Strong and Weak Contract Formalism for Third-Party Component Reuse

Irfan Sljivo, Barbara Gallina, Jan Carlson, Hans Hansson
Mälardalen Real-Time Research Centre, Mälardalen University,
Västerås, Sweden
{irfan.sljivo, barbara.gallina, jan.carlson, hans.hansson}@mdh.se

***Abstract*—Our aim is to contribute to bridging the gap between the justified need from industry to reuse third-party components and skepticism of the safety community in integrating and reusing components developed without real knowledge of the system context. We have developed a notion of safety contract that will help to capture safety-related information for supporting the reuse of software components in and across safety-critical systems.**

In this paper we present our extension of the contract formalism for specifying strong and weak assumption/guarantee contracts for out-of-context reusable components. We elaborate on notion of satisfaction, including refinement, dominance and composition check. To show the usage and the expressiveness of our extended formalism, we specify strong and weak safety contracts related to a wheel braking system.

I. Introduction

More and more standards for certification of safety-critical systems are offering support for reuse of third-party components in order to reduce time-to-market and production costs. An example is represented by the introduction of the concept *Safety Element out of Context* (SEooC) within the automotive ISO26262 standard [9]. Although this opportunity to reduce time-to-market and production costs seems attractive, reuse of third-party components is challenged by various complications [10] and can easily incur additional costs. One of the major problems for the safety-related systems is that the context in which the reusable out-of-context component is going to be used is unknown. On the one hand, if we include too much information about the context in the reusable component than it will be more difficult to reuse it in a different context.

Component reuse within standards is present through the use of commercial off-the-shelf (COTS) and modified off-the-shelf (MOTS) items. The drawback of the off-the-shelf items is usually that they lack the development process evidence required for certification by the domain-specific safety standards [13]. The basic idea behind SEooC is to bridge that gap by allowing the developer to first assume the safety-related requirements applicable to a component, and then to develop it to satisfy those requirements. Contract-based approaches emerge as one of the means to capture safety requirements and enable reuse and composition within safety-critical systems [3]–[8]. A contract is a set of assumptions and guarantees where guarantees are provided by the component if assumptions are met by the component's environment.

In our work, we are looking into means for capturing as much as possible of safety-related information for reuse, but still to keep the component more flexible, i.e., reusable. We provide a further developed formalism for safety contracts with strong and weak reasoning that enables capturing information that need to hold for all contexts, i.e., that are out-of-context, and information that are more context-specific.

To improve reuse possibilities of software components by using contracts, just as components need to be designed for reuse, so do contracts as well. For these purposes it is beneficial to provide more expressive means of capturing information for reuse. In our previous work [14], we have introduced fine-grained contract extension with strong and weak contracts reasoning. The strong contracts must always hold in order for components to be reused (in any context), while the weak contracts just offer additional information about a context in which the component can operate. For example, information such as timing are highly context-specific and should be specified separately from the conditions that are needed for the component to operate.

In this paper we use a wheel braking system as an example of a safety-critical system to show how fine-grained contract reasoning can be used to capture timing and safety information. The system is originally used within ARP4761 airborne systems recommended practice [1] to demonstrate the safety process required for the airspace domain.

The main contribution of this work is extension and adaptation of contract semantics to handle strong and weak contracts. We associate each component with a set of strong and weak contracts and define conjunction of strong and weak contracts. The format of the contract in conjuncted form is based on our previous work [14], where a contract consists of strong assumptions, strong guarantees and multiple weak assumption/guarantee pairs. In this work, we define notions of satisfaction, refinement and dominance for contracts in the conjuncted format. Further more, we show the usage of the fine-grained contracts on the wheel braking system.

Comparing to related work, we are focusing more on capturing contracts for out-of-context components where very little, or no information at all is known about the context in which the component is supposed to operate. We are putting emphasis on the contents of out-of-context contracts and the separation of mandatory and alternative/optional properties.

The rest of the paper is organized as follows: In Section II we briefly present key notions we build upon and provide essential information on the wheel braking system. We extend and adapt the fine-grained contract formalism in Section III. In Section IV we use our extended formalism to specify contracts related to the wheel braking system. Related work is presented in Section V and conclusions and future work in Section VI.

