Managing Complex Safety Cases

T P Kelly

Department of Computer Science

University of York, York, YO10 5DD, UK.

tim.kelly@cs.york.ac.uk

Abstract

Safety case reports are often complex documents presenting complex arguments. To manage the complexity of safety case construction, system safety cases are often decomposed into subsystem safety cases. In this paper we discuss the motivation and problems of partitioning the safety case, both as practiced historically, and as required in new modular, reconfigurable systems such as Integrated Modular Avionics. Recent work on managing safety cases "in-the-large" is presented. In particular, we demonstrate how notions of software and systems architecture design can be read-across to establish the concepts of "safety case architecture" and contract based reasoning for managing inter-safety case dependency. Problems of division of responsibility in safety case development will also be discussed.

1 Introduction

Safety case reports are often complex documents presenting complex arguments. Very rarely is it that safety cases are prepared by individuals. The reality is that the activity of establishing a safety case will be divided amongst a number of individuals, teams and, in some cases, organisations.

To manage the complexity of safety case construction, system safety cases are often decomposed into subsystem safety cases. Many examples of this can be observed in current safety-critical systems: System safety cases incorporate software safety cases (a division advocated by U.K. Defence Standards 00-55 (MoD, 1997) and 00-56 (MoD, 1996)). 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). The implied safety case for UK rail operations is made up of separate safety cases for station operations, infrastructure and rolling stock. However, it is well understood that safety is a system property. Extreme care must therefore be taken to ensure that safety case boundaries are drawn correctly, that arguments don't "fall between the gaps", and that formalising boundaries (e.g. through contractual agreements between organisations) doesn't prevent the development of efficient safety solutions.

Emergent safety properties, not dealt with by a "Divide and Conquer" approach to safety case construction, must also be addressed.

In the field of civil engineering if we compare the two cases of attempting to build a skyscraper and attempting to build a tree house we understand that fundamentally different approaches are required. Whilst it may be acceptable to start building the tree house with no clear idea of the overall design and by adding each piece at a time, building the skyscraper requires a clear view of the overall building architecture and guiding principles of construction. In the fields of systems and software engineering the role of *architecture* as a means of managing complexity and achieving emergent qualities such as modifiability is increasingly well understood.

In this paper we demonstrate how many of the principles of systems and software architecture can be carried across to the activity of safety case management in order to help manage complex safety cases.

2 Safety Case Architecture

In the field of software engineering, software architecture has been defined in the following terms (Bass et al., 1998):

> *"The structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them"*

Safety case architecture can be defined in very similar terms:

> *The high level organisation of the safety case into components of arguments and evidence, the externally visible properties of these components, and the interdependencies that exist between them*

Being clear of the externally visible properties of any safety case module (or 'component') allows us to appreciate its role within the overall safety case structure. The following can be regarded the key 'interface' properties for any safety case module:

1. Objectives addressed by the module
2. Evidence presented within the module
3. Context defined within the module

It is important to note that the definition of safety case architecture provided above gives equal importance to the dependencies between safety case modules (or 'components') as to the components themselves. This thinking must be at the heart of any attempt to decompose the safety case. Safety is not a "sum of parts" property. Dependencies must therefore also be recorded as part of any interface definition, perhaps along the following lines:

4. Arguments requiring support from other modules
5. Reliance on objectives addressed elsewhere
6. Reliance on evidence presented elsewhere
7. Reliance on context defined elsewhere

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. For example, the software safety argument is usefully composed with the system safety argument if the software argument supports one or more of objectives set by the system argument. At the same time, an important side-condition is that the collective evidence and assumed context of one module is consistent with that presented in the other. For example, an operational usage context assumed within the software safety argument must be consistent with that put forward within the system level argument.

Where a successful match (composition) can be made of two or more modules, a contract should be recorded of the agreed relationship between the modules. This is a commonplace approach in component based software engineering where contracts are drawn up of the services a software component *requires* of, and *provides* to, its peer components, e.g. as in Meyer's Eiffel contracts (Meyer, 1992).

There are many reasons why it is desirable to break down safety cases into smaller safety case modules for their development and presentation. Firstly, breaking down the safety case into separate components makes concurrent and separate development of the overall safety case (by different teams) possible.

