Contract-Based Justification for COTS Component within Safety-Critical Applications

Fan Ye, Tim Kelly
Department of Computer Science
University of York
York, YO10 5DD, United Kingdom
{fan.ye, tim.kelly}@cs.york.ac.uk

Abstract

Commercial-Off-The-Shelf (COTS) software components are being used within complex safety-critical applications. However, to use them with confidence, it is necessary to ensure that potential failures of the COTS component does not contribute to system level hazards. To this end, we have established a contract-based approach to capture the application-specific safety requirements, and corresponding assurance requirements, derived for a potential COTS component. This "contract" can be used to form the basis of a packaged safety argument (i.e. a safety case) for the component. This COTS component safety case (or safety case module) can then be used to form *part* of an overall system safety case. Using the previously developed concepts of compositional safety case construction (Kelly 2003), we describe the activity of matching application level safety objectives and assurance requirements to those claims and levels of assurance that can be established for the COTS component. The role of argument mitigation strategies is described for those situations where direct matches cannot be achieved. An example derived from an industrial COTS-based application is used to illustrate the approach.

Keywords: COTS, Safety-Critical, Safety Case Contract.

1 Introduction

COTS software products have been increasingly used in building complex safety-critical applications because of the perceived potential savings on development cost and time. However, the presence of COTS components can pose a significant challenge to the certification of the final safety-critical system.

It is a mandatory requirement for a safety-critical system to be certified prior to its deployment (Anderton et al. 2001). Certification is to demonstrate that the system is "safe" to operate. This is often done by means of arguments based upon the evidence about the system and its development process. These arguments (together with supporting evidence) are typically referred to as a "safety case". Whilst the evidence about a COTS component is essential in supporting system safety arguments, it may not be available due to the black-box nature of the COTS component.

It is clear from the above discussion that, to use a COTS component with confidence, it is necessary to ensure that, the impact of component behaviour on system safety is fully understood and managed. More specifically, it should be ensured that the potential failures of the COTS component cannot contribute to system level hazards. This holds the key to the successful safety case construction hence the justification of a safety-critical application integrated with COTS functionality.

In this paper, we describe a contract-based approach to the justification of COTS components within safety-critical applications. The approach starts by establishing detailed application-specific safety requirements and corresponding assurance requirements (used to indicate the level of assurance required for the satisfaction of each safety requirement depending on the significance of that requirement) derived for the expected COTS functionality. With these safety and assurance requirements, we can evaluate available COTS products and select an appropriate one through matching those application level claims and assurance requirements to the claims and levels of assurance that can be established for a COTS component. For situations where direct matches cannot be achieved, we examine how different mitigation strategies may be devised at the application context to ensure system safety. Once a successful match is made, a contract is recorded of the agreed relationship between the application and the COTS component. As the contract can be used as a direct basis for the subsequent compositional safety case construction, it is referred to as "safety case contract". A successfully established safety case enables us to justify the use of a COTS component within a safety-critical application.

An example derived from an industrial COTS-based application is used throughout the paper to illustrate the approach.

The rest of the paper is organised as follows: Section 2 briefly describes the example COTS-based system to be used throughout the paper; Section 3 introduces the concept of the safety case contract, and the derivation and specification of such a contract; Section 4 investigates how a final contract can be drawn up through direct matching or indirect matching via application mitigation strategies; Section 5 summarises the paper.

2 Background to the Illustrative Example

Within military aviation, due to spares shortage, environmental circumstances or urgent operational requirements, it is often necessary to operate aircraft with unserviceable safety-related modules. For example, an aircraft may be authorised to operate with faulty Navigation Lights provided that the aircraft is not operated at night. Service engineers are often required to assess the airworthiness of unserviceable aircraft with a view to continuing operations. Traditionally, this assessment has been done through a manual process by following some Go/No-Go list or Master Minimum Equipment Lists (MMELs). However, it is observed that, with the introduction of more complex and highly integrated avionics systems, the information contained within MMELs is no longer sufficient to support a safety-informed evaluation on the fitness of an unserviceable aircraft for continued operation (Henery 2001).

An 'operational' safety case (OSC) system is prototyped to assist the assessment of the airworthiness of unserviceable modern military aircraft (Henery 2001). The idea is to incorporate the fault tree analysis results for each system element (software components, hardware modules, and programmable units) documented in the aircraft system safety case to systematically evaluate the safety implication of flying an unserviceable aircraft. This involves using the fault trees to qualitatively evaluate the system level implication of the failure of one or more individual units, modules or components, and to quantitatively calculate the risk of such failures.

The system requires a database to hold the aircraft's system models and fault tree models presented in the safety case and to facilitate enquiries about the risks of the failure of one or more functions based on qualitative and quantitative analysis of the fault trees associated with the unserviceable system modules.

After some initial assessment, a proposal of building the prototype OSC system on top of a commercial database component is made. This provides us a good case for demonstrating the described approach. The preliminary system architecture is shown in Figure 1 (using Siemens Four-View Architecture model notation (Hofmeister, Nord, and Soni 2000)).

