Deriving Safety Contracts to Support Architecture Design of Safety Critical Systems

Irfan Sljivo*, Omar Jaradat*, Iain Bate*†, Patrick Graydon*
* Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
†Department of Computer Science, University of York, York, UK
{irfan.sljivo, omar.jaradat, patrick.graydon}@mdh.se, iain.bate@cs.york.ac.uk

Abstract—**The use of contracts to enhance the maintainability of safety-critical systems has received a significant amount of research effort in recent years. However some key issues have been identified: the difficulty in dealing with the wide range of properties of systems and deriving contracts to capture those properties; and the challenge of dealing with the inevitable incompleteness of the contracts. In this paper, we explore how the derivation of contracts can be performed based on the results of failure analysis. We use the concept of safety kernels to alleviate the issues. Firstly the safety kernel means that the properties of the system that we may wish to manage can be dealt with at a more abstract level, reducing the challenges of representation and completeness of the "safety" contracts. Secondly the set of safety contracts is reduced so it is possible to reason about their satisfaction in a more rigorous manner.**

Keywords—Safety Contracts, Safety Kernels, Incremental Certification, Modular Safety Case, Safety Argument.

I. Introduction

Contract-based approaches aimed at decreasing certification costs and increasing maintainability of safety-critical systems have been the topic of much research recently. Many works focus on the underlying contract theory [4]–[6], while not that many focus on the difficulty of specifying contracts and the problem of their (in)completeness [7], [8]. A component contract is usually defined as a pair of assumption/guarantee assertions such that the component offers the guarantee if an environment in which the component is used satisfies the assumptions. The contracts can be characterised as either strong or weak [18]. The strong contracts capture behaviours that should hold in all environments/contexts in which the component can be used, while the weak contracts capture context specific behaviours. A "safety contract" is a contract that specifically deals with behaviours of the system linked to hazard mitigation.

Developers of safety-critical systems are sometimes required to construct a safety case to show that the system is acceptably safe to operate in a given context, i.e., that the risks of hazards occurring are reduced to acceptable levels. As a way of documenting the safety case, a safety argument is often used to show how safety claims about the system are connected and supported by evidence. While the argument presents the safety-relevant information about the system in a comprehensible way, safety contracts capture the safety-relevant information in a more rigorous manner. The fact that both, the safety argument and safety contracts, deal with the same information makes the contracts an important aid in safety case maintenance [15].

As safety-critical systems are characterised by a wide-range of properties that influence safety-relevant behaviour of components, it is challenging to derive contracts with a complete set of relevant assumptions on the environment that imply the guaranteed component behaviour. When dealing with completeness of contracts without a reference point against which we can check if the contracts are complete, then the contracts are inevitably incomplete, since we cannot capture all assumptions. To talk about contract completeness we need to identify the reference point against which we can check the contracts and that we can use to derive the contracts as well. For example, safety contracts describing failure behaviours of a component can be derived from a failure analysis such as Fault Tree Analysis (FTA). Not all failure behaviours obtained by failure analysis are relevant from the perspective of hazard analysis results. Regardless of that, we still categorise contracts capturing such behaviours as *safety* contracts, since the captured behaviours can be safety-relevant in case of change to the system or for other systems in which the component can be used.

An approach to developing systems based on a "safety kernel" was first proposed by Rushby [16] and used by Wika [19]. The basic principles of their work are that:

1) The safety kernel protects the system from key (higher criticality) hazardous events by checking that data flowing out of a module of the system would not violate Derived Safety Requirements (DSR) obtained via hazard analysis.
2) The safety kernel itself is much simpler than the rest of the system.

The simplicity means that the safety kernel can be developed to the requisite high integrity even if the rest of the system cannot be. Overall, the system is at least as safe as without a safety kernel but costs may be reduced. In this paper, we extend the original concept to include safety contracts being associated with the safety kernel to help facilitate incremental certification. The simplicity of the safety kernel also means the aforementioned problems of representing contracts and achieving completeness are eased.

Potential system changes during the system lifetime may impact some parts of the safety case. These affected parts necessitate updating the safety case with respect to those changes. We refer to the updating of the safety case after implementing a system change as *incremental certification*. The intention that change impact analysis can be performed by mainly assessing whether the contracts still hold is slightly unrealistic as there are significant issues with achieving complete contracts [8]. We deem that change impact analysis can be guided by accessing the satisfaction and completeness of contracts with respect to failure analyses.

In this paper, we focus on the safety contract derivation and the issue of their (in)completeness, as these two steps form the basis for establishing safety case maintenance techniques using the safety contracts. We judge that the contract completeness can be established only with respect to a clearly identified reference point such as failure analysis. Since failure analysis itself can be incomplete, the derived safety contracts are at least as complete as the analysis itself. Although contract completeness cannot be established in general, contracts can be used for guiding the designer to the key properties of the system as part of de-risking incremental certification and making it more efficient. This is supplemented by the designer being given scenario-specific guidance on how to deal with certain likely changes. In general for safety-critical systems there is often a clear development roadmap that makes this form of guidance practical. For instance it may be known that in N years time the developers will want to change the processors used due to obsolescence or remove a hydro-mechanical backup due to weight. Maintainers updating or upgrading a system might benefit from the original designers' insight on planned change scenarios [15].

The contribution and structure of the paper is as follows. In section II, we present the related work and an illustrative example used to demonstrate the approach. An architecture and supporting development process, in section III, that allows two types of contracts to be supported that should lead to a reduction in the initial certification costs as well as making the system easier to maintain. In section IV, we demonstrate an approach to deriving safety contracts from FTA and present how the derived contracts completeness check could be performed with respect to the fault trees. In section V, we present a safety argument based on the use of the safety kernel and contracts. Finally, we present summary and conclusions in section VI.