Secondly, it can be useful for isolating change. It is desirable that when changes are required to the system safety argument (either because of challenging evidence, a changing design or a changing regulatory context (Kelly and McDermid, 2001)) we can avoid revisiting the entire safety case. In software component contracts, if a component continues to fulfil its side of the contract with its peer components (regardless of internal component implementation detail or change) the overall system functionality is expected to be maintained. Similarly, contracts between safety case modules allow the overall argument to be sustained whilst the internal details of module arguments (including use of evidence) are changed or entirely substituted for alternative arguments provided that the guarantees of the module contract continue to be upheld.

3 Requirements from Existing Safety Standards

Whilst the approach of 'modularising' the safety case may appear novel, in this section we will describe how the principles are implicit in a number of existing safety standards.

The U.K. Ministry of Defence Standards 00-55 (MoD, 1997) and 00-56 (MoD, 1996) talk of a division of a safety case across the system and software boundary.

Defence Standard 00-56 demands the production of a System Safety Case for all new systems, modifications of existing systems and non-development items. For safety-related software in defence equipment Defence Standard 00-55 requires the production of a *Software* Safety Case:

> *"**Part 1 Clause 7.1.1** The Software Design Authority shall produce a Software Safety Case as part of the equipment safety case defined in Def Stan 00-56."*

The role of the software safety case as forming part of the system safety case is clearly recognized. The software safety case must support the objectives of the system safety case. The standard reinforces this point in the following guidance provided in Part 2 Annex E:

> *"E.2.4 Relationship to System Safety Case*
>
> *E.2.4.1 Safety is a system property, and it is impossible to present safety arguments about software in isolation. The Software Safety Case should, therefore, be part of and subordinate to the System Safety Case. Traceability from the Software Safety Case to the System Safety Case should be provided for safety requirements and safety arguments."*

The principle described above of maintaining an account of the traceability between the system and software safety cases is in agreement with the notion of recording contracts between safety case modules as described in the previous section. The interface of any (sub) safety case must be clearly discernable and the 'contract' of agreement between this safety case and others clearly recorded. Defence Standard 00-55 describes further the form that this contract might take between the system and software case elements:

> *"E.3.6.1 The System Safety Case will also impose 'derived requirements' on the software and … the Software Safety Case should consist of a number of claims which link back to requirements and constraints imposed by the System Safety Case."*

Going further than the division of the overall safety case into system and software elements, 00-55 acknowledges the possibility of having to further subdivide the software safety case into separate elements, as shown in the following clause from Part 2 Annex E:

> *"E.2.5.1 For extremely large or complex software systems, it may be necessary to structure the Software Safety Case in line with the architecture of the software."*

We will describe later in section 6 some of the various 'styles' of structuring and subdivision of the safety case.

Moving away from the particular division of system and software safety cases, other examples of possible partitioning of the overall safety case can be seen in

other standards. For example, the U.K. MoD Ship Safety Management Code JSP 430 in stating the following requirement for the production of safety cases clearly acknowledges the need for multiple safety cases for any single platform:

> *"Separate Safety Cases and Safety Case Reports are to be available for all units of the ship that are capable of independent operation, i.e. landing craft, boats, aircraft, weapon systems etc."*

One might infer from this requirement that systems capable of independent operation implies the existence of independent safety cases. However, this will rarely be the case. Where systems are expected to operate as part of a coherent platform to provide a coherent capability, safety of any one element will often depend upon the safety of the other elements even if only because they define and share a common operating context and share common infrastructure (e.g. available human resource).

Outside of the defence context, the railway industry is another domain where the requirement for and existence of multiple interrelated safety cases is recognised. The Railway (Safety Case) Regulations 2000 (HSE, 2000) require that safety cases are submitted to the Health and Safety Executive (HSE) by railway operators (Infrastructure Controllers, Train Operating Companies, Station Operating Companies). The *implicit* overall safety case for safe U.K. train travel is made up of many safety cases for infrastructure, train operations and station operations. Safety cases for train operations will inevitably make assumptions of the infrastructure. Safety cases for station operators will inevitably make assumptions of the train operators. Boundaries may be drawn between these elements however interdependencies will always remain.