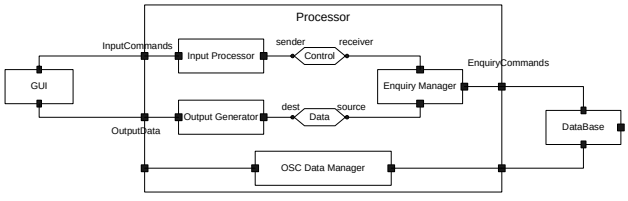

Figure 1: Operational Safety Case system architecture

The results of the case study are used throughout the paper to illustrate the described approach.

3 Contract Specification

In this section, the concept of the safety case contract is introduced first, and then the activities involved in specifying such a contract are described.

3.1 The Concept of Safety Case Contract

Safety Cases and the Assurance of Safety Arguments The concept of the "safety case" has already been adopted across many industries (including defence, aerospace, and railways). The purpose of a safety case is to communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context (Kelly and McDermid 1999). The three principle elements of a safety case are safety requirements/objectives, argument and evidence. The safety argument is that which communicates the relationship between the evidence and objectives. The development of a safety argument involves successively breaking down the top-level system safety objective into sub-objectives until a point is reached where claims can be supported by direct reference to available evidence.

Both strong and weak safety arguments exist, and this strength is based upon the extent to which the safety requirements have been satisfied by the evidence. Safety Assurance Levels (SALs), introduced by Weaver et al. (2003), can be used to examine the strength of a safety argument through evaluating the relevance of individual items of evidence and the completeness of the evidence set with respect to the safety objective. The SAL assigned to the top-level system safety objective indicates the required or target assurance of the argument, which is based upon the severity associated with the failure to meet the top-level objective. During the construction of an argument, as the top-level safety objective is successively broken down into sub-objectives, the SAL of a parent safety objective is apportioned across all its child element(s) (i.e. sub-objective or evidence) successively. The process of SAL apportionment enables one to reason about the required strength of an argument.

SAL apportionment across sub-safety objectives at different levels is described by the guidance set out in (Weaver, Fenn, and Kelly 2003). The demand for a defensible argument within a safety case prompts the need for the establishment of the required assurance level for a safety objective in addition to the identification of the safety objective itself. This is especially the case for COTS-based safety-critical system safety argument development where both application-specific safety requirements/objectives and assurance requirements are necessary for the evaluation and selection of an appropriate COTS product.

Compositional Safety Case In order to manage the complexity of safety case construction, system safety cases are often decomposed into subsystem safety cases. For example, a safety case concerned with the avionics of a complex military aircraft will be split into separate safety cases for separate systems (such as the navigation system, engine control and flight control systems). However, it is well understood that safety is a system

property, care must therefore be taken in how safety cases are divided up such that interactions are recognised and addressed within the safety argument. Based upon the above observations, Kelly (2003) formalised the concept of modular safety case and compositional safety case construction. These concepts can be usefully applied in our approach towards the justification of the use of COTS software components within safety-critical applications.

In the case of COTS-based safety-critical system development, the construction of a software system safety case can be naturally decomposed into at least two separate modules: an application safety case module and a COTS component safety case module. The interface of an application safety case module must explicitly specify the safety requirements to be satisfied by a COTS safety case module. Successful composition of these two safety case modules makes a complete safety case for the software system, hence justifies the COTS component use within the safety-critical application.

Two safety case modules can be usefully composed if their objectives and arguments complement each other – i.e. one or more of the objectives supported by a module match one or more of the arguments requiring support in the other. At the same time, an important side-condition is that the evidence and assumed context of one module is consistent with that presented in the other. Successful composition of an application safety case module and a COTS component safety case module requires that, the application-specific safety objectives and corresponding assurance requirements (i.e. SAL for each safety objective) derived for the expected COTS functionality (set out by the application safety case module) must be supported by the claims and levels of assurance that can be established for the COTS component.

Safety Case Contract With compositional safety case construction, where a successful match (composition) can be made of two or more safety case (argument) modules, a contract should be recorded of the agreed relationship between the modules. This contract aids in assessing whether the relationship continues to hold and the (combined) argument continues to be sustained if at a later stage one of the argument modules is modified or a replacement module substituted (which is likely for COTS component safety argument module due to component upgrades).

We propose the use of a safety case contract to capture the agreed relationship between an application argument module and a COTS component argument module. The contract must record an account of the match achieved between the objectives *required by* the application argument module and *addressed by* the COTS component argument module. In addition the contract must also record the collective context agreed as consistent between the participant modules. A proposed format for the safety case contract that covers each of these aspects is illustrated in Table 1.

Although a safety case contract should be drawn up upon the successful composition of two safety case modules, a partially specified contract (where only the application-specific safety objectives for the required COTS functionality are defined) can actually support the evaluation and selection of an appropriate COTS component. During the evaluation of a COTS component, safety claims and levels of assurance can then be established for the given application context. If direct matches can be made between the safety objectives set out by the application and the claims established for a COTS component, a perfect COTS component is selected, and a final safety case contract is drawn up, which justifies the use of the COTS component within the safety-critical application.

contract between application and COTS argument modules					
Application Safety Objectives for COTS			COTS Component Safety Claims		
Goal	*SAL*	*Context*	*Goal*	*SAL*	*Context*
(e.g. failure mode X of COTS shall not occur)	(e.g. SAL 3)	(e.g. operational assumptions for COTS)	(e.g. failure mode X will not occur)	(e.g. SAL 3)	(e.g. assumptions on how the component should be used)
…	…	…	…	…	…

Table 1: Format of the safety case contract

It is more than likely that direct matches will not be achieved. The black-box nature of a COTS component limits the visibility into the component, which subsequently constrains the claims and levels of assurance that can be established for the component from the evidence available. Thereby the established claims and level of assurances for a COTS component often do not match up to the expectations set out by the application. Under these circumstances, for a COTS component to be used, mitigation strategies will have to be devised at the application context to handle the mismatches. Matching of safety case modules and mismatches handling are discussed in more depth in Section 4. The task of specifying the partial safety case contract is examined in the following subsections (Sections 3.2 and 3.3).