II. Background and Motivation

In this section we present the state of the art related to contracts and modular safety arguments. In the second part of the section we provide a brief description of the computer assisted braking system used to illustrate the approach.

A. Related Work

We group the related work into two areas: contract-based approaches for safety-critical systems and approaches related to safety arguments for incremental certification.

1) Use of Contracts in Safety-Critical Systems: An "informal" contract-based approach is proposed in [7]. The approach uses dependency-guarantee relationships to capture dependencies between modules. The captured dependencies are identified by considering predicted changes in the system in order to best contain their impact. A difficulty that arises is that usually not all dependencies can be captured if contracts are restricted to the relationship between just two modules as dependency chains can span across several modules. Furthermore, the issue of contract incompleteness is not fully addressed.

A more formal contract-based approach is shown in [20]. The work presents a language for describing assumption/guarantee contracts used to capture vertical dependencies between a software application and a hardware platform. While the approach provides a benefit of automatic generation of parts of arguments, it does not support capturing the broad range of assumptions needed for a guarantee to be still valid when a change in the environment occurs.

A range of formal contract-based approaches based on contract algebra can be found in [4]–[6]. The contract algebra includes definitions of contract refinement, composition, conjunction etc., making these approaches quite powerful when it comes to contract verification. The contract examples provided in [5], [6] do not focus on failure behaviour, but rather on behaviour when no failures occur. Moreover, the presented contracts on timing behaviour require additional assumptions if they are to be used in the process of incremental or modular certification [8]. In our work we propose that in addition to contracts describing expected behaviour in a specific context captured within weak contracts, we capture strong contracts describing how the faults in the system are handled by the safety kernel. Due to the properties of the safety kernel, such contracts are generally easier to satisfy due to fewer assumptions.

2) Safety Argumentation in Support of Incremental Certification: In safety critical systems, particularly those for which a safety case should be provided, change management is a painstaking process. That is because accommodating the changes in the system domain should be followed by updating the safety case (i.e., incremental certification) in a safe and efficient manner. A process is proposed in [15] to facilitate the incremental change and evolving system capability. One objective of the Modular Software Safety Case (MSSC) process is to minimise the impact on the safety case of changes which might be expected during the life of the system. Using the process may increase the system flexibility to accept changes.

The structure of the argument has a significant role in accommodating the changes. Well structured arguments clearly demonstrate the relationships between the argument claims and evidence, therefore it is easier to understand the impact of changes on them than poorly structured arguments. Moreover, well structured arguments can be exploited to prioritise the handling of change, identify the key areas of concern, and hence de-risk the change management process. An approach is proposed in [14] to show how the safety argument structure facilitates the systematic impact assessment of the safety case after applying changes. More specifically, the proposed approach shows how it is possible to use the recorded dependencies of the goal structure to follow through the impact of a change and recover from change.

Another approach is proposed in [10] to facilitating safety case change impact analysis. In this approach, automated analysis of information given as annotations to the safety argument highlights suspect safety evidence that may need updating following a change to the system being performed.

B. Overview of the Computer Assisted Braking System

In this section we will present the computer assisted braking system of an aircraft used in ARP4761 standard [17]

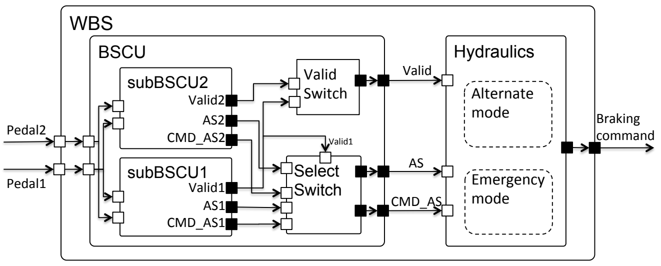

Fig. 1. Wheel Braking System - High Level View

to demonstrate the safety assessment process. The standard describes a Wheel Braking System (*WBS*) that takes two input brake pedal signals and outputs the braking command signal. The high level architecture is shown in Fig. 1. For the purposes of this paper we consider that all six components of the WBS shown in Fig. 1 are implemented in software.

The system is composed of two subsystems: Brake System Control Unit (*BSCU*) and *Hydraulics*. The brake pedal signals are forwarded to *BSCU*, which generates braking commands and sends the commands via direct link to *Hydraulics* subsystem that executes the braking commands. If the *BSCU*, which makes the normal operation mode, fails then *Hydraulics* uses an alternate mode to perform the braking. If both, normal and alternate mode fail, emergency brake is used.

In order to address the availability and integrity requirements, *BSCU* is designed with two redundant dual channel systems: *subBSCU1* and *subBSCU2*. Each of these subsystems consists of *Monitor* and *Command* components. *Monitor* and *Command* take the same pedal position inputs, and both calculate the command value. The two values are compared within the *Monitor* component and the result of the comparison is forwarded as true or false through *Valid* signal. The *SelectSwitch* component forwards the results from *subBSCU1* by default. If *subBSCU1* reports that fault occurred through *Valid* signal, then *SelectSwitch* component forwards the results from *subBSCU2* subsystem.

III. Overall Development Approach

In order to make the safety contracts more useful, i.e., applicable in more different contexts and less susceptible to changes, we use the concept of safety kernels in the development process. Safety kernels are generally simple and independent mechanisms which behaviour can be easily ensured. Due to their simplicity and high independence, safety kernel behaviours can be specified more abstractly, i.e., with fewer context-specific assumptions. A reduced number of required assumptions increases reusability of safety information captured by the contracts. This allows us to provide better support for incremental certification through reuse of evidence and safety reasoning related to contracts, and ease change management within safety arguments. Besides safety kernels, other types of failure mitigation and recovery techniques can be implemented and packaged together with components. We refer to such techniques as component wrappers.

