A Software Safety Argument Pattern Catalogue

R. Hawkins and T. Kelly

{richard.hawkins\tim.kelly}@york.ac.uk

Department of Computer Science

The University of York

Abstract

This document presents a catalogue of software safety argument patterns. The catalogue builds upon existing work and also takes account of current good practice for software safety, including from existing standards. The software safety argument pattern catalogue contains a number of patterns that may be used together in order to construct a software safety argument for the system under consideration.

The software safety argument patterns describe the nature of the argument and safety claims that would be expected for **any** *software safety case. The way the argument is supported may be different for each system but the 'core elements' of the argument (as defined by the patterns) remain.*

The effectiveness of the software safety argument patterns has been demonstrated through application to a number of case studies.

Contents

1. Introduction

Software safety argument patterns provide a way of capturing good practice in software safety arguments. Patterns are widely used within software engineering as a way of abstracting the fundamental design strategies from the details of particular designs. The use of patterns as a way of documenting and reusing successful safety argument structures was pioneered by Kelly in [1]. As with software design, software safety argument patterns can be used to abstract the fundamental argument strategies from the details of a particular argument. It is then possible to use the patterns to create specific arguments by instantiating the patterns in a manner appropriate to the application.

There exist a number of examples of safety argument patterns. Kelly developed an example safety case pattern catalogue in [1] which provided a number of generic solutions identified from existing safety cases. Although providing a number of useful generic argument strategies, the author acknowledges that this catalogue does not provide a complete set of patterns for developing a safety argument, it merely represents a cross-section of useful solutions for unconnected parts of arguments. Kelly's pattern catalogue does not deal specifically with any software aspects of the system.

The safety argument pattern approach was further developed by Weaver [2], who specifically developed a safety pattern catalogue for software. The crucial differences with this catalogue were firstly that the set of patterns in the catalogue were specifically designed to connect together in order to form a coherent argument. Secondly the argument patterns were developed specifically to deal with the software aspects of the system.

There are a number of weaknesses that have been identified with Weaver's pattern catalogue. First, the patterns take a fairly narrow view of assuring software safety, in that they focus on the mitigation of known failure modes in the design. Mitigation of failure modes is important, but there are other aspects of software assurance which should be given similar prominence.

Second, issues such as safety requirement traceability and mitigation were considered at a single point in Weaver's patterns. This is not a good approach; it is clearer for the argument to reflect the building up of assurance relating to traceability and mitigation over the decomposition of the software design.

Finally, Weaver's patterns have a rigid structure that leaves little scope for any alternative strategies that might be needed for novel technologies or design techniques.

A software safety pattern catalogue was also been developed by Ye [3], specifically to consider arguments about the safety of systems including COTS software products. Ye's patterns provide some interesting developments to Weaver's, including patterns for arguing that the evidence is adequate for the assurance level of the claim it is supporting. Although we do not necessarily advocate the use of discrete levels of assurance, the patterns are useful as they support the approach of arguing over both the trustworthiness of the evidence and the extent to which that evidence supports the truth of the claim.

The catalogue of software safety argument patterns presented in this document builds upon the existing work discussed above, and also takes account of current good practice for software safety, including from existing standards. The software safety argument pattern catalogue contains a number of patterns which may be used together in order to construct a software safety argument for the system under consideration.

The software safety argument patterns describe the nature of the argument and safety claims that would be expected for *any* software safety case. The way the argument is supported may be different for each system but the 'core elements' of the argument (as defined by the patterns) remain.

A number of case studies have been used to implement the software safety argument patterns [4]. These case studies have highlighted the benefits of utilising the patterns when developing software safety cases.

2. Software Safety Argument Pattern Catalogue Structure

The following argument patterns are provided in the software safety argument pattern catalogue:

High-level software safety argument pattern (Section 3.1) – This pattern provides the high-level structure for a software safety argument. The pattern can be used to create the high level structure of a software safety argument either as a stand-alone argument or as part of a broader system safety argument.

Software contribution safety argument pattern (Section 3.2) - This pattern provides the structure for an argument that the contributions made by software to system hazards are acceptably managed. This pattern is based upon a generic 'tiered' development model in order to make it generally applicable to a broad range of development processes and technologies.

Software Safety Requirements identification pattern (Section 3.3) - This pattern provides the structure for an argument that software safety requirements (SSRs) are correct and appropriate for each tier of the software design.

Hazardous contribution software safety argument pattern (Section 3.4) – This pattern provides the structure for an argument that hazardous errors are not introduced at each tier of software design decomposition. This includes arguing that mistakes have not been made in decomposing the design, and also that no *new* hazardous behaviour has been introduced.

Software contribution safety argument pattern with grouping (Section 3.5) - This pattern is an extension of the Software Contribution Safety Argument Pattern. It provides the option of grouping the argument to reflect natural requirements groupings in the software design.

When instantiated for the target system, these patterns link together to form a single software safety argument for the software.

The argument patterns are documented using the pattern extensions to the Goal Structuring Notation (GSN), described in Appendix A.

3. The Software Safety Argument Pattern Catalogue

3.1. High-Level Software Safety Argument Pattern

High-Level Software Safety Argument Pattern			
Author	Richard Hawkins		
Created	09/12/08	**Last modified**	08/06/09

INTENT

This pattern provides the high-level structure for a software safety argument. The pattern can either be used to create the high level structure of a 'stand alone' software safety argument considering just the software aspects of the system, or alternatively can be used to support claims relating to software aspects within a broader system safety argument.

STRUCTURE

The structure of this argument pattern is shown in Figure 1. Note that there are a number of different possible top goals for this pattern, as indicated by the public goals in the argument structure below.