3.2 Deriving the Application Safety Requirements

The proposed approach towards justification of a safety-critical application integrated with COTS components starts with specifying a partial safety case contract from the application context. This requires establishing the detailed safety requirements for the expected COTS functionality. The key activity for COTS safety requirements establishment, as described in (Ye and Kelly 2004a), is the component level criticality analysis. The criticality analysis is used to understand the impact of the failure of a COTS component with respect to system safety. Major activities involved in the criticality analysis are as follows (Ye and Kelly 2004b):

- **System Hazard Analysis** – identify all (significant) hazards at the system level and to further examine the severity of each identified hazard;
- **COTS Failure Mode Identification** – identify all the potential failure modes for the required COTS

functionality (i.e. a hypothetical component) that is derived from the system requirements;

- **Fault Tree Analysis** – construct a fault tree for each system hazard to establish the causal relationship between the system hazard and COTS component's failure modes, and use the minimal cut sets (Villemeur 1992) of the fault tree to identify the level of protection associated with each component failure mode;
- **Component Failure Mode Criticality Determination** – determine the criticality for each COTS component failure mode. After careful studying of the literature on software component risk analysis, Safety Integrity Level (SIL) determination (DSTO 1998, MoD 1996), and industrial practices on assigning Development Assurance Level (DAL) to software component/functionality (by following SAE 1996), we propose to use the following matrix (Table 2) as the basis for assigning the criticality to the COTS component's failure modes. "System hazard severity" and "degree of protection" are the two factors that are used to determine the criticality level of a component failure mode. There are four levels (1 to 4) of criticality with 4 as the highest level. For more detailed discussion on component failure mode criticality determination, see Ye and Kelly (2004b).

Component Failure Mode Criticality		**Degree of protection**		
		0	**1**	**2**
System Hazard Severity	**Catastrophic**	4	3	2
	Critical	3	2	1
	Marginal	2	1	1
	Negligible	1	1	1

(note: 0 degree of protection means single-point of failure)

Table 2: Component failure mode criticality matrix

Criticality Analysis Results for the Example System Described in Section 2

System Hazard Analysis: an obvious hazard is identified as "the advisory OSC system causes an unserviceable aircraft to operate with higher than acceptable level of risk", that is, the provided information misleads the service engineer to mistakenly believe that the risk of flying the unserviceable aircraft is acceptable. In the **worst case**, this could cause an aircraft to crash; hence the severity of consequences is **catastrophic**.

System hazard analysis also identifies that the OSC system could fail in some other ways. For example, the system could produce incorrect results indicating higher than actual risk, the system could fail to provide required information within the specified time limit, or the system could fail to produce any results (total failure of the system). However, all these three types of failures have not been regarded as system hazards because, in the worst case, the first type of failures would cause some operable aircraft to be unavailable; the last two types of failures would force service engineers to fall back to manual procedures, and in the worst case, this could means that some operable aircraft would not be available in time.

The identified system hazard and corresponding severity level are documented in Table 3.

ID	**Hazard Description**	**Severity**
A	OSC produces incorrect results indicating lower than actual risk associated with the failure of Function X	Catastrophic

Table 3: System hazard(s) and severity level(s)

Component Failure Mode Identification: after some initial study on the major functionality provided by several commercial database products, together with derived functional requirements for the database component from the system context, the essential functions for the potential COTS database component are identified as follows:

- **Add** Represents System Model (aircraft's architecture – systems, sub-systems, equipment, and functions that can contribute to system hazards) and its corresponding Fault Tree Model (fault trees) as data entries and stores them into the Data Store
- **Retrieve** Fetch data entry/entries from the Data Store
- **Remove** Eliminate data entry/entries from the Data Store
- **Edit** Modify data entry/entries and save it/them to the Data Store
- **Enquire** Performs enquiry over data entry/entries and produces qualitative and quantitative fault tree analysis results (including effect of failure of specific function(s), remaining risk mitigation, single or/and double points of failure, conditions for function to fail)

Through applying both FFA (Functional Failure Analysis) and Software HAZOPs (Hazard and Operability studies) methods, the failure modes for the potential database component are summarised in Table 4.

Fault Tree Analysis: based upon the previous results gathered from FFA and Software HAZOP, a fault tree is constructed for the identified system hazard (Figure 2).

