timing watchdog. Recovery from detected processor faults is attempted by restarting the offending processor. Where recovery from failures is not possible, the offending component is taken out of service.

Within this section we have briefly presented a preliminary safety argument which has derived a number of architecture dependent criteria to be achieved, if the system is to be safe. The undeveloped goals in our safety arguments represent the criteria for judging the appropriateness of any architecture under consideration. The criteria could be met by a number of different architectural combinations. For example, the component reliability requirement may be achieved using either an ultra-reliable component, or a network of replicated components. The manner in which the requirements are satisfied will be part of the developing system design and will be presented in the final (operational) safety case. Production of the preliminary safety case has increased confidence that the final certification case can be made.

6. Conclusions

Building a Preliminary Safety Case can be a worthwhile exercise both for the purposes of reaching agreement of the safety justification approach with the customer and for identifying, early on in a project, the key safety objectives and constraints to be addressed. In producing the Preliminary Safety Case for a distributed aero-engine controller we found the Goal Structuring Notation a useful basis for mapping the principal elements and structure of the *Preliminary Safety Argument*. By evolving this structure in parallel with development of the architecture, certification concerns were addressed as an integral part of the design process and safety features were built into, rather than 'bolted on' to the design. In our experience, such an approach can help to reduce the risk of having to perform large amounts of rework in order to obtain system certification.

7. Acknowledgements

The authors would like to acknowledge the financial support given by Rolls-Royce plc for the work reported in this paper.

8. References

[1] R. J. Cullen, "Safety as a Design Tool", in *Proceedings of the 1996 Safety and Reliability Society Symposium*, Swindon, U.K., Edited by R. F. Cox, Safety and Reliability Society, October 1996

[2] *Defence Standard 00-56 (Issue 2): Safety Management Requirements for Defence Systems*, U.K. Ministry of Defence, December 1996

[3] *JSP430: Ship Safety Management System Handbook – Volume 1 Issue 1 – Policy and Guidance on MoD Ship and Equipment Safety Management*, U.K. Ministry of Defence, January 1996

[4] *Defence Standard 00-55: Requirements for Safety-Related Software in Defence Equipment*, U.K. Ministry of Defence, July 1996

[5] *Interim Defence Standard 00-58 Issue 1: HAZOP Studies on Systems Containing Programmable Electronics*, U.K. Ministry of Defence, July 1996

[6] S. P. Wilson, T. P. Kelly and J. A. McDermid, "Safety Case Development: Current Practice, Future Prospects" in Proceedings of 12th Annual CSR Workshop, Bruges, Belgium 1995, Springer-Verlag

[7] R. A. Edwards *ASAAC Phase I Harmonized Concept Summary*, ERA Report 94-0973, ISBN 0 7008 0587 7, Paper 10.3, ERA Technology 1994 Avionics Conference and Exhibition, London, 30 November / 01 December 1994

[8] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM 20(1) pp40-61, 1973

[9] *Road Vehicles – Interchange of Digital Information – Controller Area Network*, ISO 11898, International Standards Organisation, 1993

[10] H. Kopetz and W. Ocheneiter, "Clock Synchronisation in Distributed Real-Time Systems", IEEE Transactions on Computers, 36(8) pp933-940, August 1987

[11] *RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics, December 1992