We build our development approaches that utilise the notions of safety kernels and component wrappers on the well-established practices recommended by safety standards. The proposed development approaches can be summarised by the following steps:

1) Perform a hazard analysis as required by most standards.
2) Perform causal analysis (e.g., FTA) to understand how the hazards can occur.
3) Create strong contracts for the fault handling behaviours that are offered in all contexts. Such behaviours that are specified more abstractly can be achieved with the use of safety kernels.
4) Create weak contracts for the fault handling behaviours that are context specific. Such behaviours are usually achieved by failure mitigation and recovery techniques (e.g., component wrappers) that are not developed with high independence from the context.
5) Create an architecture which includes:
    a) Features to enforce the separation between the safety kernel and components. The safety kernel can only provide sufficient protection to allow it to provide fault tolerance if it can be argued that failures of the components do not interfere with its operation.
    b) A design for the safety kernel that provides fault tolerance, principally fault detection and recovery, with respect to the mitigation of the more critical hazards.
    c) A design for component wrappers that provides fault tolerance, principally fault detection and recovery, with respect to the mitigation of the less critical hazards. This largely deals with signal validation for data flowing in and out of the component. It is noted that some signals will be protected by both a wrapper and a safety kernel where used by multiple components.
6) Revise the fault tree to include the safety kernel and wrapper in the possible causes of hazards and judge whether the residual risks are acceptable. If the risks are not acceptable, judge whether more complex wrappers or more safety kernel functions would address the issues.

The development approach follows a typical set of stages except for the addition of contracts and the use of a safety kernel and wrappers. After deriving the safety contracts, the development approach continues to revise the contracts by checking if they are sufficiently complete and whether the described behaviours are sufficient to show that all identified hazards have been adequately addressed. Additional evidence backing the contracts is provided during the verification steps.

IV. Definition of Safety Contracts

In this section we present part of the FTA performed on WBS with (section IV-B) and without (section IV-A) safety kernels. Later in section IV-C, we show how the results of the analysis can be used to derive safety contracts capturing corresponding safety behaviour of components addressed within the fault trees. In the second part of section IV-C we discuss the problem of incompleteness of the safety contracts and propose how the contract completeness checking could be addressed.

A. Causal Analysis and Contracts for WBS

This section reuses the existing safety assessment of WBS presented in Appendix L of the ARP4761 document. Building upon the existing hazard analysis from Appendix L, we identified failure condition *reduced responsiveness of wheel*

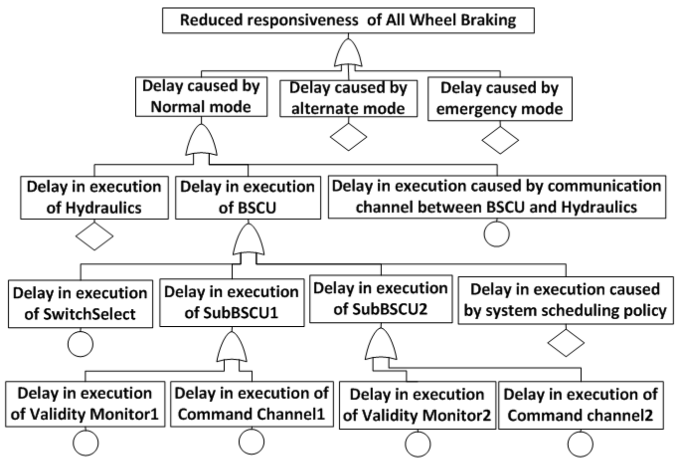
Reduced responsiveness of All Wheel Braking
Delay caused by Normal mode
Delay caused by alternate mode
Delay caused by emergency mode
Delay in execution of Hydraulics
Delay in execution of BSCU
Delay in execution caused by communication channel between BSCU and Hydraulics
Delay in execution of SwitchSelect
Delay in execution of SubBSCU1
Delay in execution of SubBSCU2
Delay in execution caused by system scheduling policy
Delay in execution of Validity Monitor1
Delay in execution of Command Channel1
Delay in execution of Validity Monitor2
Delay in execution of Command channel2

Fig. 2. Reduced responsiveness of all wheel braking Fault Tree

braking as hazardous, e.g., when it occurs during taxi phase it can lead to low-speed vehicle collision.

In order to prevent the delayed response from the brakes, we specify a timing safety requirement *SR1* that the WBS response time (i.e., time from the receipt of pedal brake signals to issuing the braking command) shall be no more than 10 ms. The fault tree in Fig. 2 addresses the reduced responsiveness failure condition. It shows that the delay in issuing the braking command can be caused by either of the three modes. The fault tree focuses on the normal mode and demonstrates that BSCU, Hydraulics or the communication channel between the two can all contribute to causing a delay in normal mode.

After identifying the hazards and specifying the requirements, the safety process continues to design the system to satisfy the specified requirements. Consequently, the safety contracts are captured to show compliance with the safety requirements. Strong safety contracts (denoted as a pair of strong assumptions and guarantees $\langle A, G\rangle$) allow us to specify behaviours that always must hold, i.e., strong assumptions (A) must be satisfied and strong guarantees (G) must be offered [18]. On the other hand, weak contracts (denoted as a pair of weak assumptions and guarantees $\langle B, H\rangle$) allow us to capture properties that change depending on the context in which the component is used. The weak guarantees (H) are offered only when all the strong contracts and the corresponding weak assumptions (B) are satisfied. The benefit of using the strong and weak contracts distinction is twofold: (1) it provides methodological distinction between properties that must hold and those that may hold in certain cases (e.g., weak contracts are used to describe multiple context-specific behaviours), and (2) when performing contract checking in a particular environment, violation of the strong assumptions is not tolerated, while violation of the weak assumptions is allowed (since some of the weak contracts might not be relevant for the particular context).