The fault tree has not included the manual process using MMELs or Go/No-Go lists as a mitigation mechanism for the top-level hazard. Although as an advisory system, intervention by humans is possible, and theoretically the results produced by the OSC system could be verified by manual processes. However, as MMELs or Go/No-Go lists are designed to provide coverage for single point failures only, the results on the assessment of the impact of multiple unserviceable equipment provided by the OSC system cannot be easily verified by the manual process; further more, the comprehensive quantified analysis based on complex calculation over a large amount of data provided by the OSC system also means little can be checked by a manual process. It is observed that, in systems where timing is critical, or where operators may be unable to check the information, risk mitigation by manual process may not be feasible (DSTO 1998).

The minimal cut sets (for minimal cut sets determination see Villemeur 1992) of the fault tree illustrate that each basic event related to COTS database failure modes is a single point failure. That is, each individual basic event (the occurrence of a COTS database failure mode) itself can lead to the occurrence of the top event.

Function	Failure Mode	Local Effect	System Effect
Add	Fail to add	Data source incomplete	Hazard A
	Entry duplicated	Data source inconsistent	Hazard A
	Entry modified	Data source incorrect	Hazard A
Edit	Fail to update	Data source incorrect	Hazard A
	Incorrect entry	Data source incorrect	Hazard A
	Entry corrupted	Data source incorrect	Hazard A
Retrieve	Fail to retrieve	No/incomplete enquiry result	Hazard A
	Incorrect entry	Incorrect enquiry results	Hazard A
	Additional entry	Incorrect enquiry results	Hazard A
Remove	Fail to remove	Data source incorrect	Hazard A
	Incorrect entry	Data source incomplete	Hazard A
Enquire	Late	No local effect	Late results
	No result	No local effect	No result
	Incorrect results	No local effect	Hazard A
	Incomplete results	No local effect	Hazard A

Table 4: COTS database failure modes

Assigning Safety Criticality: by following the Criticality Matrix (Table 2), as severity of the top event is catastrophic, and the level of protection for each COTS component basic event is determined as 0, each offending failure mode of the database component is assigned a criticality level 4.

The criticality analysis reveals that, the impact of any failure of the potential COTS database with respect to system safety is significant, and there is no effective mitigation mechanism currently implemented to prevent such component failures from causing the top event.

In this way, we have determined both the functional and assurance requirements for the required database component.

3.3 Establishing Contract Terms from Application Requirements

It is straightforward to transfer the previously established application-specific safety requirements derived for a COTS component into a partial safety case contract:

- Safety objectives/goals – from dangerous COTS failure modes
- SALs – from the criticality level assigned to each COTS failure mode
- Context – from system architecture, operational environment and assumptions made about COTS component

Part of a partial safety case contract derived for the example system is illustrated in the following table.

Goals, SALs and Context matched between Application and COTS Argument Modules					
Application Safety Objectives for COTS			COTS Claims		
Goal	*SAL*	*Context*	*Goal*	*SAL*	*Context*
"fail to retrieve an entry" shall not occur	4	…			
"remove incorrect entry" shall not occur	4	…			
"return incorrect enquiry results" shall not occur	4	…			
…	…	…			

Table 5: Partial safety case contract

4 Contract Satisfaction

The safety requirements derived for the required COTS functionality (as in the partial safety contract) must be satisfied by a COTS component. The process of matching the application safety requirements to the claims that can be established for a suitable COTS component is referred to as contract satisfaction. In so doing, one needs to understand how an argument is satisfied or the types of argument support. In this section, a brief introduction on the types of argument support is given first, and then the contract satisfaction is investigated in terms of both direct match and indirect match via application mitigation strategies.

4.1 Type of Argument Support

Within an argument, a parent objective is supported by one or more child element(s) (i.e. child objectives or evidence). A child element on its own can either totally satisfy the entire parent objective or can partially satisfy the parent objective. Argument support provided by the child element set can have one of three forms (Weaver, Fenn, and Kelly 2003):

- Single support – one child element satisfies the entire parent objective
- Linked support – several child elements interdependently satisfies the parent objective
- Convergent support – several child elements each separately satisfies the parent objective

Each safety objective is associated with a SAL and a set of contextual constraints. Therefore support of a safety objective also involves satisfying its assurance requirement and the contextual constraints. The SAL indicates the minimum required assurance level for the safety objective to be met. The set of contextual constraints include the context from which the safety objective is established, assumptions made about the parent and child safety objectives, and known constraints on the satisfaction of the safety objective.