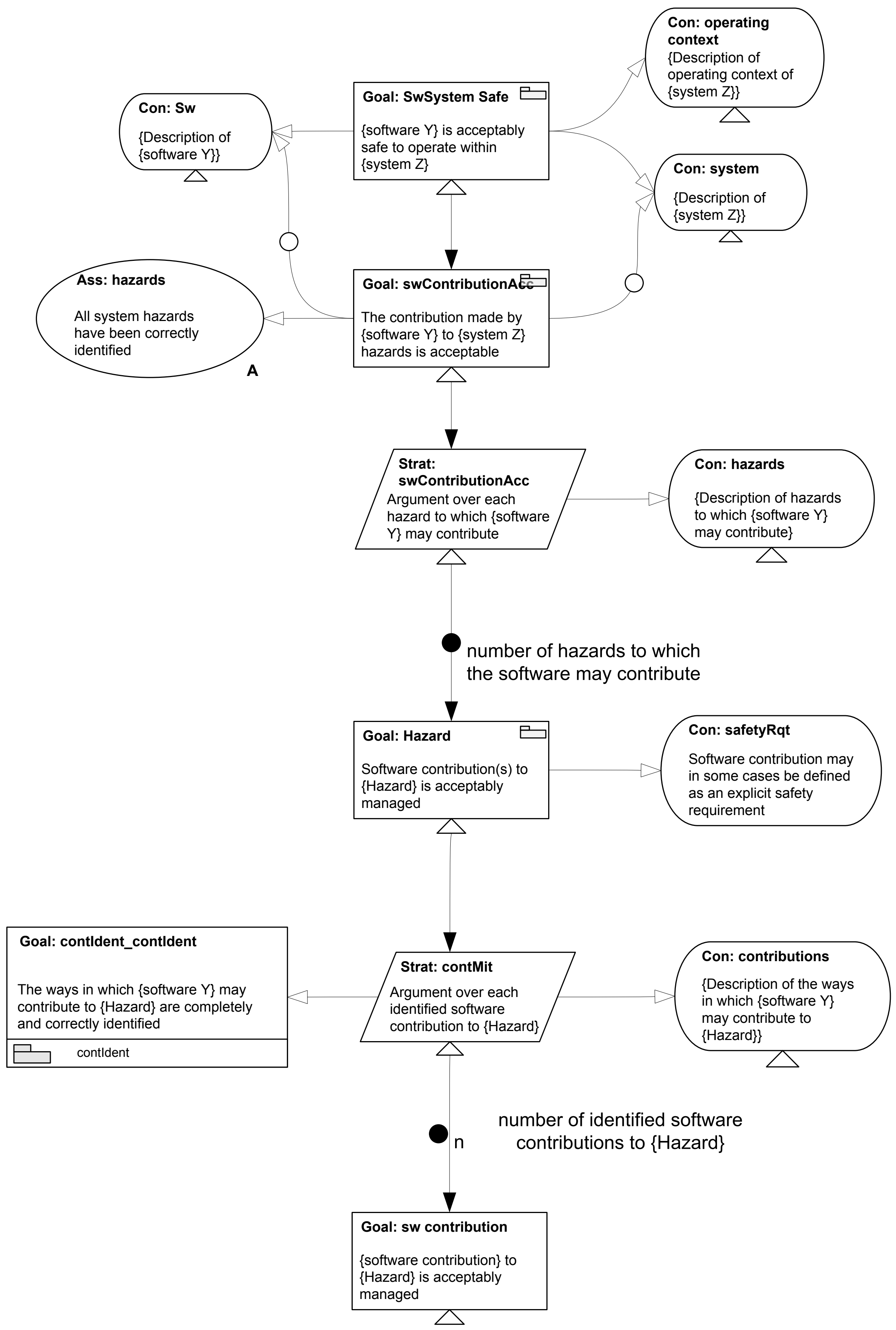

Con: operating context
{Description of operating context of {system Z}}
Goal: SwSystem Safe
{software Y} is acceptably safe to operate within {system Z}
Con: Sw
{Description of {software Y}}
Con: system
{Description of {system Z}}
Ass: hazards
All system hazards have been correctly identified
A
Goal: swContributionAcc
The contribution made by {software Y} to {system Z} hazards is acceptable
Strat: swContributionAcc
Argument over each hazard to which {software Y} may contribute
Con: hazards
{Description of hazards to which {software Y} may contribute}
number of hazards to which the software may contribute
Goal: Hazard
Software contribution(s) to {Hazard} is acceptably managed
Con: safetyRqt
Software contribution may in some cases be defined as an explicit safety requirement
Goal: contIdent_contIdent
The ways in which {software Y} may contribute to {Hazard} are completely and correctly identified
contIdent
Strat: contMit
Argument over each identified software contribution to {Hazard}
Con: contributions
{Description of the ways in which {software Y} may contribute to {Hazard}}
number of identified software contributions to {Hazard}
n
Goal: sw contribution
{software contribution} to {Hazard} is acceptably managed

Figure 1 – High Level Software Safety Argument Pattern Structure

PARTICIPANTS

Goal: SwSystem Safe

If a stand-alone software safety argument is being produced then this goal should be used as the top goal in the argument since it clearly sets out the overall objective of the software safety argument. It is necessary to provide the three items of context to make the scope of the software safety argument clear to the reader. This goal has been designated as a public goal to indicate that it may be used as the top goal in the argument.

Goal: swContributionAcc

This goal makes it clear that a hazard directed approach is adopted, by considering the contributions made by the software to the system's hazards. If the pattern is being used as part of a system safety argument, then this goal may provide the link in to that argument (hence a public goal). This would be the case if the system safety argument considers the contribution of the software all in one place. It is not necessary to include the context to provide descriptions of the system and the software if this is already clear from the system safety argument.

Ass: hazards

The system hazards can only be identified at the system level. Identification of system hazards is therefore outside of the scope of the software safety argument. It is acceptable therefore to make this assumption as long as the assumption is demonstrated elsewhere at the system level. If an argument to support this assumption exists with a system safety argument then it would be appropriate to link to that argument at this point instead of making an assumption.

Strat: swContributionAcc

To ensure traceability from the software to system hazards, the strategy adopted is to argue explicitly over each of the hazards identified at the system level.

Goal: Hazard

For each hazard there may be one or more potential contributions from the software identified at the system level. An instance of this goal is created for each of the system hazards to which the software may contribute. At the system level the software will only be considered from a 'black-box' point of view, so the contribution may be identified in the form of high-level functionality, or safety requirements. These contributions would be considered base events at the system level, and would not generally be developed further in a system level argument.

Goal: contIdent

It is necessary to ensure that all the software contributions are correctly identified at the system level. This is crucial to the assurance of the argument as it provides the warrant for the adopted strategy of arguing over the software contributions. This goal provides context to the strategy contMit and must be supported by an argument contained in a separate module (contIdent). Software contributions are often identified as base events in a fault tree analysis performed at the system level. The argument in module contIdent would, in such a case, reason about the rigour and suitability of that analysis.

Goal: sw contribution

An instance of this goal is created for each of the identified software contributions to each of the system hazards. The Software contribution safety argument pattern (section 3.2) may be used to generate an argument to support this goal.

APPLICABILITY

This pattern should be applied whenever a software safety argument is required as part of a safety case.

CONSEQUENCES

Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. Firstly 'Goal: sw contribution' must be supported. The Software contribution safety argument pattern presented in this catalogue can be used to support this goal. In addition, an argument to support 'Goal: contIdent' must also be developed in module contIdent. This argument will be based on analysis performed at the system level, so in some cases a sufficient argument may exist at the system level which can be used to support this claim.

IMPLEMENTATION

There are a number of different possible top goals for this pattern, as indicated by the public goals. The appropriate top level goal for the argument must be determined through consideration of the structure of any system safety argument which the software safety argument supports. If the pattern is being used to support a system level safety argument, the top goal from this pattern may not actually appear at the top of the overall argument structure. Instead it will appear as a child-goal within the system safety argument. It is important that a stand-alone software safety argument begins with the top goal 'Goal: swSystem Safe' to capture the overall objective of the argument and all the required contextual information.

POSSIBLE PITFALLS

The software contributions may not have been adequately identified at the system level. This may then necessitate further analysis at the system level. It is therefore clearly advantageous to ensure software is considered as part of the system level safety activities.

RELATED PATTERNS

This pattern is supported by the Software contribution safety argument pattern.

3.2. Software Contribution Safety Argument Pattern

Software Contribution Safety Argument Pattern			
Author	Richard Hawkins		
Created	09/12/08	**Last modified**	08/06/09

INTENT

This pattern provides the structure for arguments that the contributions made by software to system hazards are acceptably managed.

MOTIVATION

It is necessary to consider all of the ways in which errors may be introduced into the software which could lead to the software contribution. The software development process used will vary between different projects, however in all cases the software development is undertaken through varying levels of design. At each level the design must satisfy requirements of the higher level. These requirements may be explicitly captured as part of a requirements specification, or identified implicitly from the design itself. In [5] Jaffe et al propose an extensible model of development which captures this relationship between components at different tiers. Figure 1 illustrates the multi-tiered relationship between successively more detailed requirements and design information. Figure 2 illustrates in more detail the relationship among a tier n component's requirements, its design representation, and the tier n+1 requirements of the tier n+1 (sub) components identified in the design representation.