The European Railway Standard CENELEC 50129 (CENELEC, 1998) recognises the importance of recording the relationship between any single safety case any other safety cases to the extent that a section of the recommended safety case structure is reserved for this purpose. CENELEC 50129 talks of safety cases being structure into six parts:

- **Part One** – Definition of the System
- **Part Two** – Quality Management Report
- **Part Three** – Safety Management Report
- **Part Four** – Technical Safety Report
- ***Part Five*** *– Related Safety Cases*
- **Part Six** – Conclusions

Part Five of the safety case is given a dual role. Firstly, it should be used to record references to the safety cases of any subsystems or equipment on which the main

safety case depends. Secondly, it should be used to present an account of the evidence of satisfying safety conditions from other safety cases. Perhaps because of the complexity of trying to provide a "joined up" safety case in the railway operations context, as described above, CENELEC 50129 has taken the very enlightened view that "no safety case is an island" by making these explicit recommendations for Part Five.

CENELEC 50129 also supports the principles of establishing multiple related safety cases in stating that the following three different types of safety case can be considered:

- *A generic product safety case* provides evidence that a generic product is safe in a variety of applications
- *A generic application safety case* provides evidence that a generic product is safe in a specific class of applications
- *A specific application safety case* that is relevant to one specific application

Within this framework the UK Railtrack Engineering Safety Management Handbook (Railtrack, 2000) (commonly known as the "Yellow Book") talks of possible re-use of safety evidence. It gives the example of a specific application Safety Case for a re-signalling scheme referring to a generic application safety case for the use of a points machine in a particular type of junction which may in turn refer to a generic product safety case for that points machine. This can be thought of as effectively adopting a modular safety case approach whereby new safety cases are made through the composition of existing safety cases with *new* arguments and evidence.

4 Emerging Classes of Systems

The previous section has sought to highlight that the requirements for, and principles of, a modular approach safety case construction are not new in so much as the approach is already required in reasoning about "conventional" systems. However, the need for a modular safety case approach is made even more apparent when considering some of today's emerging classes of systems, including the following:

- Off-The Shelf Components designed for use in the Safety Critical Sector
- "Systems of Systems"
- Integrated Modular Avionics

In the following sub-sections we will highlight how each is an example area where safety cases must be thought of in modular terms in order to reliably reason about whole system safety.

4.1 Safety Cases for COTS Marketed into the Safety Critical Sector

For a commercial vendor wishing to sell their component (e.g. a Real Time Operating System - RTOS) into the safety-critical marketplace *modules* there is an obvious commercial and marketing advantage for them to obtain some form of component certification. However, system is a safety property and they are only marketing a component that will eventually form *part* of a system. It is therefore infeasible for them to produce a self-contained safety case and any claim about overall system safety or integrity should be viewed with scepticism.

The current response of some vendors to this problem is to offer "certification packs" (usually for an additional fee!) which contains the raw evidence (often process based) from which any system integrator can attempt to construct a case as to why the component has sufficient integrity to be used in a particular application. This situation is analogous to a builder giving a house purchaser the raw building materials and asking *them* to build the house.

Many purchasers of COTS components supposedly designed specifically for the safety-critical sector want more than a bag full of evidence – they would like some form of *partial* safety case. In such a situation the component vendor will attempt to attempt to envisage the claims of their component that will typically be demanded in any system safety case. For example, in the case of an RTOS, it is easy to imagine that the system integrator will need to make claims regarding non-interference between process memory spaces or regarding the integrity of the scheduling policy and implementation. The component vendor will then assemble, as far as is reasonably practicable, the arguments and evidence required to support these claims.

The collection of component claims and supporting arguments and evidence do not make a complete safety case for the component. Rather they form a partially constructed safety case with arguments in a "ready to use" form within a target application safety case. With the house purchasing analogy, this would equate to a purchaser being able to buy prefabricated sections of their house ready to be assembled and completed within the context of a particular house design.

These partial safety cases can be thought of safety case modules, along the lines being suggested in this paper, and should be similarly managed. It is important to expose clearly the claims being supported by these cases, the context assumed and the evidence presented such that there can be an aware and intelligent application of this information in a system safety case context.

4.2 Safety Cases for "System of Systems (SoS)"

One class of systems increasingly being discussed in the safety critical sector (especially in relation to defence systems) are "System of Systems". The Oxford English Dictionary defines a (single) system in the following terms:

"An organised or connected group of objects

A set or assemblage of things connected, associated or interdependent so as to form a complex unity"

The description "System of Systems" (SoS) can therefore be used to refer to an assemblage of *systems* connected, associated or interdependent so as to form a complex unity. Examples of SoS include integrated tactical battlefield systems with land, sea and air systems coordinating to reach an overall objective (with obvious potential for unsafe interactions) and the implicit SoS established for safe control of European airspace by the coordinated action of multiple national Air Traffic Control agents. With such systems it is not possible to look at any single part and ask, "Is it safe"? The safety of the component part depends upon how it contributes to the safety of the whole the safety of the peer components with which it interacts.