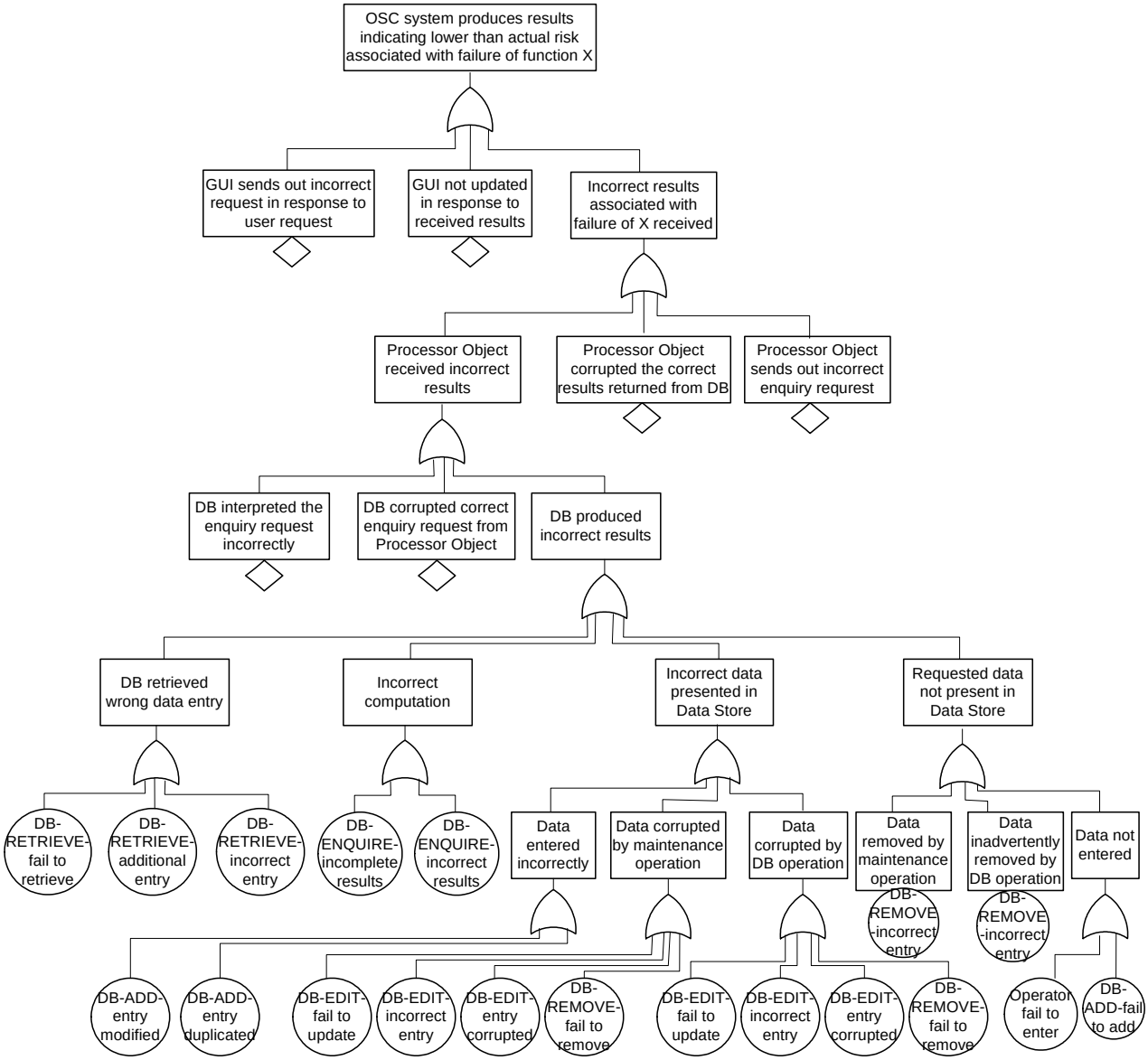

Figure 2: Fault tree for the OSC hazard

In the case of matching between the application safety argument module and a COTS component argument module, COTS component claims should provide total support for the safety objectives set out by the application. Hence an application safety objective and its corresponding assurance requirement can be viewed as a parent objective requiring support from one or more child elements – claims and levels of assurance established for the COTS component.

The satisfaction of a safety case contract requires the match of all the three elements of an argument between two contract parties. For a COTS-based safety-critical application, the contract satisfaction involves the match between a safety objective set out by the application and a set of safety claims established for the COTS component, and the match between the corresponding SALs and context.

The partial safety case contract specified from the application context can be satisfied in either of the two ways: direct support by the claims established for a COTS component, indirect match via the introduction of application mitigation strategies.

4.2 Direct Match

Direct match can be made where the safety objectives required from the application context can be directly supported by the claims established from a COTS component with evidence available. This also implies the match of the assurance requirements as well as the assumed context between these two parties. The diagram shown in Figure 3 illustrates such a direct match.

A few points can be made about the above direct match example:

- Within the application argument module (i.e. the upper part in Figure 3), the claim tree shows the causal relationship between a system level hazard and individual failure modes of system components (including COTS component and some other in-house developed components). Intermediate events and events related to failures of other system components

are represented as small blank rectangles. The claim tree is transformed directly from a fault tree (as in Figure 4).

- With a black-box COTS component, the claim of non-occurrence of a failure mode is established from the evidence (denoted as circles in Figure 3) available.
- The assurance requirements (e.g. SAL 3) attached to safety objectives within application argument module are established based upon the criticality analysis.
- The level of assurance established for a COTS component claim is based upon the *type* (e.g. direct, backing or reinforcement evidence) and *strength* (determined by the completeness, coverage and relevance) of the evidence available in supporting the claim (Weaver, McDermid, and Kelly 2002).
- With a component failure modes-oriented approach, direct matches only involve single support (i.e. the simplest form of argument support types).