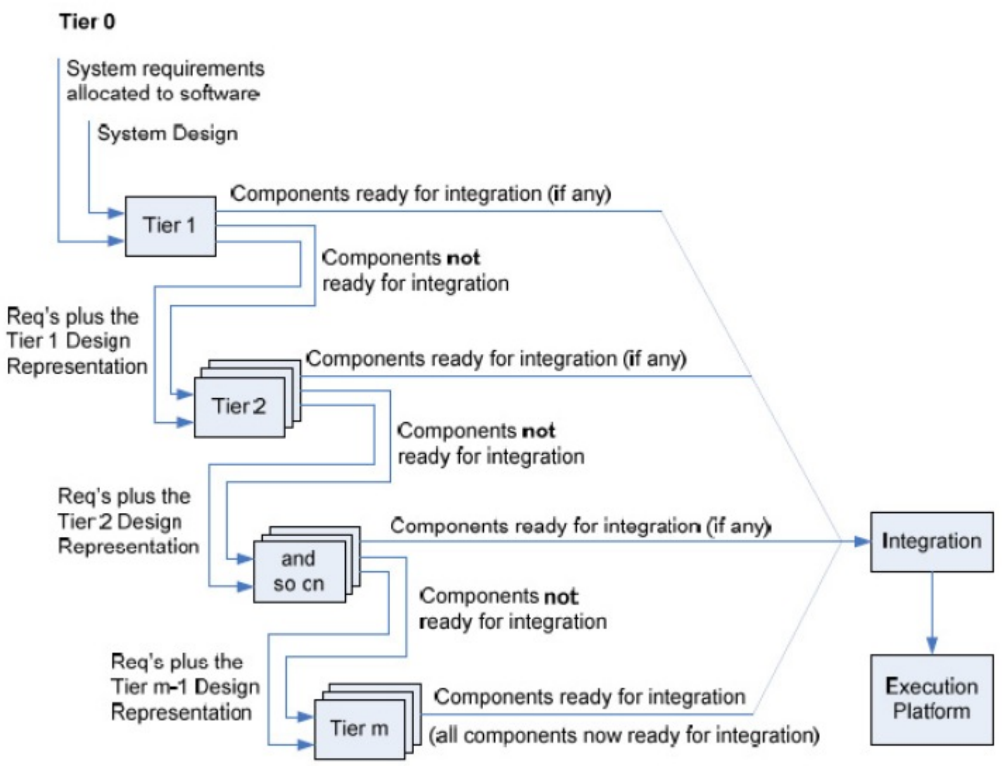

Tier 0
System requirements allocated to software
System Design
Tier 1
Components ready for integration (if any)
Components **not** ready for integration
Req's plus the Tier 1 Design Representation
Tier 2
Components ready for integration (if any)
Components **not** ready for integration
Req's plus the Tier 2 Design Representation
and so on
Components ready for integration (if any)
Components **not** ready for integration
Req's plus the Tier m-1 Design Representation
Tier m
Components ready for integration
(all components now ready for integration)
Integration
Execution Platform

Figure 2 - Illustration of a multi-tiered relationship

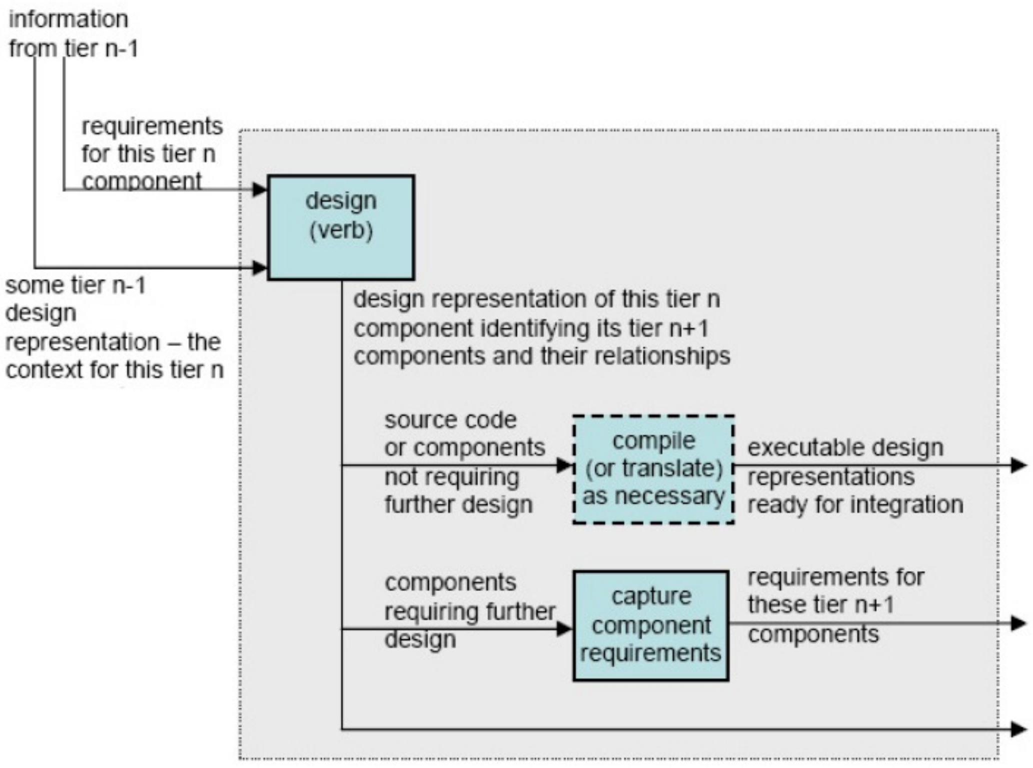

information from tier n-1
requirements for this tier n component
some tier n-1 design representation – the context for this tier n
design (verb)
design representation of this tier n component identifying its tier n+1 components and their relationships
source code or components not requiring further design
compile (or translate) as necessary
executable design representations ready for integration
components requiring further design
capture component requirements
requirements for these tier n+1 components

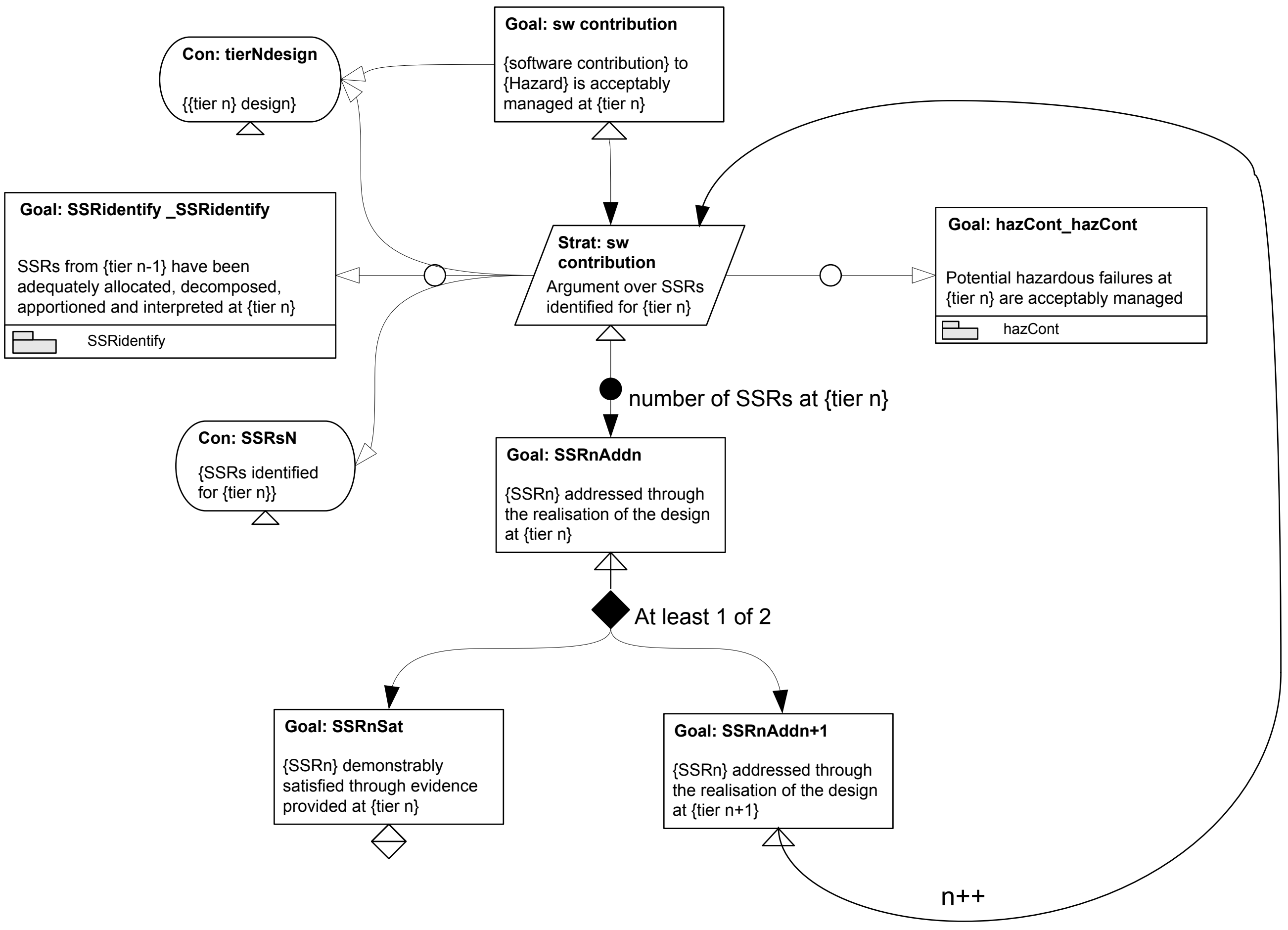
Figure 3 - More detailed illustration for a tier n component

From a safety perspective, it is necessary to ensure that at each tier, the software safety requirements derived at the previous tier are adequately addressed. This involves making design decisions which mitigate potential failures and adequately allocating and decomposing the software safety requirements (SSRs) through consideration of the design at that tier. At each tier it is also possible to introduce errors into the software which could manifest themselves as hazardous failures. It is therefore important in the software safety argument to also consider additional hazardous contributions that may be introduced at each tier.