As the contracts need to be supported by evidence, we attach evidence information (E) with the contracts. We represent the contract/evidence pair as "C: $\langle A, G\rangle; E$", which can be read as follows: contract C, which under assumptions A offers guarantees G, is supported by evidence E. The motivation for connecting the evidence with the contracts is not to argue contract satisfaction (rationale description is needed for that), but to support change management. Besides identifying which parts of safety case are affected by change, safety contracts, when enriched by evidence information, can also be used to identify which evidence should be revisited. The evidence can be associated with a contract either directly, or indirectly through the associated contracts. Since the underlying contract formalism assumes hierarchical structure of components and contracts, all evidence needed to support a higher level contract are not associated with that contract directly, but can support the contract indirectly through the associated lower level contracts. The relation between a contract and its supporting contracts is established through the dependency assumptions.

WBS_Weak_1:
$\langle$ **B1:** *Platform=x and Compiler=y* AND Hydraulics delay $\leq$ 4 ms AND BSCU delay $\leq$ 4 ms AND communication delay $\leq$ 0.1 ms AND emergency mode $\leq$ 1 ms;
H1: WBS delay $\leq$ 10 ms $\rangle$;
E1: WBS timing analysis under assumed conditions

Fig. 3. WBS weak contract

WBS_BSCU_Weak_1:
$\langle$ **B2:** *Platform=x and Compiler=y* AND subBSCUx delay $<$ 3 ms and SelectSwitch delay $<$ 1 ms AND scheduler policy does not cause delay;
H2: BSCU delay $\leq$ 4 ms $\rangle$;
E2: BSCU timing analysis under assumed conditions

Fig. 4. BSCU weak contract

Using component-based development notions, such as contracts, within safety-critical systems has some difficulties. The out-of-context idea of safety contracts causes difficulties that relate to both the nature of safety as a system property and context dependent behaviours such as timing [8]. When it comes to the nature of safety and contracts, it is difficult to capture all failure scenarios that the component can contribute to since what is safety relevant in one system might not be in another. For example, the difficulty with capturing timing properties within out-of-context contracts is not only that timing depends on many factors, but that the timing analysis is usually calculated with incompatible or simplified assumptions [1], [9], [13], which makes the timing information captured within contracts nearly impossible to reuse. While the inevitable solution in that case would be to re-run the timing analysis, the information captured within contracts can still be useful in highlighting impact of the change on the safety case.

Based on the causal analysis we specify the contract *WBS_Weak_1* (Fig. 3). *WBS_Weak_1* contains dependency assumptions capturing connection between WBS and its subcomponents, while the guaranteed property is the response time of WBS. In order to guarantee timing properties, such as those noted in [8], we need to include additional assumptions that are not provided in the causal analysis. In case of *WBS_Weak_1* contract we included additional assumptions on platform and compiler configuration, as such assumptions can be easily omitted from the causal analysis, and any change or inconsistency related to these properties may invalidate the corresponding contracts. We can note that the causal analysis is useful for capturing dependency assumptions within the safety

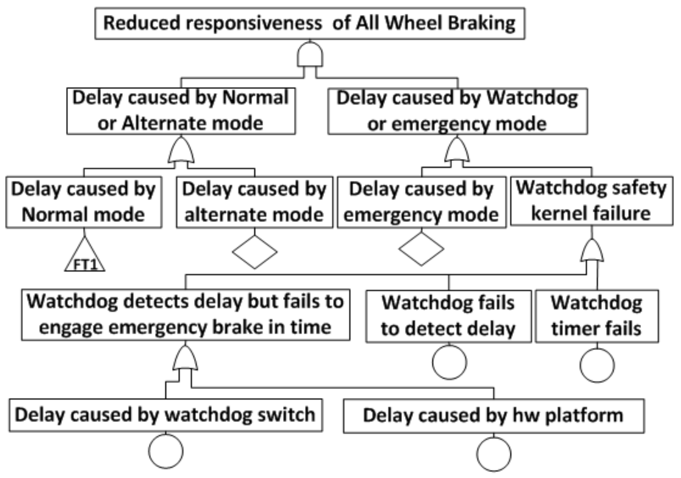
Reduced responsiveness of All Wheel Braking
Delay caused by Normal or Alternate mode
Delay caused by Watchdog or emergency mode
Delay caused by Normal mode
Delay caused by alternate mode
Delay caused by emergency mode
Watchdog safety kernel failure
FT1
Watchdog detects delay but fails to engage emergency brake in time
Watchdog fails to detect delay
Watchdog timer fails
Delay caused by watchdog switch
Delay caused by hw platform

Fig. 5. Reduced responsiveness of all wheel braking Fault Tree (updated)

contracts, but it is not sufficient as additional assumptions need to be captured as well. The Ariane 5 rocket is an example of how causality analysis does not cover some important assumptions. A piece of software that should perform certain computations right before liftoff was reused from the previous rocket version. Since restarting the software during liftoff might take time, the engineers decided to leave it running even after liftoff. The software then continued the unneeded computation during the flight time and caused an exception due to a floating-point error which rebooted the processor [11].

The contracts in Fig. 3 and 4 focus on the behaviour of WBS when there is no fault in the system. However the contracts don't describe behaviour of the system in situation when anomalous behaviours occur, e.g. when BSCU delay is greater than 4 ms or the communication channel fails. As mentioned earlier, it is difficult to describe behaviour of a component in all the failure scenarios, e.g. in some cases it is reasonable to consider communication channel failure in others it may not be the case. While the described behaviour in contracts *WBS_Weak_1* and *WBS_BSCU_Weak_1* can be useful to know in certain situations, it is very difficult to reuse such information in case of platform change or moving the component from one system to another, as argued earlier. That is why this behaviour is specified within a weak contract, as it cannot be guaranteed in all systems. Further on we investigate how these weak contracts can be complemented with strong contracts capturing behaviour that prevents bad things from happening that is guaranteed wherever the component is used.