II. Background

In this section we briefly provide some background information on off-the-shelf items for safety-critical systems, support for reuse from safety standards, and assumption/guarantee contracts. Finally, we also provide essential information related to the wheel braking system.

A. *Off-The-Shelf Items*

Off-the-shelf (OTS) solutions offer reduced time-to-market and increased affordability, and are expected to support services with multiple safety-criticality levels [11]. There are many types of OTS items including commercial OTS, modified OTS, Software of Unknown Pedigree (SOUP) etc. While ones are developed according to standards - COTS and MOTS, the others are not - SOUP. On the other hand, some are to be used "as is" without changes - COTS, and some can be modified and changed - MOTS.

The use of OTS items within safety-critical systems has been debated for years [13], since most of the safety-critical systems need to be certified by a domain-specific safety standard that requires some kind of evidence about the safety of the system, that usually doesn't come with OTS items. As all the other reusable components, OTS items as well suffer from the three basic issues in the creation and use of reusable components illustrated by the well-known "3C's Model" [12]: Concept, Content and Context. The third issue is the most problematic for the safety community when it comes to reusable components. The problem of context is usually addressed by the concept of *separation of concerns*, where different aspects of a component are kept as independent as possible to maximize the reuse potential of the component. Due to the system-wide nature of the safety-related properties it is impossible to completely separate concerns in the context of safety-related systems.

B. *Safety Standards and Reuse*

Safety standard authorities have been making an effort to bridge the gap between the separation of concerns and the system-wide nature of safety properties. As mentioned in the introduction, an example is represented by the introduction of the concept *Safety Element out of Context* (SEooC) within the automotive ISO26262 standard. A SEooC is a safety-related element which is not developed for a specific item, but is developed based on "assumptions on an intended functionality, use and context" [9].

Within avionics domain, regulated by DO178B(C) safety standard, the regulatory agency introduced the concept of *Reusable Software Component* (RSC) [2]. The concept allows developers of RSC to satisfy only a part of requirements mandated by the safety standard, while the integrator of the developed RSC is expected to complete the safety standard objectives.

Besides the above-mentioned problem with separation of concerns, another problem that occurs is the criticality of the components developed out-of-context. Safety Integrity Level (SIL) represents a measurement for quantifying risk reduction. Different safety standards have different SIL categorizations that range from events that have no risk involved to events that may result in harm to human life and can be classified as hazardous and catastrophic. The components not only need to be developed according to a safety standards, but they usually must be developed at a specific SIL. Within the automotive industry standards, this problem is addressed by ASIL decomposition, where ASIL is in fact Automotive SIL. ASIL decomposition allows a developer to use a component with a lower SIL by attaching a safety function with the same lower SIL and showing that the two are independent. The avionics industry defines five SILs and refers to them as *Design Assurance Level* (DAL). DALs are categorized from catastrophic failure conditions denoted with DAL A to failure conditions that have no effect on safety denoted with DAL E.

C. *Fine-grained Contracts*

Traditional assumption/guarantee contract is a pair of assertions $C = \langle A, G \rangle$ where a component makes assumptions A on its environment and if those assumptions are met it offers guarantees G in return. Contract semantics are defined in terms of environments and implementations. It is said that an environment satisfies a contract $C = \langle A, G \rangle$ if it provides all of the contract assumptions A. An implementation satisfies a contract C if provided the assumptions A it satisfies the guarantees G.

As we mentioned in our previous work [14], moving properties captured in-context to out-of-context reusable component is a difficult work because many implicit and hidden assumptions need to be identified about the specific context, for the property to hold out-of-context. That is why we extended the traditional contract-based formalism to allow for distinguishing between properties that are context-specific and properties that must hold for all contexts by adapting strong and weak contract reasoning. The extended contract format consists of strong assumptions and guarantees and multiple weak assumption/guarantee pairs. While the strong assumptions and guarantees must be satisfied always in order for component to be used, the weak pairs offer additional information in some specific contexts where besides the strong assumptions, the weak assumptions are to be met as well.