This pattern therefore reflects the tier model discussed above in order to make it generally applicable to a broad range of development processes.

STRUCTURE

The structure of this argument pattern is shown in Figure 4 below.



Goal: sw contribution
{software contribution} to {Hazard} is acceptably managed at {tier n}
Con: tierNdesign
{{tier n} design}
Goal: SSRidentify _SSRidentify
SSRs from {tier n-1} have been adequately allocated, decomposed, apportioned and interpreted at {tier n}
SSRidentify
Strat: sw contribution
Argument over SSRs identified for {tier n}
Goal: hazCont_hazCont
Potential hazardous failures at {tier n} are acceptably managed
hazCont
number of SSRs at {tier n}
Con: SSRsN
{SSRs identified for {tier n}}
Goal: SSRnAddn
{SSRn} addressed through the realisation of the design at {tier n}
At least 1 of 2
Goal: SSRnSat
{SSRn} demonstrably satisfied through evidence provided at {tier n}
Goal: SSRnAddn+1
{SSRn} addressed through the realisation of the design at {tier n+1}
n++

Figure 4 – Software Contribution Safety Argument Pattern Structure

PARTICIPANTS

Goal: sw contribution

An instance of this goal is required for each of the identified software contributions to each of the system hazards. For this top claim in the pattern, {tier n} will in this case refer to the highest tier in the development process. This highest tier is generally referred to as (high-level) software requirements.

Strat: sw contribution

The strategy adopted is to argue over all the SSRs which are identified at this tier. These SSRs are either derived from the DSSRs of the previous tier, or through consideration of additional hazardous contributions that may occur at this tier.

Goal: SSRidentify

The SSRs from the previous tier must be allocated to the tier n design appropriately, having been suitably decomposed where necessary, and correctly apportioned across the design as part of that decomposition. The SSRs may also require interpretation to reflect the tier n design. As part of supporting this goal it is necessary to consider the design decisions that are taken in order to mitigate failures, including mechanisms for failure detection and response. At the highest tier, there are no SSRs from the previous tier, instead, the software contribution itself must be considered. This goal is crucial to the assurance of the argument as it provides the warrant for the adopted strategy of arguing over SSRS identified for tier n. This goal must be supported by an argument contained in a separate module (SSRidentify). The SSR identification software safety argument pattern may be used to generate an argument to support this goal. This goal is optional, since it may not necessarily be required to provide direct traceability at every tier. The decision as to whether this is required at a particular tier must be based on a consideration of assurance. It may be necessary to justify such a decision by providing an argument. The Argument justification software safety argument pattern may be used to provide such an argument. It would be possible, instead of supporting this goal, to simply provide an assumption node stating that SSRs from the previous tier have been adequately allocated, decomposed, apportioned and interpreted. This would however significantly reduce the assurance achieved, so the impact of such a decision must be considered.

Goal: hazCont

At any tier in the development there is the possibility of introducing additional contributions to hazards due to errors made at that tier. This goal claims that such potential hazardous contributions are addressed through the specification of additional SSRs. Supporting this goal requires that the potential hazardous contributions at tier n are adequately identified, and that SSRs sufficient to address those hazardous contributions are specified. This goal is crucial to the assurance of the argument as it provides the warrant for the adopted strategy of arguing over SSRs identified for tier n. This goal must be supported by an argument contained in a separate module (hazCont). The Hazardous contribution software safety argument pattern may be used to generate an argument to support this goal. This goal is optional, since it may not necessarily be required to identify hazardous contributions at every tier. The decision as to whether this is required at a particular tier must be based on a consideration of the impact on assurance. It may be necessary to justify such a decision by providing an argument. The Argument justification software safety argument pattern may be used to provide such an argument. It would be possible, instead of supporting this goal, to simply provide an assumption node stating that DSSRs have been correctly identified at tier n to address the identified potential additional hazardous contribution. This would however significantly reduce the assurance achieved, so the impact of such a decision must be considered.

Goal: SSRnAddn

An instance of this goal is created for each SSR identified at tier n (represented as SSRn). There is an option for how this goal is supported. It can be supported by either, or both of goals 'SSRnSat' and 'SSRnAddn+1'. It may be necessary to justify such a decision by providing an argument. The Argument justification software safety argument pattern may be used to provide such an argument.

Goal: SSRnAddn+1

It is possible to demonstrate that the SSRs at tier n are addressed by showing traceability down to the subsequent tier of development. The argument then continues through a further instantiation of 'Strat: sw contribution'. {tier n+1} then becomes {tier n}.

Goal: SSRnSat

It is possible at any tier to provide verification evidence of the satisfaction of the SSRs for that tier. This may be, for example, testing or analysis performed at that tier. Not all software is subject to the same number of tiers of development. Also, not all aspects of any particular software are necessarily developed over the same number of tiers. It is therefore also possible for implementation to occur at any tier. At the tier of implementation it is possible to provide argument and evidence to demonstrate that the SSR is satisfied by the implementation (such as different types of testing or analysis).

APPLICABILITY

This pattern should be applied as part of any hazard-directed software safety argument.

CONSEQUENCES

Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. 'Goal: SSRIdentify' must be supported. The DSSR identification software safety argument pattern presented in this catalogue can be used to support this goal. 'Goal: hazCont' must be supported. The Hazardous contribution software safety argument pattern presented in this catalogue can be used to support this goal. Finally 'Goal: SSRnSat' must be supported. As discussed, detailed guidance on the development of the argument to support this goal will be the subject of future work.

IMPLEMENTATION

This pattern should be instantiated as part of a software safety argument. An instantiation of 'Goal: SSRIdentify' must be created for each identified software contribution to each system hazard. {tier n}, and {tier n+1} must be instantiated with the names of the relevant tier. Note that as the argument is developed over multiple tiers, {tier n} will refer to different tiers. {SSRn} is used to refer to a SSR at tier n, and should be instantiated with the SSR itself or a unique identifier for the SSR. Note that in this pattern the looping link represents a repeating pattern of argument, and would not appear in such a manner in an instantiated argument.

POSSIBLE PITFALLS

Whilst acknowledging that in many cases not all the optional goals may be provided at each tier, it is also important to note the significance of this pattern on the achieved assurance. Assurance deficits introduced in instantiating this pattern can have a potentially large impact. In such cases the additional support may prove necessary. It is therefore important that the assurance impact of decisions taken at each tier of development are fully considered, to avoid additional work at a later date.

RELATED PATTERNS

Consideration should be given to the application of the Argument justification software safety argument pattern wherever significant decisions about how to instantiate the optional aspects of this pattern are made. The Argument justification software safety argument pattern should be instantiated in context to this pattern to justify the acceptability of any residual assurance deficits as a result of the instantiation decisions. This pattern supports the High-level software safety argument pattern. Support for ‘Goal: SSRIdentify’ and ‘Goal: hazCont’ can be provided using the SSR identification software safety argument pattern and the Hazardous contribution software safety argument pattern respectively.

3.3. SSR Identification Software Safety Argument Pattern

SSR Identification Software Safety Argument Pattern			
Author	Richard Hawkins		
Created	09/12/08	**Last modified**	08/06/09

INTENT

This pattern provides the structure for arguments that software safety requirements (SSRs) from a previous tier of development have been adequately captured at the next tier of development through the allocation, decomposition, apportionment or interpretation of the SSRs from the previous tier. This is achieved either through making design decisions which mitigate the SSR, or through the definition of additional SSRs.

STRUCTURE

The structure of this argument pattern is shown in Figure 5 below.