B. Causal Analysis and Contracts on WBS with Safety Kernels

In the current design, the reduced responsiveness of WBS can be caused by either of the modes. In order to reduce the criticality of timing requirements in the Normal and Alternate modes to an appropriate level, a design decision was made to use a simple and sufficiently independent safety kernel. This safety kernel acts as a last resort failure mechanism in case of failures that might prevent Normal or Alternate mode from generating the braking command. The safety kernel in form of a watchdog timer is installed within Hydraulics component. Once WBS receives the pedal signals the watchdog timer is started. Unless either Normal or Alternate mode does not provide the braking command within the required time interval, the watchdog timer engages the emergency brake.

WBS_Strong_1:
$\langle$ **A1:** sufficient independence of the safety kernel and emergency brake from normal and alternate mode
G1: if braking command not received from normal or alternate mode before watchdog timer expiry then kernel activates the emergency brake $\rangle$;
E3: Causal analysis; Contract completeness report
WBS_Weak_1:
$\langle$ **B1:** (*Platform=x and Compiler=y* AND Hydraulics delay $\leq$ 4 ms AND BSCU delay $\leq$ 4 ms AND communication delay $\leq$ 0.1 ms) OR (watchdog timer expiry $\leq$ 9 ms and emergency brake delay $\leq$1 ms);
H1: WBS delay $\leq$ 10 ms $\rangle$;
E1: New WBS timing analysis under assumed conditions

Fig. 6. WBS contracts

WBS_SKC_Strong_1:
$\langle$ **A2:** -
G2: if the braking command signal not provided within 9 ms from the receipt of the pedal signals, then activate emergency brake within 1 ms $\rangle$;
E4: Formal verification report

Fig. 7. Safety Kernel strong contract

With introduction of the safety kernel in the WBS architecture, the initial FTA needs to be revisited to address both: changes to the criticality of Normal and Alternate modes; and extension of the current fault tree to include possible faults related to the kernel itself. The updated fault tree is shown in Fig. 5. The changes in the fault tree consequently influence the contracts to be revisited. More specifically, the *WBS_Weak_1* contract needs to be updated with the new information relating to the watchdog timer and the emergency brake. The updated *WBS_Weak_1* contract is shown in Fig. 6.

When using the safety kernels, we focus on capturing with the contracts how the component handles faults in the system. Due to simplicity of the kernel and its high independence from the rest of the system, we can specify strong safety contracts for the kernel that are easier to satisfy because of fewer assumptions. The strong contracts in Fig. 6 and 7 complement the weak contracts in Figures 3 and 4 by describing behaviour of the safety kernel when the normal or alternate mode fail. The assumption of sufficient independence in the contract *WBS_Strong_1* can be identified through the AND connection in the fault tree in Fig. 5 between *normal or alternate mode delays* and *kernel and emergency mode delays*. The corresponding guarantee describes the behaviour of the kernel in that situation. The *WBS_SKC_Strong_1* contract on the safety kernel addresses possible delay because of the kernel itself by guaranteeing its timing behaviour for all systems in which the kernel is used.

This example demonstrates that for the safety kernels we can specify the strong safety contracts with fewer assumptions (due to the simplicity and independence of the safety kernel). Fewer assumptions means that the corresponding contracts are easier to satisfy. Moreover, by reducing criticality of requirements addressed by the weak contracts, the stringency of evidence required to support the weak contracts is reduced. Consequently, overall less effort should be required for producing evidence to support such weak contracts.

C. Contract Derivation and Completeness Checking Methods

To talk about completeness of contracts we need to identify with respect to what should that completeness be checked. The safety contracts focus on failure behaviours of the system that can be obtained by failure analysis (e.g., FTA) as these are most often the causes of hazards. In this work we use FTA, a well-established method recommended by safety standards, for contract derivation and completeness check. Deriving contracts from fault trees is performed as follows:

1) Identify fault tree nodes directly related to the component for which the contract is being derived such that the nodes do not belong to each others sub-branches.
2) For each identified node:
   a) Create a safety contract that guarantees to prevent or minimise the faulty state described by the node.
   b) Identify candidate nodes for stating dependency assumptions such that the assumption node belongs to the same branch as the guarantee node, and that it refers to behaviour either of first level subcomponent of the current component, other components in the environment that the current component is connected to or other system properties.
3) The logical connection of the assumptions within the contract is switched comparing to the connection in the fault tree (e.g., AND connections in the fault trees become OR in the contracts), similarly as the guarantees can be regarded as negations of the corresponding nodes (e.g., a node "delay in execution" in a fault tree becomes a guarantee "does not cause delay in execution").

The assumptions on the first-level subcomponents are included to capture dependencies between the two layers identified by FTA, and in that way facilitate independent development and change management. For example, *BSCU* is independently developed by a contractor. Based on the specified dependency assumptions we can identify if the provided (or replaced) component offers required behaviour to achieve the WBS behaviour. This can be done by checking if the WBS dependency assumptions are satisfied by BSCU contracts.

Once the change occurs in the system or the component is moved to another system, the completeness of the contracts needs to be checked with respect to the fault trees. In our case, the contract *WBS_Weak_1* had to be changed after introducing the safety kernel as the contract was not complete with respect to the new fault tree in Fig. 5. Consequently, the evidence required to support this contract had to be updated.