Again, the safety cases for component systems in a SoS context require management as modular safety cases.

4.3 Integrated Modular Avionics

Traditionally aircraft computer systems have been *federated* – each system provided on a number of dedicated hardware units. The containment and physical isolation offered by federated systems has supported separate safety analysis and safety justification. However, the aviation industry is increasingly wishing to move towards Integrated Modular Avionics (IMA). Using the definition provided by Rushby (Rushby, 1999), in IMA a single computing system (with internal replication to provide fault tolerance) provides a common computing resource to several functions.

Integrated Modular Avionics (IMA) offer potential benefits of improved flexibility in function allocation, reduced development costs and improved maintainability, it poses significant problems in the development of the safety case. A principal motivation behind IMA is that there is through-life (and potentially run-time) flexibility in the system configuration. An IMA system can support many possible mappings of the functionality required to the underlying computing platform.

In constructing a safety case for IMA an attempt could be made to enumerate and justify all possible configurations within the architecture. However, this approach is unfeasibly expensive for all but a small number of processing units and functions. Another approach is to establish the safety case for a specific configuration within the architecture. However, this nullifies the benefit of flexibility in using an IMA solution and will necessitate the development of completely new safety cases for future modifications or additions to the architecture.

A more promising approach is to attempt to establish a modular, compositional, approach to constructing safety cases that has a correspondence with the modular

structure of the underlying architecture. However, although the system architecture may be thought of as a neatly partitioned structure, safe overall system behaviour may not be so easily compartmentalized. IMA potentially suffers problems of poor fault containment. Owing to shared infrastructure (e.g. shared memory spaces, shared communications busses) the failures in one system element may propagate and cause failures in other system elements. Consequently, one must think very carefully about the structure of the overall safety case. Whilst parts of the safety case may directly correspond to system elements, other parts must explicitly address issues such as the integrity of the infrastructure and the intended and unintended interactions between elements. The need for a modular approach to safety case management in such cases is hopefully apparent. The architecture of the safety case must receive explicit and considerable attention.

5 'Hazards' in Safety Case Decomposition

The principle of decomposing system safety cases into subsystem safety cases may appear straightforward. However, there are a number of possible hazards in adopting this approach, including the following:

- **Failing to consider interactions between subsystem safety cases** – In the drive to "Divide and Conquer" the safety case we must be careful to ensure that at least one element of the safety case addresses the safety considerations posed by the interactions between elements.
- **Disproportionate effort allocated across overall safety case development** - It is important that a proportionate level of effort is dedicated to each subsystem safety case. It is unsatisfactory for one element of the overall safety case to be pursued in excess whilst others are under investigated or represented. When the safety case is taken as a whole a weak subsystem safety case may undermine other stronger dependent parts of the safety case.
- **Duplication of effort when apportioning responsibility** - This, of course, is not a safety problem but rather an efficiency problem. When dividing up the responsibilities of the subsystem safety cases care must be taken to clearly identify the scope and boundary of the subsystem safety case objectives.
- **Safety objectives "falling down the cracks" when apportioning responsibility** – This *is* a significant safety issue. It is all too easy when dividing up responsibilities among subsystem safety cases for one sub safety case to assume that another is tackling an objective and vice versa. Obviously, if unnoticed, an incomplete safety case will result.

An underlying cause of these problems is a poor understanding of the overall structure of the system safety case and safety argument. By explicitly planning (at an early stage in the lifecycle), representing, and managing the safety case architecture we can hope to alleviate some of these problems.

6 Safety Case Architecture Styles

In the field of software architecture there are a number of recognised architectural 'styles'[1]. These styles define patterns of component types and possible interactions and include examples such as "Data-Centric", "Data Flow", "Event-Driven" styles (Bass et al., 1998). They describe fundamental principles and constraints governing the organisation of the software architecture. In the same way it is possible to describe styles in the organisation of safety case architecture.