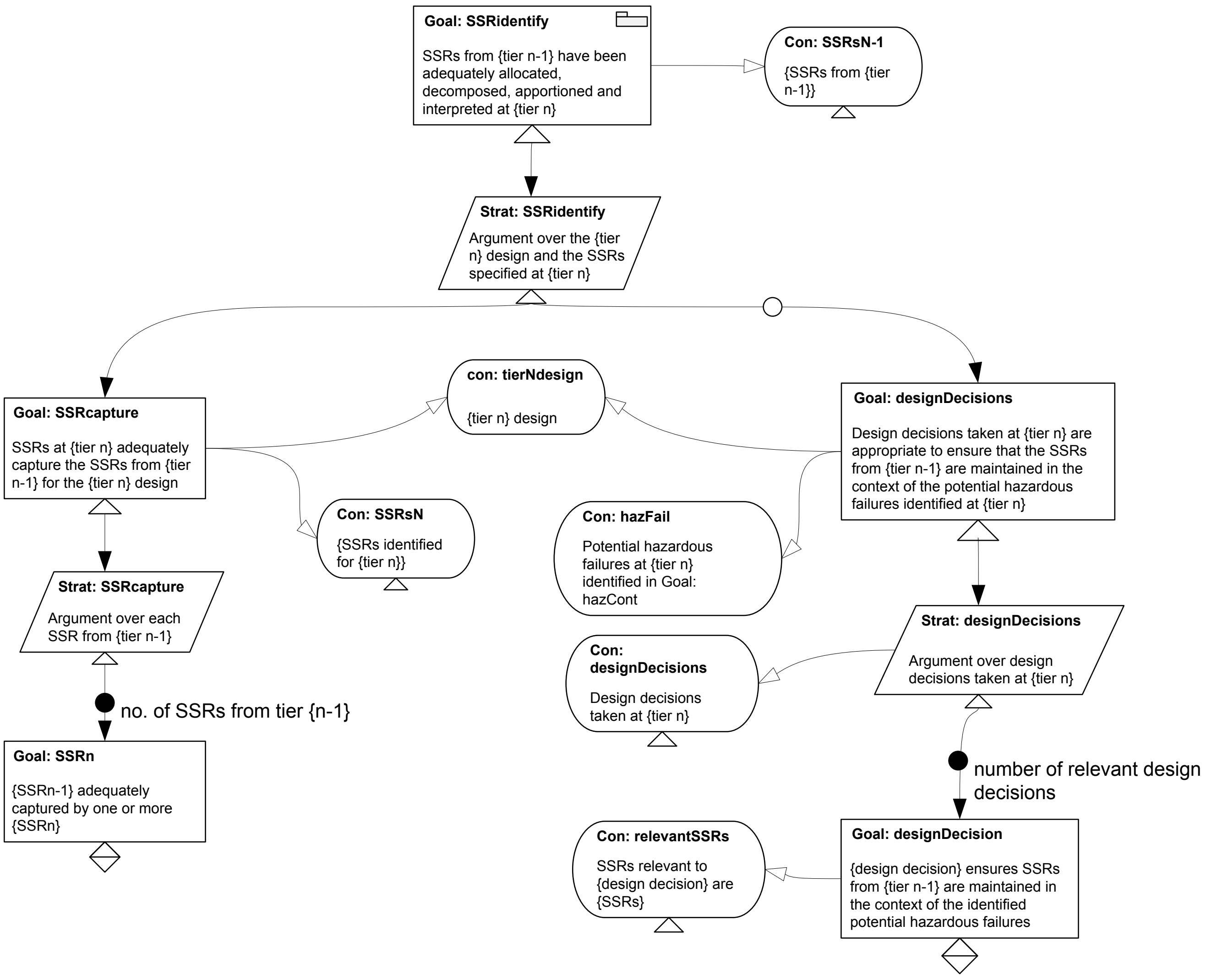

Goal: SSRidentify
SSRs from {tier n-1} have been adequately allocated, decomposed, apportioned and interpreted at {tier n}
Con: SSRsN-1
{SSRs from {tier n-1}}
Strat: SSRidentify
Argument over the {tier n} design and the SSRs specified at {tier n}
con: tierNdesign
{tier n} design
Goal: SSRcapture
SSRs at {tier n} adequately capture the SSRs from {tier n-1} for the {tier n} design
Con: SSRsN
{SSRs identified for {tier n}}
Goal: designDecisions
Design decisions taken at {tier n} are appropriate to ensure that the SSRs from {tier n-1} are maintained in the context of the potential hazardous failures identified at {tier n}
Con: hazFail
Potential hazardous failures at {tier n} identified in Goal: hazCont
Strat: SSRcapture
Argument over each SSR from {tier n-1}
no. of SSRs from tier {n-1}
Goal: SSRn
{SSRn-1} adequately captured by one or more {SSRn}
Con: designDecisions
Design decisions taken at {tier n}
Strat: designDecisions
Argument over design decisions taken at {tier n}
number of relevant design decisions
Con: relevantSSRs
SSRs relevant to {design decision} are {SSRs}
Goal: designDecision
{design decision} ensures SSRs from {tier n-1} are maintained in the context of the identified potential hazardous failures

Figure 5 – SSR Identification Software Safety Argument Pattern Structure

PARTICIPANTS

Goal: SSRidentify

This is a public goal in a separate argument module which can be referenced from other software safety argument modules using an away goal reference. This claim is applicable wherever an argument is being presented over the tiers of the software development lifecycle. {tier n} refers to the current tier being considered in the argument. {tier n-1} refers to the previous tier of development. At each tier it is necessary to demonstrate that the SSRs from {tier n-1} are adequately captured in the design of {tier n}.

Strat: SSRidentify

This is achieved either through making design decisions at {tier n} which facilitate the satisfaction of the {tier n-1} SSR, or through the definition of SSRs for {tier n} which consider the {tier n} design. In some cases a mixture of appropriate design decision and SSR definition might be required to capture all of the {tier n} SSRs. In other cases just one approach may be sufficient, this will depend on a number of factors including the nature of the SSRs, which tier is being considered and the nature of the design of {tier n}. The Argument justification software safety argument pattern may be used to justify the adopted strategy.

Goal: SSRcapture

This goal claims that the design of {tier n} has been considered in order to define SSRs for {tier n} which adequately capture the SSRs from {tier n-1}.

Con: tierNdesign

The design of {tier n} will be determined by the design decisions made, some of which may have been influenced by {tier n-1} SSRs. The {tier n} design will also determine the nature of the SSRs defined at {tier n}. This context is therefore common to both 'Goal: SSRcapture' and 'Goal: designDecisions'.

Goal: SSRn

An instance of this goal is created for each SSR from {tier n-1}. To adequately reflect each {tier n-1} SSR, one or more SSRs may be required at {tier n}.

Goal: designDecisions

It may be possible to facilitate the satisfaction of some of the {tier n-1} SSRs through decisions taken in the design of {tier n}. For example, a decision to have redundant components may be taken in order to help satisfy a SSR relating to the availability of an item of data. Alternatively a decision may be taken to introduce into the design a mechanism for detecting and handling failures which may lead to the breach of an SSR. It may also be possible, for example, to prevent interference between components through ensuring physical or logical partitioning in the design. This goal allows claims to be made that such decisions reflect the SSRs from {tier n-1}. The appropriate of the design decisions will depend upon the nature of the SSRs.

Goal: designDecision

An instance of this goal is created for each design decision taken which is relevant to the satisfaction of a SSR from {tier n-1}. Each instance of this goal requires a supporting argument which demonstrates how the design feature supports the SSR satisfaction.

Con: relevantSSRs

This context specifies the SSRs which this design decision helps to satisfy.

APPLICABILITY

This pattern should be applied as part of any hazard-directed software safety argument to provide a warrant for an argument that SSRs from one development tier are adequately addressed at the next tier.

CONSEQUENCES

Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. An instance of 'Goal: SSRn' must be supported for each SSR from {tier n-1}. An argument should be provided which demonstrates that one or more SSRs specified at {tier n} adequately capture the {tier n-1} SSR for the design at {tier n}. An instance of 'Goal: designDecision' must be supported for each design decision which was made to facilitate the satisfaction of SSRs at {tier n}. 'Goal: HSFMdetect' and 'Goal: SSRprevent', if created, must also be supported.

IMPLEMENTATION

{tier n}, and {tier n-1} must be instantiated with the names of the relevant tier. This could for example be class design and high-level software design respectively.

POSSIBLE PITFALLS

The SSRs defined at {tier n} must adequately reflect the {tier n} design. If that design changes, it is necessary to check that the SSRs defined at {tier n} are still valid, and if necessary update the SSRs to reflect the design changes. For this reason it would be advantageous to have a reasonably stable design for {tier n} before defining SSRs for that tier. Since the SSRs from {tier n-1} may influence the design, it is important that this is considered early in the design of {tier n}, such that any resulting design changes are not required late in the development.