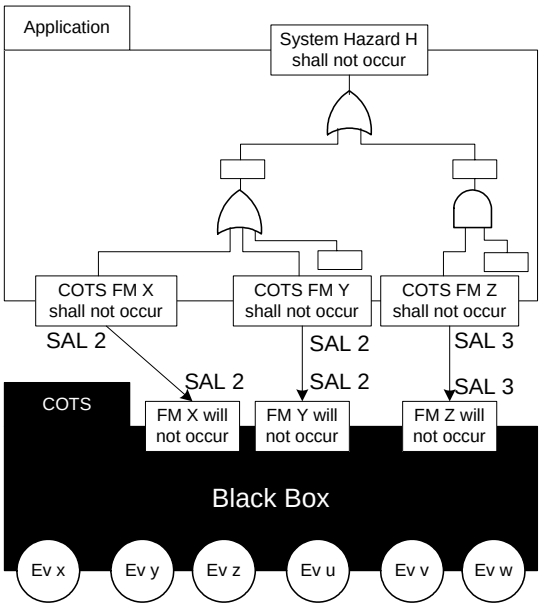

Figure 3: An example of direct match

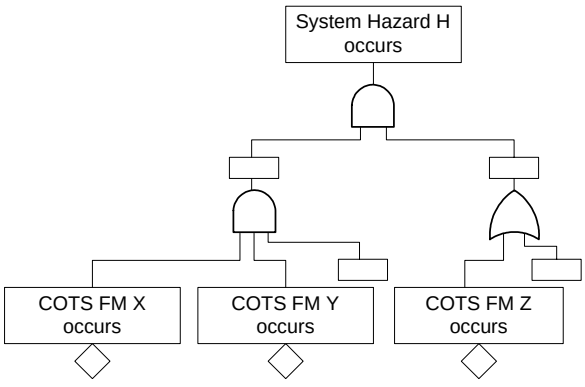

Figure 4: Fault tree for hazard H

4.3 Indirect Match via Application Mitigation Strategies

It is rarely the case that all safety requirements for the required COTS functionality derived from the application context can be perfectly supported by claims established from a COTS component. This is mainly due to the limited visibility that can be gained into the internals of a COTS component. Constrained by the limited visibility, available evidence – the basis on which COTS component claims and levels of assurance are established, is often limited (e.g. white-box evaluation evidence is often not available). This often results in claims established for a COTS component being narrower than the safety objectives set out by an application, and the levels of assurance established for the COTS component weaker than the required assurance levels.

One solution to the above-described goal and SAL mismatches is to acquire more evidence about the COTS component under consideration, so that a broader claim can be made and the claim can be assured more strongly to match up to the expectation of a safety-critical application. Some commercial vendors wishing to sell their component (e.g. a Real Time Operating System - RTOS) into the safety-critical marketplace are willing to offer "certification packs" (usually for an additional fee). These certification packs typically contain raw evidence (often process based). Evaluation carried out by a trusted third party is another source for additional evidence about a COTS component and bringing up the assurance level for COTS component claims.

When additional evidence cannot be obtained for a COTS component, mismatches can be dealt with by means of introducing mitigation strategies at the application level. Mismatch handling using application mitigation strategies are further discussed in the subsequent subsections: context mismatch handling, goal and SAL mismatch handling, and mitigation by failure types.

Context Mismatch Handling Within an argument, context attached to each safety goal represents the assumptions about the goal. These assumptions have to be true for the safety objective to be valid. Hence during the matching between application requirements and COTS claims, assumptions that each made about the other must be checked for their validity. Where there is a conflict among the assumptions that one made about the other, context mismatch arises. Context mismatches must be addressed before any matches between goals and SALs to be carried out.

Major context mismatches such as the cases of architectural mismatch described in (Garlan, Allen, and Ockerbloom 1995) may be extremely hard to handle. In this case, the COTS component under consideration should be rejected. Dealing with other minor context mismatches often involves adjusting assumptions (or contextual constraints) at one side to suit the other. This normally results in changes to the nature of the claim and its corresponding assurance level. For example, claims and levels of assurance based on "proven-in-use" evidence often need to be adjusted to reflect that the intended application context might have not been proven by the COTS component in actual use.

Goal and SAL Mismatch Handling Goal mismatch occurs when claims established for a COTS component are narrower than the safety objectives required by an application, and SAL mismatch happens when the level

of assurance established for the COTS component claim is weaker than the required assurance levels. Sometimes goal mismatch and SAL mismatch can happen at the same time. It may seem that goal mismatch and SAL mismatch are two distinct situations and should be dealt with differently. However, within an argument, a goal and its assurance level are strongly linked to each other and a trade off can be made between these two. Although a broad and bold claim like "COTS component failure mode X will never occur" can be made, it can only be weakly assured with the evidence available. With the same evidence, a narrower claim such as "COTS component failure mode X will not occur under system configuration Y" can be more strongly assured. Therefore goal mismatch and SAL mismatch are essentially the same problem and can be handled in the same manner.

The problem of goal and SAL mismatch can be generically represented as part 1 of Figure 5, in which the claim established for COTS component G_{COTS} is narrower than the objective required by the application G_{App} (denoted as $G_{App} > G_{COTS}$), and the assurance level for COTS claim (SAL n) is lower than that of application objective (SAL m) (denoted as m > n). In order to address this problem, trade offs must be made between the claim and its assurance level established for the COTS component, so that a goal match (part 2 of Figure 5) or a SAL match (part 3 of Figure 5) can be achieved first. Through applying application mitigation strategies, these two cases can then be dealt with using different argument support patterns.