The simplest safety case architecture style is for the safety case decomposition into sub-safety cases to attempt to follow (as far as is possible) the system decomposition. This approach is suggested within Annex E of Part 2 of Defence Standard 00-55:

> *"E.2.4.2 Production of the Software Safety Case should follow the approach adopted for the System Safety Case. For example, one possible approach for a complex system might involve developing a top-level System Safety Case which is supported by lower level Subsystem Safety Cases. Each Subsystem which contains SRS should then have its own Software Safety Case. It may also be useful to prepare a top-level Software Safety Case which covers common aspects of all software in the system."*

This form of safety case division may be particularly appropriate when system components are being developed to different levels of integrity or assurance. This is also recognised by 00-55:

> *"{A} Software Safety Case for a large software system composed of large subsystems of differing safety integrity levels might be structured as:*
>
> *(a) a set of Subsystem Software Safety Cases justifying the safety of each software subsystem;*
>
> *(b) a top-level Software Safety Case justifying how the combination of subsystems is safe."*

As described earlier in section 4.3, this approach whilst being possible for conventional (statically defined) software systems would prove infeasible for highly configurable and adaptable modular systems such as IMA. Rather than reasoning about all possible combinations of systems (of which there will be many!) we require a safety case architecture exhibiting one or both of the following styles:

- **Reasoning about configuration blueprints** – A set of blueprints of possible system combinations, ideally covering *classes* of combinations is

[1] Architectural 'Styles' are synonymous with Architectural 'Patterns'

identified for the IMA system a priori. The safety case architecture, in addition to having a conventional subsystem safety case structure would also have a safety case element (or 'module') that argues the safety of the blueprint definition. Such a safety case architecture permits safe operation of any system configuration that can be said to fall within the system blueprint. Blueprints are discussed further in section 8.

- **A 'Backplane' Safety Argument** – the extent of required reasoning about possible interactions between system components can be limited if it is possible to establish a sound 'backplane' argument concerning the infrastructure of the modular system. Particularly, such an argument must support claims regarding the system partitioning. For example, if it is possible to establish an argument (for the infrastructure) that memory spaces are properly segregated and that possible interference via communications busses is predictable and manageable then this limits the extent to which *each* subsystem safety case (e.g. for each avionics application) must reason about such issues.

Rather than decomposition of safety case according to system structure another 'style' is to decompose the case according to safety *functions*. Functional division may cut across system (and therefore organisational) boundaries. Multiple systems may contribute to the performance of a particular safety function. Similarly, systems may contribute to the performance of more than one safety function. This style has one advantage over the subsystem decomposition style in that it promises to be more cohesive *from a safety perspective*. For someone wishing to look at all of the issues surrounding the performance of a particular safety function they will find them largely addressed within a single sub safety case. However, in systems that where safety functions are heavily integrated we run the risk of failing to address the safety issues associated with function interaction. To combat this, it is essential to ensure that either function interactions are considered within each sub function safety case (*good* for comprehensibility but *bad* for maintainability) or within a separate 'interaction' sub safety case (*less* comprehensible but *easier* to maintain).

In the same sprit as the functional decomposition style, it is also possible to decompose the safety case according the structure of identified hazards. This has obvious advantages for aiding understanding of safety. However, hazards cut across system and function boundaries and unless the work division and organizational structure for producing the safety cases can be divided upon along those lines this approach may be hard to realise in practice. In effect, hazard IPTs (Integrated Project Teams) would need to be created to support this approach. As with functional decomposition, we also need to ensure that interactions (in this case between hazards) are addressed somewhere within the overall structure of the safety case.

The styles described above are only a few of many potential ways of organizing the safety case structure “in-the-large”. Other examples include division across process argument and product argument boundaries, and organization according to styles of evidence (e.g. having Deterministic safety case elements distinct from Probabilistic safety case elements). As people become more familiar with, and practiced in, reasoning about safety case architecture more architectural styles will naturally emerge.

7 Representing Modular Safety Cases

The Goal Structuring Notation (GSN) (Kelly, 1997) - a graphical argumentation notation - explicitly represents the individual elements of any safety argument (requirements, claims, evidence and context) and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). The principal symbols of the notation are shown in Figure 1 (with example instances of each concept).

The principal purpose of a goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach (assumptions, justifications) and the context in which goals are stated (e.g. the system scope or the assumed operational role). For further details on GSN see (Kelly, 1997).