RELATED PATTERNS

This pattern is used to provide context to the Software contribution safety argument pattern. Consideration should be given to the application of the Argument justification software safety argument pattern wherever significant decisions about how to instantiate the optional aspects of this pattern are made.

3.4. Hazardous Contribution Software Safety Argument Pattern

Hazardous Contribution Software Safety Argument Pattern			
Author	Richard Hawkins		
Created	09/12/08	**Last modified**	08/06/09

INTENT

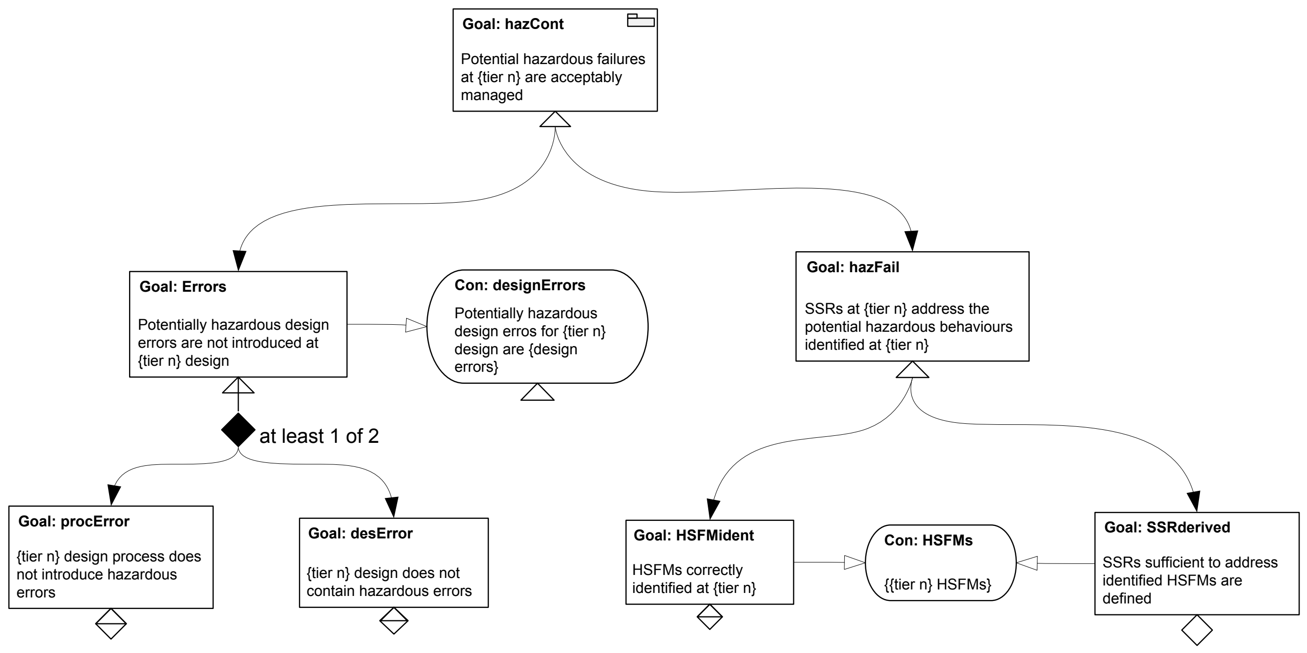
This pattern provides the structure for arguments that potential hazardous failures that may arise at {tier n} are acceptably managed.

MOTIVATION

At each tier of software development it is possible that hazardous failures may manifest themselves. This argument demonstrates how the hazardous failures are prevented. This is achieved in two ways. Firstly potential hazardous failure modes are identified, and appropriate SSRs defined in response. Secondly, the absence of design errors which could cause hazardous failures must also be demonstrated. It should be noted that this aspect of the argument will often consider more generally how errors are removed from the design.

STRUCTURE

The structure of this argument pattern is shown in Figure 6 below.



Goal: hazCont
Potential hazardous failures at {tier n} are acceptably managed
Goal: Errors
Potentially hazardous design errors are not introduced at {tier n} design
Con: designErrors
Potentially hazardous design erros for {tier n} design are {design errors}
Goal: hazFail
SSRs at {tier n} address the potential hazardous behaviours identified at {tier n}
at least 1 of 2
Goal: procError
{tier n} design process does not introduce hazardous errors
Goal: desError
{tier n} design does not contain hazardous errors
Goal: HSFMident
HSFMs correctly identified at {tier n}
Con: HSFMs
{{tier n} HSFMs}
Goal: SSRderived
SSRs sufficient to address identified HSFMs are defined

Figure 6 - Hazardous Contribution Software Safety Argument Pattern Structure

PARTICIPANTS

Goal: hazCont

This is a public goal in a separate argument module which can be referenced from other software safety argument modules using an away goal reference. This claim is applicable wherever an argument is being presented over the tiers of the software development lifecycle. {tier n} refers to the current tier being considered in the argument. This goal claims that the potential hazardous failures at the current tier are acceptably managed.

Goal: Errors

The design process at any tier may be flawed. This goal claims that potentially hazardous design (or code) errors have not been introduced at the current tier. This supported by arguing about the design process adopted at the current tier, and about the design artefact itself.

Goal: desError

This goal claims that the design (or code) produced at the current tier does not contain potentially hazardous errors.

Goal: procError

This goal must be supported by argument and evidence about the integrity of the design process that is used at the current tier. Note that at the lowest level tiers this may include the coding process.

Goal: hazFail

This goal claims that SSRs are identified, sufficient to address the potential hazardous behaviours identified at {tier n}. The goal is supported by demonstrating that hazardous software failure modes (HSFMs) (that is failure of the software which could contribute to a hazard at the system level) at {tier n} are sufficiently identified, and that each of these HSFMs is addressed through the definition of one or more SSRs.

APPLICABILITY

This pattern should be applied as part of any hazard-directed software safety argument to provide a warrant for an argument that SSRs from one development tier are adequately addressed at the next tier.

CONSEQUENCES

Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. 'Goal: deviations' must be supported by an argument provided in a 'deviations' safety argument module. An instance of 'Goal: HSFMaddress' must be supported for each HSFM identified at {tier n}. 'Goal: HSFMs' must also be supported.

IMPLEMENTATION

The techniques most appropriate to use to identify potential deviations from intended behaviour at each tier will vary. Appendix B provides some examples of the types of hazard and failure analysis techniques that may be used at some of the possible tiers.

RELATED PATTERNS

This pattern is used to provide context to the Software contribution safety argument pattern.

3.5. Software Contribution Safety Argument Pattern with Grouping

Software Contribution Safety Argument Pattern with Grouping			
Author	Richard Hawkins		
Created	07/12/10	**Last modified**	07/12/10