D. *Motivating Example*

In this subsection we provide essential information related to the wheel braking system that we use to show the usage and expressiveness of our extended formalism. This information is based on [1] and is taken from two previous works [7] and [6].

The example describes a Wheel Braking System (*WBS*) within an aircraft that takes two input brake pedal signals and outputs the brake signal that is applied on the wheel. The high level architecture is shown in Figure 1.

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 them to *Hydraulics* subsystem that executes the braking commands. If the *BSCU*, which makes the normal operation mode, fails then *Hydraulics* uses an alternate or emergency mode to perform the braking.

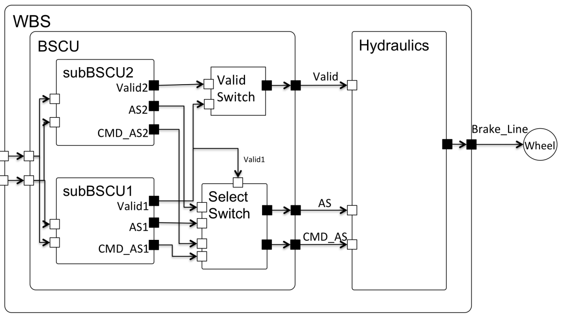

Fig. 1. Wheel Braking System - High Level View

The *WBS* is designed so that it addresses requirement that loss of all wheel braking is less probable than 1.0E-7 per flight hour ("loss of all wheel braking" failure condition is classified as hazardous). In order to address the availability and integrity requirements imposed on *BSCU*, *BSCU* is designed with two redundant dual channel systems: *subBSCU1* and *subBSCU2*, shown in Figure 2. 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.

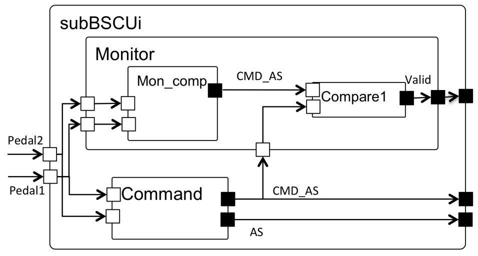

Fig. 2. SubBSCUi

In this work we use contracts to capture safety and timing properties of the system. The timing requirement on the system is that its execution is no more than 10ms. We will detail more about what is needed to be assumed for this requirement to be guaranteed in the Section IV. The addressed safety requirement is that no single failure within the *BSCU* shall lead to "inadvertent braking due to *BSCU*".

III. Fine-grained contracts further development

In this section we extend the theoretical foundations of the fine-grained contracts presented in [14] and define contract relations and operations.

Contract-based approaches usually assume that a number of contracts in the form of assumption/guarantee pairs is attached to a component. The different contracts can be associated with different aspects or viewpoints of the system. We distinguish between properties that must hold in all contexts and properties that are context-specific by categorizing the contracts associated to components into strong and weak contracts. Strong contracts $\langle A, G \rangle$ are composed of strong assumptions (A) and strong guarantees (G), and weak contracts $\langle B, H \rangle$ of weak assumptions (B) and weak guarantees (H). While strong assumptions must hold in order for a component to be used in any context, weak assumptions and guarantees just provide additional information for particular contexts. The weak contracts ensure that in particular contexts satisfying the strong (A) and the weak assumptions (B), the component offers the weak guarantees (H).

For the purpose of defining operations and relations on the component contract we need to conjunct the weak and strong contracts to form a single component contract. For the conjuncted contract C we use the format presented in [14]:

$$\langle A, G, \{\langle B_1, H_1 \rangle, \ldots, \langle B_n, H_n \rangle\}\rangle$$

where A and G are above-mentioned strong assumptions and guarantees, and $\{\langle B_n, H_n \rangle\}$ represent a set of weak assumption/guarantee pairs i.e., weak contracts. Since all strong assumptions define a single set of environments ($\mathcal{E}_C$) in which the component can operate, we conjunct all strong assumptions into a single strong assumption (A) and all strong guarantees into a single strong guarantee (G). Each weak contract is valid in only a subset ($\mathcal{E}_n$) of that single set of environments defined by strong assumptions, i.e., $\mathcal{E}_n \subseteq \mathcal{E}_C$. Hence we don't conjunct weak contracts that are valid in different subsets but represent them in the contract as multiple weak assumption/guarantee pairs.