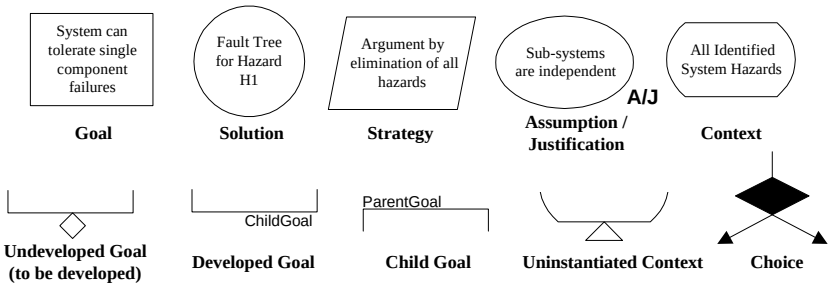

Figure 1 – Principal Elements of the Goal Structuring Notation

GSN has been widely adopted by safety-critical industries for the presentation of safety arguments within safety cases. However, to date GSN has largely been used for arguments that can be defined ‘stand-alone’ as a single artefact rather than as a series of modularised interconnected arguments. In order to make the GSN support the concepts of modular safety case construction it has been necessary to make a number of extensions to the core notation.

The first extension to GSN is an explicit representation of modules themselves. This is required, for example, in order to be able to represent a module as providing the solution for a goal. For this purpose, the package notation from the Unified Modelling Language (UML) standard has been adopted. The new GSN symbol for safety case module is shown in Figure 2 (Right Hand Side).

As has already discussed, in presenting a modularised argument it is necessary to be able to refer to goals (claims) satisfied within other modules. Figure 2 (left hand side) introduces a new element to the GSN for this purpose – the "Away Goal". An away goal is a goal that is not supported within the module where it is presented but is instead supported in another module. The Module Identifier (shown at the bottom of the away goal next to the module symbol) should show the unique reference to the module where support for the goal can be found.

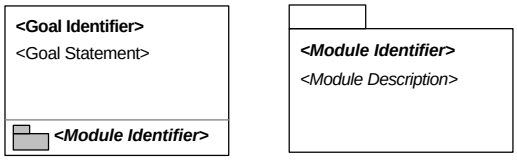

Figure 2 – GSN Elements Introduced to Handle Modularity

Away goals can be used to provide *support* for the argument within a module, e.g. supporting a goal or supporting an argument strategy. Away goals can also be used to provide contextual backing for goals, strategies and solutions.

Representation of away goals and modules within a safety argument is illustrated within Figure 3. The annotation of the top goal within this figure "SysAccSafe" with a module icon in the top right corner of the goal box denotes that this is a 'public' goal that would be visible as part of the published interface for the entire argument shown in Figure 3 as one of the "objectives addressed".

The strategy presented within Figure 3 to address the top goal "SysAccSafe" is to argue the safety of each individual safety-related function in turn, as shown in the decomposed goals "FnASafe", "FnBSafe" and "FnCSafe". Underlying the viability of this strategy is the assumed claim that all the system functions are independent. However, this argument is not expanded within this "module" of argument. Instead, the strategy makes reference to this claim being addressed within another module called "IndependenceArg" – as shown at the bottom of the goal symbol. This form of reference to a goal being addressed within another (named) module is called an "Away Goal". The claim "FnASafe" is similarly not expanded within this module of argument. Instead, the structure shows the goal being supported by another argument module called "FnAArgument". The "FnBSafe" claim is similarly shown to be supported by means of an Away Goal reference to the "FnBArgument" module. The final claim, "FnCSafe", remains undeveloped (and therefore requiring support) – as denoted by the diamond attached to the bottom of the goal.