INTENT

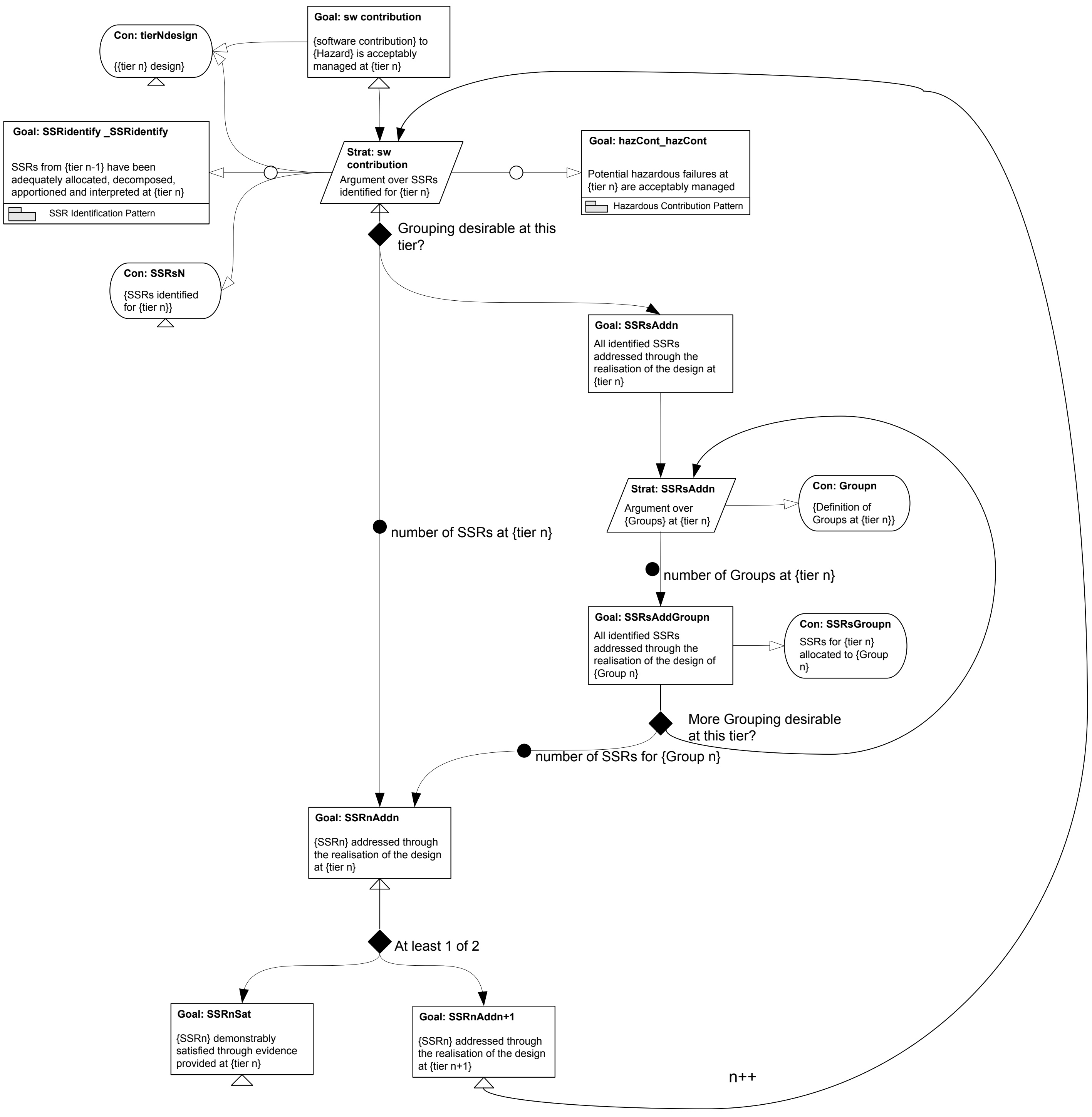
This pattern is an extension of the Software Contribution Safety Argument Pattern. It provides the option of grouping the argument to reflect natural requirements groupings in the software design. For example, for an instantiation of the Software Contribution Safety Argument Pattern at the software architecture level, it may be desirable to create groupings in the argument which reflect each of the individual architectural design elements.

MOTIVATION

Grouping aspects of the Software Contribution Safety Argument Pattern can help to manage the safety argument where there exist a large number of claims at a particular tier of decomposition by splitting the argument into manageable chunks.

STRUCTURE

The structure of this argument pattern is shown in Figure 7 below.



Goal: sw contribution
{software contribution} to {Hazard} is acceptably managed at {tier n}
Con: tierNdesign
{{tier n} design}
Goal: SSRidentify _SSRidentify
SSRs from {tier n-1} have been adequately allocated, decomposed, apportioned and interpreted at {tier n}
SSR Identification Pattern
Strat: sw contribution
Argument over SSRs identified for {tier n}
Goal: hazCont_hazCont
Potential hazardous failures at {tier n} are acceptably managed
Hazardous Contribution Pattern
Grouping desirable at this tier?
Con: SSRsN
{SSRs identified for {tier n}}
Goal: SSRsAddn
All identified SSRs addressed through the realisation of the design at {tier n}
Strat: SSRsAddn
Argument over {Groups} at {tier n}
Con: Groupn
{Definition of Groups at {tier n}}
number of SSRs at {tier n}
number of Groups at {tier n}
Goal: SSRsAddGroupn
All identified SSRs addressed through the realisation of the design of {Group n}
Con: SSRsGroupn
SSRs for {tier n} allocated to {Group n}
More Grouping desirable at this tier?
number of SSRs for {Group n}
Goal: SSRnAddn
{SSRn} addressed through the realisation of the design at {tier n}
At least 1 of 2
Goal: SSRnSat
{SSRn} demonstrably satisfied through evidence provided at {tier n}
Goal: SSRnAddn+1
{SSRn} addressed through the realisation of the design at {tier n+1}
n++

Figure 7 – Software Contribution Safety Argument Pattern with Grouping Structure

PARTICIPANTS

In this section the participants of the Software Contribution Safety Argument Pattern are not restated. These participants are documented fully in the Software Contribution Safety Argument Pattern. Just the participants of the grouping addition are described here.

Goal: SSRsAddn

An instance of this goal is created if grouping is desirable at the tier of instantiation. This goal claims that all the SSRs that have been identified for the current tier of design have been realised through the design.

Strat: SSRsAddn

The strategy adopted is to provide an argument over a number of groups of design elements at {tier n}

Con: Groupn

This context defines what the groups are over which the argument will be structured.

Goal: SSRsAddGroupn

An instance of this goal is created for each group over which the argument will be made at {tier n}. {Group n} should be instantiated with the name of the design grouping which is being considered.

Con: SSRsGroupn

This context defines which of the SSRs that were identified for {tier n} have been allocated to {Group n}. All SSRs must be allocated to a group.

Goal: SSRnAddn

An instance of this goal is created for each SSR allocated to {Group n} (represented as SSRn). There is an option for how this goal is supported. It can be supported by either, or both of goals 'SSRnSat' and 'SSRnAddn+1'. It may be necessary to justify such a decision by providing an argument. The Argument justification software safety argument pattern may be used to provide such an argument.

APPLICABILITY

This pattern should be applied whenever it is desirable to group the structure of the argument at a particular tier to reflect natural requirements groupings in the software design.

IMPLEMENTATION

The key implementation decision is when to create groupings in the argument. The option to argue over a group of SSRs could be implemented for any large components in the software design (e.g logical partitions) in order to split the argument into manageable chunks. There is however no obligation to create any groupings. It should be noted that arguing over a group of SSRs is simply an organising principle to make the argument more easily managed and the decision to group (or not) will *not* affect the argument that is ultimately provided as to how those SSRs have been satisfied.

POSSIBLE PITFALLS

The option of grouping the argument as defined in this pattern should only be used to group together SSRs at an existing level of decomposition. It should not be used when decomposing a design, or deriving additional SSRs. In this case the ‘normal’ decomposition option in the pattern (as defined in the Software contribution safety argument pattern) should always be used.

RELATED PATTERNS

This pattern extends the Software contribution safety argument pattern.

4. Conclusions

This document has presented a catalogue of software safety argument patterns. The catalogue contains a number of patterns which may be used together in order to construct a compelling software safety argument for the system under consideration.

The software safety argument patterns describe the nature of the argument and safety claims that would be expected for *any* software safety case. The way the argument is supported may be different for each system but the ‘core elements’ of the argument (as defined by the patterns) remain.

The effectiveness of the software safety argument patterns has been demonstrated through application to a number of case studies. These case studies (see [4]) have highlighted the benefits of utilising the patterns when developing software safety cases.

The authors actively seek feedback from users of the patterns presented in this document, and will update the contents of the catalogue, where required, based on user experiences. If you have comments, please contact the authors directly via email.

5. References

[1] T. Kelly, *Arguing Safety – A Systematic Approach to Managing Safety Cases,* PhD Thesis, Department of Computer Science, The University of York, 1998.

[2] R. Weaver, *The Safety of Software – Constructing and Assuring Arguments,* PhD Thesis, Department of Computer Science, The University of York, 2003.

[3] F. Ye, *Justifying the Use of COTS Components within Safety Critical Aplications,* PhD Thesis, Department of Computer Science, The University of York, 2005.

[4] R. Hawkins, K. Clegg, R. Alexander, T. Kelly, *Using a Software Safety Argument Pattern Catalogue: Two Case Studies,* in F. Flammini, S. Bologna, and V. Vittorini (Eds.): SAFECOMP 2011, LNCS 6894, pp. 185 - 198. Springer, Heidelberg, 2011.