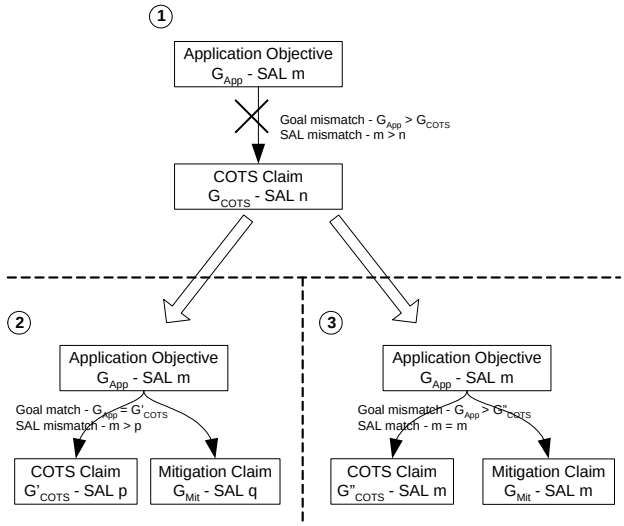

Figure 5: Mismatch handling

In the case of SAL mismatch as in part 2 of Figure 5 where goal match has been achieved, a mitigation strategy can be introduced to bring up the assurance level for the COTS claim to that which is required by the application safety objective. The COTS claim and mitigation claim each satisfies the applications safety objective but not with the required level of assurance. This forms a convergent support. The minimum assurance level for mitigation claim can be set by following the SAL apportionment guidance (Weaver, Fenn, and Kelly 2003). An example of using convergent support to address a SAL mismatch is illustrated in Figure 6[2]. In this example, the high assurance (i.e. SAL 4) application objective (i.e. AppGoal – absolute non-occurrence of a COTS failure mode) can only be satisfied by the COTS component claim (i.e. COTSClaim) weakly (i.e. SAL 2), by introducing a SAL 4 mitigation claim (i.e. MitClaim – detection and handling of failure mode occurrence), the overall application objective can then be fully satisfied.

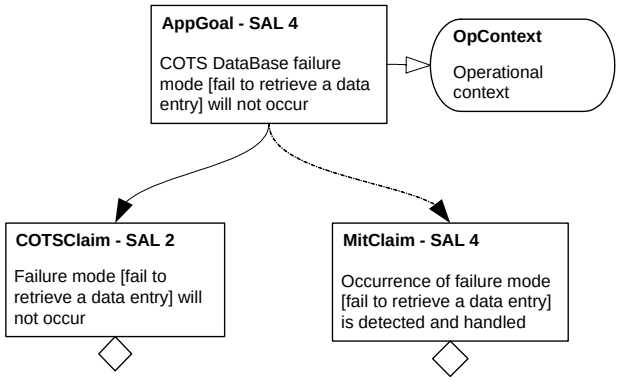

Figure 6: An example of resolving a SAL mismatch

In the case of goal mismatch as in part 3 of Figure 5 where SAL match has been achieved, a mitigation strategy can be introduced to provide a complementary claim so that together with the COTS claim, they provide complete support for the application safety objective. The COTS claim and mitigation claim provide a *linked support* to the safety objective set out by application. It worth noting that assurance level for mitigation claim must at least be the same as the required assurance level for application objective.

How different mitigation strategies may be used according to component failure types is discussed in the subsequent section.

Mitigation by Failure Types According to Pumfrey (2000), the failures of a software component as a service provider can be classified into the following three categories: service provision (omission, commission), service timing (early, late), and service value (detectably incorrect, undetectably incorrect). These failure types (i.e. omission, commission, early, late, value) have already been used as guidewords in HAZOP to prompt for the identification of software component failure modes. During contract satisfaction, hazardous COTS component failures need to be addressed by appropriate mitigation strategies. Although there are a number of mitigation strategies (or fault tolerance mechanisms) available, choosing the suitable one for mitigating a particular COTS component failure can be difficult. This motivates us to investigate how different mitigation strategies can be used to tackle different types of component failure.

When dealing with value failures (i.e. software provides incorrect results). A range of mitigation techniques can be

[2] The diagram is constructed using the Goal Structuring Notation (GSN) (Kelly 1999)

used to check the value produced by a COTS component and provide some appropriate corrective action if necessary. A simple coarse checking technique can be implemented to carry out some reasonableness check over the COTS results. This technique can detect most out-of-range or unreasonable value errors. “Control + Monitor” is the simplest form of design diversity, which involves using a COTS control component and an in-house developed monitor component. The monitor component implements a much simpler logic to provide a degraded service compared to the complex service provided by the COTS component. If the outputs from the two components are equivalent within some tolerance, the results from the COTS component are deemed correct and used by the system; otherwise the results from the monitor component will be used. N-version programming involves using N (N>2) versions of independently implemented software components to provide the same services or gradually degraded services. A majority voting mechanism is implemented to detect and mask value errors from faulty component(s).