Completeness with respect to a specific failure analysis does not imply contract completeness in general, but only with respect to the analysis. Confidence in the completeness check stems from the confidence in the failure analysis against which the check is performed. In our work we use FTA for completeness check under assumptions that producing fault trees is well-established and that the resulting fault trees are reasonably complete. It must be emphasised that the approach does not rely on the fault trees actually being complete, as

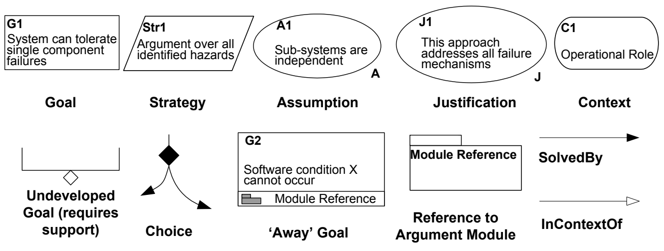
G1
System can tolerate single component failures
Goal
Str1
Argument over all identified hazards
Strategy
A1
Sub-systems are independent
A
Assumption
J1
This approach addresses all failure mechanisms
J
Justification
C1
Operational Role
Context
Undeveloped Goal (requires support)
Choice
G2
Software condition X cannot occur
Module Reference
'Away' Goal
Module Reference
Reference to Argument Module
SolvedBy
InContextOf

Fig. 8. Overview of the Goal Structuring Notation (GSN)

the aim is to de-risk change rather than have a change process where only contracts have to be checked following a change. The derived contracts usually require additional assumptions that can be derived from different analyses and used to enrich the contracts, hence increase their overall completeness. The contracts completeness check with respect to a specific analysis is performed to ensure that there are no inconsistencies between the dependencies captured within the contracts and those identified by the analysis. The results of such check can indicate that the contracts are incomplete with respect to the analysis (in case of changes to the system, and to the analysis), or the analysis can be incomplete with respect to the contracts (if we have enriched the contracts using other types of analyses). The contract completeness check with respect to the fault trees is performed as follows:

1) Identify nodes in the fault tree that correspond to the contract guarantees.
2) Identify nodes in the fault tree corresponding to the assumptions.
3) For the identified assumptions within the fault tree, check whether they belong to the branch corresponding to the identified node related to the guarantee.
4) Identify the following inconsistencies:
   a) Nodes that are included in the assumptions but do not belong to the same branch as the guarantee node.
   b) Nodes within the same branch as the guaranteed node that are not covered by the assumptions (not all nodes of the branch should be captured by assumptions but all should be covered, i.e., if the node itself is not included, then its sub-nodes or leaves of its branch should be included for the node to be covered).
5) If assumptions cover all nodes within the guarantees node branch then the contract is complete with respect to the fault tree, but if there are additional nodes that are assumed but do not belong to the same branch, the inconsistency should be reported as either fault tree is not complete, or the contract should be revised.

V. Safety Argument

In this section we present an overview of the graphical notation (section V-A) used to construct our arguments. The WBS safety argument is presented in section V-B.

A. Overview of Goal Structuring Notation

The Goal Structuring Notation (GSN) [12] – 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 Fig. 8 (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 ("solved by") sub-goals until a point is reached where claims can be supported by direct reference to available evidence. 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 [12]. GSN has been widely adopted by safety-critical industries for the presentation of safety arguments within safety cases. While GSN is mainly used to record monolithic safety arguments, an extension facilitates the creation of modular arguments. As a part of the modularised form of GSN, an away goal statement can be used to support the local claim by referring to a claim developed in another module. In this paper the modularised form of GSN, as first introduced in [2], [3], is used.

B. Wheel Braking System Safety Argument

Fig. 9 shows the safety argument fragment for WBS represented using GSN. The argument focuses on the timing requirement *SR1*: "WBS response time shall be no more than 10 ms", specified in Section IV and represented by the goal *WBSSafetyExeTime* within the argument. We base the argument that *SR1* is satisfied on the *WBSSWSafetyReq* justification that the software safety requirements are addressed by the safety contracts. Moreover we provide an away goal *SWSafetyContracts* presenting the required evidence to support safety contract consistency, their correctness with respect to the associated safety requirements and completeness with respect to the failure analysis. In the presented argument we focus on the product rather than the process by which we ensure that these contract properties are achieved.

Based on the *WBSSWSafetyReq* justification we address the *WBSSafetyExeTime* goal by the *WBS_weak_1* contract that supports the *SR1* requirement. In order to clarify the goal, we create a context statement to identify the *WBS_weak_1* contract that addresses the goal, and to provide a reference to WBS system description. To further develop the *WBSSafetyExeTime* goal, we use the dependency assumptions of the associated contract *WBS_weak_1* to identify the supporting sub-goals: *WBSWeakContract1.1*, *WBSHydraulicsDelay1.2* and *WBSWeakContract1.3*. The context statements for these sub-goals are provided in the same way as for the *WBSSafetyExeTime* goal. Further development of the sub-goals follows the same procedure as for the *WBSSafetyExeTime* goal, i.e. by identifying dependency assumptions of the associated contract to the particular goal, we derive sub-goals until we reach the lowest level component, i.e. where we have directly relevant evidence that supports the goal.

As WBS architecture changed with addition of the safety kernel, the corresponding safety argument needs to be updated as well. Based on the derived safety contracts for the safety kernel provided in Figures 6 and 7, we extend the safety argument from Fig. 9 with an additional supporting goal *WBSSafetyKernel* to the *WBSSafetyExeTime* claim, as shown in Fig. 10. The goal *WBSSafetyKernel* is clarified with context statements by referring to the corresponding contract *WBS_Strong_1* (Fig. 6), and providing definitions of the timer interval of 9 ms, and notions of emergency brake and safety kernel definition. The *WBSSafetyKernel* goal is further supported by an away goal *WBSSafetyKernelReliability* claiming that the kernel has been developed to meet the required reliability level, and a sub-goal *WBSDelaysWDogEmerg* based on the *WBS_SKC_Strong_1* contract.