[5] M. Jaffe, R. Busser, D. Daniels, H. Delseny, G. Romanski, *Progress Report on Some Proposed Upgrades to the Conceptual Underpinnings of DO178B/ED-12B*, In Proceedings of the 3rd IET International Conference on System Safety, 2008.

[6] T. Kelly, *Concepts and principles of compositional safety case construction,* Technical Report COMSA/2001/1/1, The University of York, 2001.

6. Appendix A

A.1 Goal Structuring Notation (GSN)

Goal Structuring Notation (GSN) is a structured graphical argument notation that is widely used to clearly represent safety arguments. The basic GSN symbols are shown in Figure 8.

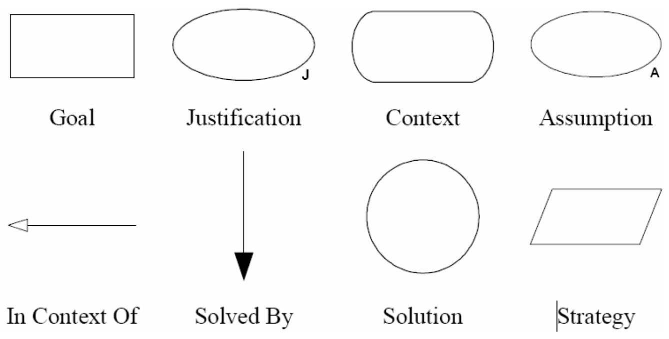


Figure 8 - Core GSN Elements

These symbols can be used to construct an argument by showing how safety claims (goals) are broken down into sub-claims, until eventually they can be supported by evidence (solutions). The strategies adopted, and the rationale (assumptions and justifications) can be captured, along with the context in which the goals are stated. More details on the use of GSN can be found in [1].

A.2 Modular GSN

Modular safety cases provide a means of organising large or complex safety cases into separate but interrelated component modules of argument and evidence. When splitting an argument into modules it becomes necessary to be able to refer to goals that exist within other modules. To refer to goals in other modules, the GSN element "Away Goal" is used. As seen in Figure 9. Each away goal contains a module identifier, which is a reference to the module where the goal can be found. Away goals can only be used to reference goals that have explicitly been declared as public in another module. Away goals can be used as a way of providing support for a goal in one module, with a goal in another module. Away goals can also be used to provide contextual backing for goals, strategies and solutions. More details on the use of modular GSN can be found in [6].

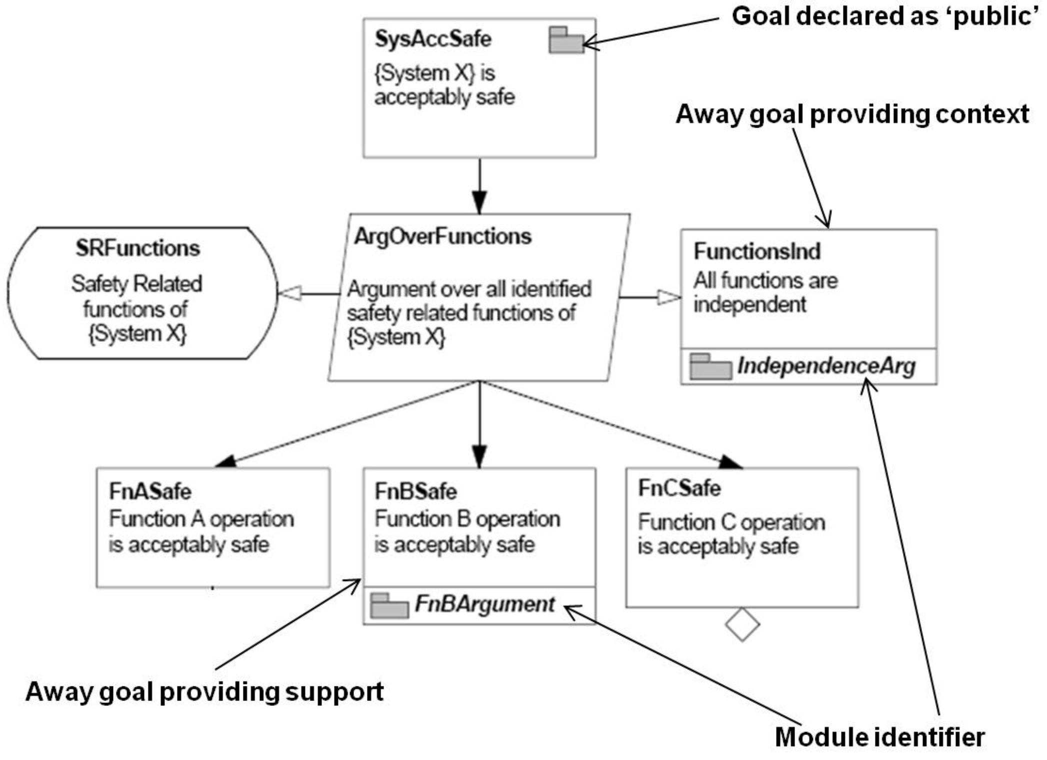

Goal declared as 'public'
SysAccSafe
{System X} is acceptably safe
Away goal providing context
SRFunctions
Safety Related functions of {System X}
ArgOverFunctions
Argument over all identified safety related functions of {System X}
FunctionsInd
All functions are independent
IndependenceArg
FnASafe
Function A operation is acceptably safe
FnBSafe
Function B operation is acceptably safe
FnBArgument
FnCSafe
Function C operation is acceptably safe
Away goal providing support
Module identifier

Figure 9 - Modular Extensions to GSN

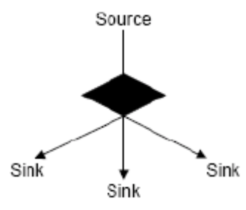
A.3 GSN Pattern Notation

To create safety argument patterns, GSN is extended to support multiplicity, optionality and abstraction. The multiplicity extensions shown in Figure 10 are used to describe how many instances of one entity relate to another entity. They are annotations on existing GSN relational arrows.

——●→ n	A solid ball is the symbol for many (meaning zero or more). The label next to the ball indicates the cardinality of the relationship.
——○→	A hollow ball indicates "optional" (meaning zero or one).
——→	A line without multiplicity symbols indicates a one to one relationship (as in conventional GSN).

Figure 10 - GSN Multiplicity Extensions

The optionality extension shown in Figure 11 is used to denote possible alternative support. It can represent a 1-of-n or an m-of-n choice.



1 source has three possible sinks

Multiplicity relations can be combined with optionality relations. Placing multiplicity symbols prior to the 'choice' vertex (squashed diamond) describes a multiplicity over all the optional relations. Placing a multiplicity symbol on individual optional relations (i.e. just prior to the sink) describes a multiplicity over that relation only.

Figure 11 - GSN Optionality Extensions

The abstraction extensions shown in Figure 12 allow GSN elements to be generalised for future instantiation.

△ **Uninstantiated Entity**	This placeholder denotes that the attached entity remains to be instantiated, i.e. at some later stage the 'abstract' entity needs to be **replaced** (instantiated) with a more concrete instance. Instantiation may be provided either through offering a concrete instance or subtypes denoted by the Is_A relation.
◇ **Undeveloped Entity**	This placeholder denotes that the attached entity requires further development, i.e. at some later stage the entity needs to be (hierarchically) decomposed and further supported by sub-entities. Unlike uninstantiated elements, undeveloped elements are **not** replaced, they are further elaborated **in** the goal structure, i.e. as with undeveloped events in conventional Fault Tree Notation.

Figure 12 - GSN Abstraction Extensions

Kelly [1] has suggested a method of documenting patterns, such that the information necessary for their successful instantiation is captured. He suggests that for each pattern, information is documented under the following headings:

Pattern name The name of the pattern should communicate the central argument being presented in the pattern.

Intent Should state what the pattern is trying to achieve.

Motivation Communicates why the pattern was constructed.

Structure Here GSN is used to present the structure of the argument pattern.

Participants Provides additional information on each of the elements of the GSN argument.

Applicability Records under what circumstances the pattern can and should be applied.

Consequences What remains to be done after having applied the argument pattern.

Implementation This should describe how to implement the pattern, in particular providing hints and techniques for the successful application of the pattern, describing ways in which it is possible to get it wrong, and recording any common misinterpretations of the terms or concepts used.

Related patterns Identify any related safety argument patterns.