Watchdog timer is commonly used to deal with failures related to service timing. It can detect whether a COTS task overrun its scheduled processing time, and the absence of outputs from a COTS component (omission type of failure). Where the absence of some output is deemed hazardous, a watchdog timer could be used to detect the absence of the COTS component output and to trigger the use of some other logic to produce an alternative output.

In the described case study (see Section 2), high assurance data (i.e. aircraft safety case data – system data model and fault tree data model) is to be stored and handled by a COTS database. In order to ensure the data integrity, message authentication codes (MACs) (e.g. CRCs, Hash Algorithms) could be applied to those data and checked by the application.

Wrapping is not a specific mitigation strategy. A wrapper is a specialised component inserted between a COTS component and its application environment to deal with the flows of control and data going to and/or from the component. Different mitigation strategies can be implemented into a wrapper. For example, reasonableness check logic, monitor, and watchdog timer can all be devised in a wrapper between a COTS component and its target safety-critical application to deal with different types of failures may arise from the COTS component.

Other generic fault tolerance principles such as partitioning and isolation should also be considered when designing the architecture of a COTS-based safety-critical application, as this may greatly help manage system safety integrity.

With the understanding on how different types of component failure can be addressed by individual, or a combination of, mitigation strategies, and understanding of cost-benefit of those different mitigation strategies, the task of contract satisfaction can be made easier.

5 Conclusion

The use of black-box COTS components within a safety-critical system often poses significant challenges to the system certification. For the successful use of COTS functionality, application-specific safety consideration must be taken into account seriously and early, and the impact of COTS component behaviour on system safety must be fully understood and managed. In this paper, we propose the use of safety case contract to capture the safety aspects of the functional relationship between a safety application and a potential COTS component. We also discussed how the safety contract helps COTS component evaluation and selection through investigating the matching between application requirements and COTS component claims.

We believe that the safety case contract established during COTS component evaluation and selection can serve the purpose for the final system certification. The safety case contract contains all three essential elements required for a defensible safety argument (i.e. safety objectives and assurance, argument structure, and evidence). A final safety case constructed from the safety case contract serves the purpose of justifying the use of COTS components within a safety-critical application.

6 References

Anderton, B., Armstrong, J., Frankis, D., Saddleton, D., Taylor, J., and Thombs, D. (2001): Can You Afford COTS Software in Safety Critical Applications? *In Proceedings of the 2nd European COTS User Group (ECUA) Workshop*, Orsay, Paris, France.

DSTO (1998): DEF(Aust) 5679 - The Procurement of Computer-Based Safety-Critical Systems. Australian Defence Standard.

Garlan, D. (1995), Allen, R., and Ockerbloom, J.: Architectural Mismatch: Why Reuse is So Hard. *IEEE Software*, Vol. 12, No. 6, pp. 17-26.

Henery, L. (2001): The Eurofighter 'Operational' Safety Case. MSc Thesis, Department of Computer Science, University of York, UK.

Hofmeister, C., Nord R., and Soni D. (2000): *Applied Software Architecture*. Addison-Wesley.

Kelly, T. P. (1999): Arguing Safety - A Systematic Approach to Safety Case Management. Ph.D. Thesis, Department of Computer Science, University of York, UK.

Kelly, T. P. (2003): Managing Complex Safety Cases. *In Proceedings of 11th Safety Critical System Symposium (SSS'03)*, Springer-Verlag.

Kelly T. P. and McDermid J. A. (1999): A Systematic Approach to Safety Case Maintenance. *In Proceedings of the 18th International Conference on Computer Safety, Reliability and Security (SAFECOMP’99)*, Toulouse, Springer-Verlag.

MoD (1996): 00-56 – Safety Management Requirements for Defence Systems. Ministry of Defence, Defence Standard.

Pumfrey, D. J. (2000): The Principled Design of Computer System Safety Analyses. Ph.D. Thesis, Department of Computer Science, University of York, UK.

SAE (1996): ARP 4761 – Guidelines and Methods of Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. The Society for Automotive Engineers.

Villemeur, A. (1992): *Reliability, Availability, Maintainability and Safety Assessment, Vol. 1 – Methods and Techniques*: Wiley.

Weaver, R. A., Fenn, J., and Kelly, T. P. (2003): A Pragmatic Approach to Reasoning about the Assurance of Safety Arguments. *In Proceedings of 8th Australian Workshop on Safety Critical Systems and Software (SCS'03)*, Canberra, Australia. P. Lindsay and T. Cant, (Eds.), Australian Computer Society.

Weaver, R. A., McDermid, J., and Kelly, T. P. (2002): Software Safety Arguments: Towards a Systematic Categorisation of Evidence. *In Proceedings of 20th International System Safety Conference (ISSC'02)*, Denver, Colorado, USA, System Safety Society.

Ye, F. and Kelly, T. P. (2004a): COTS Product Selection for Safety-Critical Systems. *In Proceedings of 3rd International Conference on COTS-Based Software System (ICCBSS'04)*, R. Kazman and D. Port (Eds.), LNCS 2959, Redondo Beach, CA, USA, Springer-Verlag.

Ye, F. and Kelly, T. P. (2004b): Criticality Analysis for COTS Software Components. *To appear in proceedings of 22nd International System Safety Conference (ISSC'04)*, Providence, Rhode Island, USA, System Safety Society.