A. Contract relations and operations

For a contract C in the conjuncted form we say that an environment E is satisfying the contract if it satisfies the strong assumptions (A) i.e. if $E \in \mathcal{E}_C$. We refer to environments that satisfy the contract as correct environments. The set $\mathcal{E}_C$ is the set of all correct environments of contract C.

Any environment $E \in \mathcal{E}_C$ can satisfy some weak assumptions and be incompatible with others. The more weak assumptions there are satisfied by the environment E the more information about the component behaviour described by the weak contracts can be reused in this context.

The rich component concept we are assuming encompasses both implementation and contracts. We say that a component implementation I satisfies a contract $C = \langle A, G\{\langle B_n, H_n \rangle\}\rangle$ under two conditions: (1) implementation I satisfies the strong guarantees G in all correct environments of C, and (2) for all weak pairs within C, in all environments $E \in \mathcal{E}_C$ satisfying weak assumptions B_n, the implementation I satisfies corresponding weak guarantees H_n. An implementation I that satisfies the contract C is called a correct implementation of the contract C.

We assume a hierarchical component structure where a component can be primitive, i.e., atomic, or composite, i.e., consisting of subcomponents. By composing subcomponents we must ensure that resulting component implementation and contract holds. We check composition consistency of a composite component and its subcomponents with contracts in conjuncted form by (1) checking that the strong assumptions that are not satisfied by the interconnected subcomponents are assumed by the strong assumptions of the composite, and

(2) checking that the composite contract follows from the subcomponent contracts.

For two traditional assumption/guarantee contracts C_1 and C_2 we get a composition contract by (1) composing assumptions that are not satisfied by the interconnected components, and (2) intersection of the guarantees. We compose two contracts in conjuncted form by (1) composition of the strong pairs, and (2) composition of the weak pairs such that there exists at least one environment satisfying the resulting weak pair within the set of all correct environments of the resulting contract, i.e., the intersection of the set of environments satisfying strong assumptions and the set of environments satisfying weak pair is not empty.

Relations of dominance and refinement are essential for checking composition and decomposition of contracts. We adapt the notion of dominance and refinement from [3] and [6] by including the weak and strong contract reasoning. Refinement coincides with weakening of assumptions and strengthening the guarantees within traditional assumption/guarantee contracts. While refinement of contracts in traditional form can be applied to strong and weak contracts individually, we say that refinement holds for two contracts C and C_1 in conjuncted form where C_1 refines C by (1) checking that the strong assumption/guarantee pair $\langle A_1, G_1\rangle$ of C_1 refines strong assumption/guarantee pair $\langle A, G\rangle$ of C, and (2) that for each weak pair $\langle B, H\rangle$ within C such that B is related to C_1 there is at least one strong or weak contract within C_1 such that it refines or implies $\langle B, H\rangle$.

We say that composite component contract C dominates subcomponent contracts C_1 and C_2 if: (1) composition of any correct implementations of C_1 and C_2 forms a correct implementation of C, (2) for every subcomponent contract C' we say that correct implementations of other subcomponent contracts and a correct environment of the composite contract constitute a correct environment for the subcomponent contract C'. This means that for every weak assumption/guarantee pair $\langle B, H\rangle$within C such that B is related to a subcomponent C_n there is at least one strong or weak contract within C_n that implies or refines it. With our updated correct environment and implementation notions, conditions for checking dominance stay the same as in [3] and [6].

IV. Case Study

In this section we show the usage and expressiveness of the extended contract formalism using the strong and weak contracts on the wheel braking system described in Section II-D. We show that specifying contracts to provide better support for reuse requires additional constructs for contract specification and that the proposed contract formalism is more expressive for capturing this information.