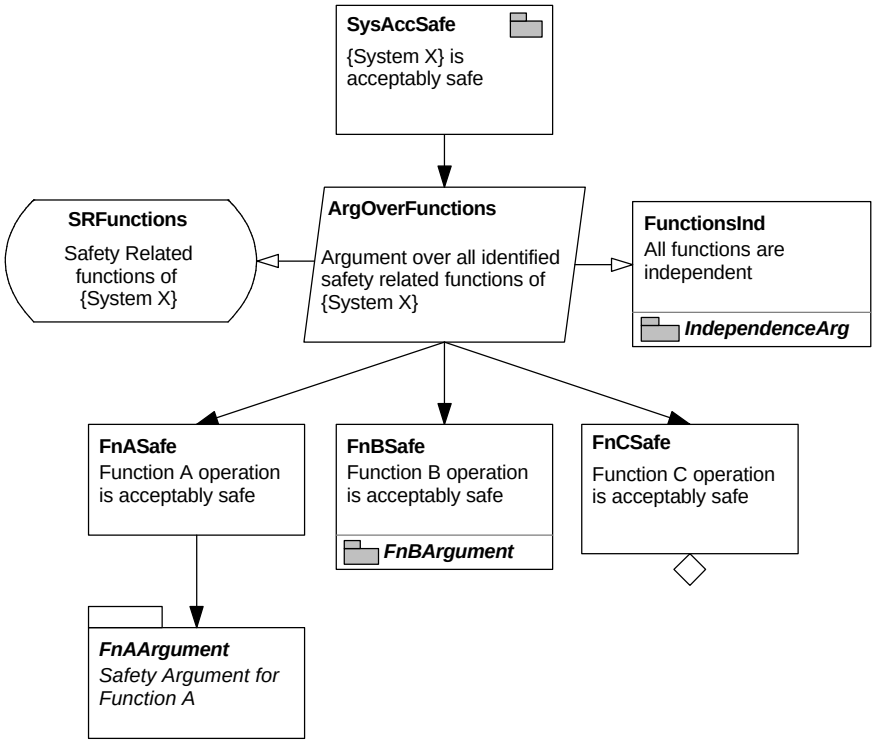

Figure 3 – Representing Safety Case Modules and Module References in GSN

In the same way that in can it be useful to represent the aggregated dependencies between software modules in order to gain an appreciation of how modules interrelate "in-the-large" (e.g. as described in the "Module View" of Software Architecture proposed by Hofmeister et al. in (Hofmeister et al., 1999)) it can also be useful to express a module view between safety case modules.

If the argument presented within Figure 3 was packaged as the "TopLevelArg" Module, Figure 4 represents the module view that can be used to summarise the dependencies that exist between modules. Because the "FnAArgument" and "FnBArgument" modules are used to support claims within the "TopLevelArg" module a supporting role is communicated. Because the "IndependenceArg" module supports a claim assumed as context to the arguments presented in "TopLevelArg" a contextual link between these modules is shown.

In a safety case module view, such as that illustrated in Figure 4, it is important to recognise that the presence of a *SolvedBy* relationship between modules A & B implies that there exists at least goal within module A that is supported by an argument within module B. Similarly, the existence of an *InContextOf* relationship between modules A & B implies that there exists at least one contextual reference within module A to an element of the argument within module B.

Alongside these extensions to the graphical notation of GSN, the following items of supporting documentation are required:

- **Interface declaration for each safety case module** – along the lines outlined in section 2, the external visible properties of any safety case module must be recorded – e.g. the goals it supports, the evidence (solutions) it presents, the cross-references ('Away Goal' references) made to / dependencies upon other modules of argument.
- **Contracts for composed modules** – where co-dependent safety case modules are used together within a system safety case a contract must be recorded of the dependencies resolved between the separate arguments.

Due to space limitations it is not possible to describe details of safety case module interface definitions or contracts. For further details see (Kelly, 2001).

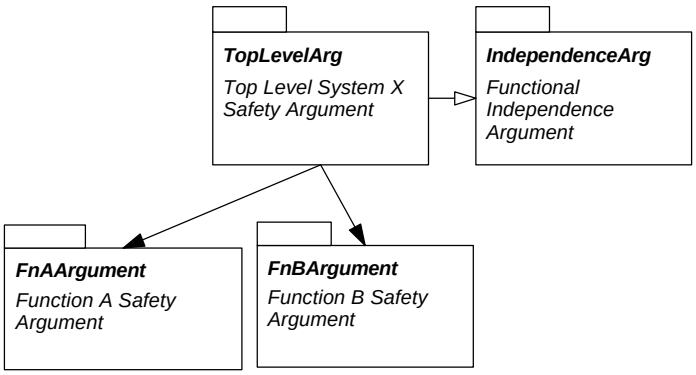

Figure 4 – Example Safety Argument Module View

8 Implications for Certification Processes

A modular approach to safety case construction has implications on the certification and acceptance processes. Whereas, traditionally certification has involved accepting, at a single point in time, a single safety case for an entire system for the benefits of a modular safety case approach to be realised requires a certification process that acknowledges the structure of a partitioned safety case that can be extended and modified without instantly requiring re-evaluation of the entire case. The guidance document ARINC 651 (ARINC, 1991) recognises this fact for suggests that for IMA-based systems the certification tasks are comprised of the following three distinct efforts:

- Confirmation of the general environment provided by the cabinet
- Confirmation of the operational behaviour of each function (application) intended to reside within a cabinet
- Confirmation of the resultant composite of the functions

ARINC 651 also recognises that conventional safety standards (such as DO178B (RTCA, 1992)) may need to be updated to reflect these new distinct tasks. ARINC 651 also talks explicitly of the need for "building block qualification" whereby it is possible to "separately quality certain building blocks of an IMA architecture in order to reduce the certification effort required for any particular IMA-hosted function". Example building blocks listed include specific arguments relating to the (ARINC 629) global data bus, the ARINC 659 backplane bus, the robust partitioning environment and the cabinet hardware / software environment. However, no detail regarding how these building block arguments are to be represented and managed is presented within ARINC 651.

In order to design and validate the various building blocks involved in IMA, ARINC 651 identifies the need for "rules which govern how the building blocks work together". It additionally describes that, "a feature of these rules of application is that they can be used to limit the work associated with certifying and re-certifying an IMA function to proof of compliance with the rules, and qualification of the function itself. Regulatory agency discussion is encouraged to establish how certification credit may be granted for adherence to these rules". This concept of defining rules between building blocks relates strongly to the principles of establishing well-defined module interfaces and contracts between safety case modules put forward within this paper. As the quote above clearly highlights, a necessary part of a new certification process based upon modular safety cases is to clearly give credit (i.e. limit the required re-certification) where contracts between safety case modules are upheld in the light of change to, or reconfiguration of, modules within the overall safety case.

9 Summary

In this paper we have described a key mechanism for managing safety case complexity – the concept of *safety case architecture* and (consequently) *modular safety cases*. Whilst the idea might appear new we have shown how it relates clearly to existing concepts of constructing whole system safety cases from a number of sub-system safety cases as described in current safety standards. To manage a modular safety case development *safely* we must be explicit about the interfaces of safety case modules, and acknowledge and represent clearly the dependencies that exist between modules. Extensions to the Goal Structuring Notation (GSN) for this purpose have been described.

Safety is not a "sum of parts" property. Care must therefore be taken in how safety cases are divided up such that interactions are recognised and addressed within the safety argument. Example hazards posed by safety case decomposition and styles of decomposition (safety case architectural 'styles') have been discussed. Finally, we have highlighted the possible implications for certification processes of adopting a modular safety case approach.

10 Acknowledgements

The author would like to acknowledge the financial support given by QinetiQ for some of work reported in this paper.

11 References

ARINC (1991) Design Guidance for Integrated Modular Avionics, Aeronautical Radio, Inc.

Bass, L., Clements, P. and Kazman, R. (1998) *Software Architecture in Practice*, Addison-Wesley.

CENELEC (1998) *ENV 50129 Railway applications - Safety related electronic systems for signalling*, European Committee for Electrotechnical Standardisation.

Hofmeister, C., Nord, R. and Soni, D. (1999) *Applied Software Architecture*, Addison-Wesley.

HSE (2000) *Railway Safety Cases - Railway (Safety Case) Regulations 2000 - Guidance on Regulations*, HSE Books.

Kelly, T. (2001) Concepts and Principles of Compositional Safety Case Construction (Contract Research Report for QinetiQ COMSA/2001/1/1), Department of Computer Science, University of York (available from www.cs.york.ac.uk/~tpk/pubs.htm)

Kelly, T. P. (1997) *A Six-Step Method for the Development of Goal Structures*, York Software Engineering.

Kelly, T. P. and McDermid, J. A. (2001) A Systematic Approach to Safety Case Maintenance, *Reliability Engineering and System Safety*, **71,** 271.

Meyer, B. (1992) Applying Design by Contract, *IEEE Computer*, **25,** 40-52.

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

MoD (1997) *Defence Standard 00-55, Requirements of Safety Related Software in Defence Equipment*, Ministry of Defence.

Railtrack (2000) Engineering Safety Management - Issue 3, Electrical Engineering and Control Systems, Railtrack

RTCA (1992) Software Considerations in Airborne Systems and Equipment Certification, RTCA

Rushby, J. (1999) *Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance (NASA Contractor Report CR-1999-209347)*, NASA Langley Research Center.