VI. Summary and Conclusions

Means to capture failure behaviour within safety contracts have received little attention in contract-based approaches for safety-critical systems. Moreover, handling of inevitable contract incompleteness, implied by a great number of assumptions that need to be captured, is not sufficient for showing that the system is acceptably safe. We have presented a method for deriving safety contracts from fault tree analysis and demonstrated it on an example. The derived contracts from failure analysis results are at least as complete as the analysis itself. While particular analysis itself can be incomplete, different analyses can be used to enrich the contracts and increase their completeness. We have proposed a completeness check method that identifies inconsistencies between the contracts and the failure analysis and acts as guidance for change management. We have used safety kernels to demonstrate how simple components characterised by high independence from the rest of the system allow for capturing the components safety contracts with fewer assumptions. Such contracts are easier to reason about in a more rigorous manner, both in terms of difficulty of capturing such contracts and representing them.

Future work will focus on developing safety contract-based change management techniques, which should cover both the safety argument and associated evidence. Furthermore, we plan to investigate techniques for identifying additional assumptions needed to enrich the contracts derived from failure analysis. To make the approach useful in production environments, we plan to provide tool support for both deriving and checking the contracts, and safety case change management guidance using the derived safety contracts.

Acknowledgement

We acknowledge the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project.

References

[1] I. Bate and A. Burns. An integrated approach to scheduling in safety-critical embedded control systems. *Real-Time Systems Journal*, 25(1):5–37, Jul 2003.

[2] I. Bate and T. Kelly. Architectural considerations in the certification of modular systems. In *Proceedings of the Computer Safety, Reliability and Security - 21st International Conference, SAFECOMP 2002*, volume LNCS 2434, pages 321–333, 2002.

[3] I. Bate and T. Kelly. Architectural considerations in the certification of modular systems. *Reliability Engineering and System Safety*, 81:303–324, 2003.

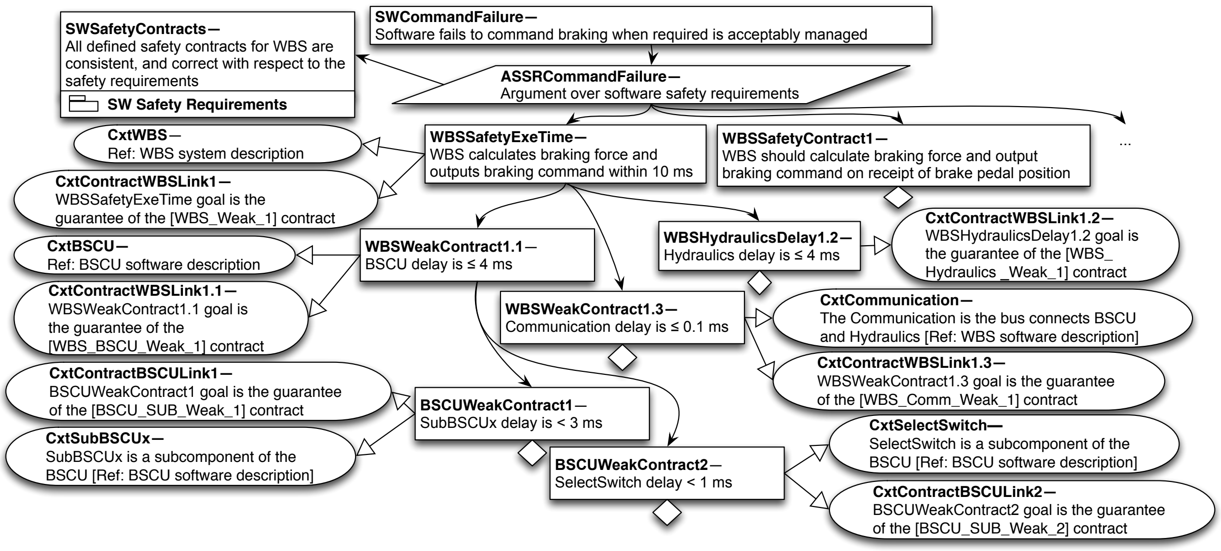
SWSafetyContracts—
All defined safety contracts for WBS are consistent, and correct with respect to the safety requirements
SW Safety Requirements
SWCommandFailure—
Software fails to command braking when required is acceptably managed
ASSRCommandFailure—
Argument over software safety requirements
CxtWBS—
Ref: WBS system description
WBSSafetyExeTime—
WBS calculates braking force and outputs braking command within 10 ms
WBSSafetyContract1—
WBS should calculate braking force and output braking command on receipt of brake pedal position
...
CxtContractWBSLink1—
WBSSafetyExeTime goal is the guarantee of the [WBS_Weak_1] contract
CxtBSCU—
Ref: BSCU software description
WBSWeakContract1.1—
BSCU delay is ≤ 4 ms
WBSHydraulicsDelay1.2—
Hydraulics delay is ≤ 4 ms
CxtContractWBSLink1.2—
WBSHydraulicsDelay1.2 goal is the guarantee of the [WBS_ Hydraulics _Weak_1] contract
CxtContractWBSLink1.1—
WBSWeakContract1.1 goal is the guarantee of the [WBS_BSCU_Weak_1] contract
WBSWeakContract1.3—
Communication delay is ≤ 0.1 ms
CxtCommunication—
The Communication is the bus connects BSCU and Hydraulics [Ref: WBS software description]
CxtContractBSCULink1—
BSCUWeakContract1 goal is the guarantee of the [BSCU_SUB_Weak_1] contract
BSCUWeakContract1—
SubBSCUx delay is < 3 ms
CxtContractWBSLink1.3—
WBSWeakContract1.3 goal is the guarantee of the [WBS_Comm_Weak_1] contract
CxtSubBSCUx—
SubBSCUx is a subcomponent of the BSCU [Ref: BSCU software description]
CxtSelectSwitch—
SelectSwitch is a subcomponent of the BSCU [Ref: BSCU software description]
BSCUWeakContract2—
SelectSwitch delay < 1 ms
CxtContractBSCULink2—
BSCUWeakContract2 goal is the guarantee of the [BSCU_SUB_Weak_2] contract

Fig. 9. WBS safety argument before introducing the safety kernel

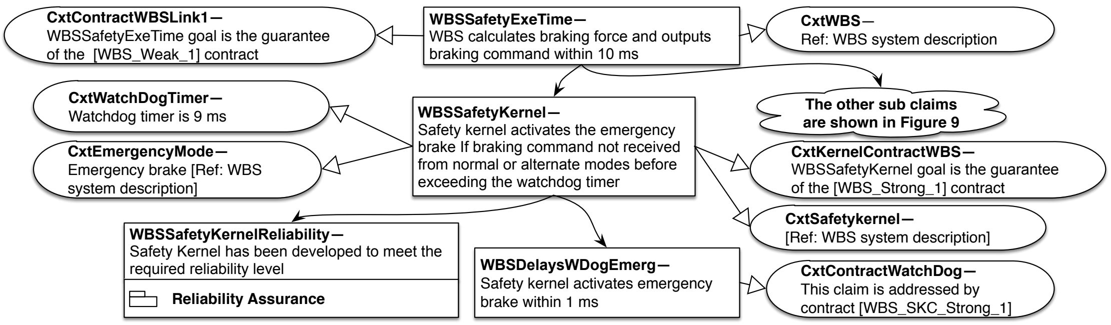
CxtContractWBSLink1—
WBSSafetyExeTime goal is the guarantee of the [WBS_Weak_1] contract
WBSSafetyExeTime—
WBS calculates braking force and outputs braking command within 10 ms
CxtWBS—
Ref: WBS system description
CxtWatchDogTimer—
Watchdog timer is 9 ms
WBSSafetyKernel—
Safety kernel activates the emergency brake If braking command not received from normal or alternate modes before exceeding the watchdog timer
The other sub claims are shown in Figure 9
CxtEmergencyMode—
Emergency brake [Ref: WBS system description]
CxtKernelContractWBS—
WBSSafetyKernel goal is the guarantee of the [WBS_Strong_1] contract
WBSSafetyKernelReliability—
Safety Kernel has been developed to meet the required reliability level
Reliability Assurance
CxtSafetykernel—
[Ref: WBS system description]
WBSDelaysWDogEmerg—
Safety kernel activates emergency brake within 1 ms
CxtContractWatchDog—
This claim is addressed by contract [WBS_SKC_Strong_1]

Fig. 10. The updated WBSSafetyExeTime goal after introducing the safety kernel

[4] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In *Formal Methods for Components and Objects*, volume 5382 of *Lecture Notes in Computer Science*, pages 200–225, 2007.

[5] A. Cimatti and S. Tonetta. A property-based proof system for contract-based design. In *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 21–28. IEEE Computer Society, 2012.

[6] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2011.

[7] J. L. Fenn, R. D. Hawkins, P. Williams, T. P. Kelly, M. G. Banner, and Y. Oakshott. The who, where, how, why and when of modular and incremental certification. In *Proceedings of the 2nd IET International Conference on System Safety*, pages 135–140. IET, 2007.

[8] P. Graydon and I. Bate. On the nature and content of safety contracts. In *Proceedings of the 15th IEEE International Symposium on High Assurance Systems Engineering*, pages 245–246, January 2014.

[9] P. Graydon and I. Bate. Realistic safety cases for the timing of systems. *The Computer Journal*, 57(5):759–774, May 2014.

[10] O. Jaradat, P. J. Graydon, and I. Bate. An approach to maintaining safety case evidence after a system change. In *Proceedings of the 10th European Dependable Computing Conference (EDCC)*, August 2014.

[11] J.-M. Jézéquel and B. Meyer. Design by contract: The lessons of ariane. *IEEE Computer*, 30(1):129–130, Jan. 1997.

[12] T. Kelly. *Arguing Safety - A Systematic Approach to Safety Case Management*. PhD thesis, Department of Computer Science, University of York, 1999.

[13] T. Kelly, I. Bate, J. McDermid, and A. Burns. Building a preliminary safety case: An example from aerospace. In *Proceedings of the 1997 Australian Workshop on Industrial Experience with Safety Critical Systems and Software*, October 1997.

[14] T. Kelly and J. McDermid. A systematic approach to safety case maintenance. *Reliability Engineering and System Safety*, 71(3):271 – 284, 2001.

[15] Modular software safety case (MSSC) — process description. https://www.amsderisc.com/related-programmes/, Nov 2012.

[16] J. Rushby. *Kernels for Safety?*, chapter 13, pages 210–220. Blackwell Scientific Publications, 1989.

[17] SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Dec. 1996.

[18] I. Sljivo, B. Gallina, J. Carlson, and H. Hansson. Strong and weak contract formalism for third-party component reuse. In *Proceedings of the 3rd International Workshop on Software Certification*, pages 359–364, 2013.

[19] K. Wika and J. Knight. On the enforcement of software safety policies. In *Proceedings of the 10th Annual IEEE Conference on Computer Assurance*, June 1995.

[20] B. Zimmer, S. Bürklen, M. Knoop, J. Höfflinger, and M. Trapp. Vertical safety interfaces–improving the efficiency of modular certification. In *Proceedings of the 30th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, pages 29–42, 2011.