In this example we use contracts to capture safety and timing analysis of WBS. We use contract language based on patern-based Requirement Specification Language as used in [7] and Othello System Specification from [6]. For specifying timing properties we use $Change(\{P\})$ for the event of change of the ports $\{P\}$, and $Delay\ between(p1, p2)$ to specify delay between the two changes $p1$ and $p2$.

As mentioned in our previous work [14], capturing timing information for reuse purposes requires additional constructs in identifying a set of assumption required for the reused timing information to be valid. In our work, we specify timing properties for reusable components within weak contracts i.e., weak assumption/guarantee pairs, because the timing information may be reused only if all of the assumptions related to timing are met, otherwise we can not reuse the behaviour of the component as described by the corresponding guarantees. Specifying timing information within weak contracts allows us to describe the timing behaviour of the component in different alternative contexts, e.g., timing properties of a component for different compilers or different compiler configurations. We present all contracts for this example in the conjuncted form.

A. Usage of the strong and weak contracts

TABLE I. WBS CONTRACT

A: *Pedal1==Pedal2*
G: -
$\{\langle$**B1:** Platform==x and Compiler==y;
H1: Delay between ($Change$(*Pedal1,Pedal2*), $Change$(*Brake_Line*)) $\leq$ 10ms$\rangle$;}

The *WBS* component is composed of *BSCU* and *Hydraulics* subcomponents as shown in Figure 1, Section II-D. The WBS component contract in Table I describes a set of environments in which the system component can work by imposing the constraint that the two input pedals must be the same, otherwise all the mechanisms and redundancy within the WBS make no sense. As an additional information describing the component for certain correct environments, we make timing contract within weak assumption/guarantee pair by stating that for this particular platform, compiler and compiler configuration the component terminates within 10ms. This does not mean that the component shouldn't be used in an environment that doesn't satisfy those weak assumptions, but just that the timing behaviour of the component in that environment is known.

Tables II and III show *BSCU* and *Hydraulics* component contracts. While *BSCU* contract states that it requires the input pedals to be equal for the component to be able to operate, the *Hydraulics* contract requires only that the correct *Valid* signal from *BSCU* is received. We can see that composition of the subcomponents is correct since *WBS* takes *Pedal1==Pedal2* assumption from the *BSCU* subcomponent contract, since it cannot be satisfied by the interconnected components, while *BSCU.Valid* assumption from *Hydraulics* contract is satisfied by the interconnected *BSCU* component.

TABLE II. BSCU CONTRACT

A: Pedal1==Pedal2
G: -
$\{\langle$**B1:** (SubBSCU1.Valid or SubBSCU2.Valid);
H1: *BSCU.Valid* $\rangle$;
$\langle$**B2:** Platform==x and Compiler==y;
H2: Delay between ($Change$(*Pedal1,Pedal2*), $Change$(*CMD_AS,AS*)) $\leq$ 5ms$\rangle$;}

The timing contracts in Tables I, II and III are defined for the same environment described by the assumed platform, compiler and compiler configuration. In order for the decomposition to be correct, the dominance should hold. The first

TABLE III. HYDRAULICS CONTRACT

A: BSCU.Valid;
G:-

$\{\langle$**B1:** Platform==x and Compiler==y
H1: Delay between ($Change$(*Valid*), $Change$(*Brake_Line*)) $\leq$ 5ms$\rangle$;$\}$

condition for dominance specifies that correct implementations of *BSCU* and *Hydraulics* contracts form a correct implementation of *WBS* contract, which ensures that the timing contract of the composite is implied by the timing contracts of the subcomponents. Based on the refinement relation defined in Section III-A, this way we imply that both related components *BSCU* and *Hydraulics* of *WBS* timing contract must either have a contract that refines or implies it. Refinement between the *WBS* contract and the subcomponent contracts holds since the strong pair is refined by the subcomponent contract strong pairs and the timing pair is refined and implied by the timing pairs of the subcomponents contracts.

TABLE IV. SUBBSCUI CONTRACT

A: *Pedal1==Pedal2*
G: -

$\{\langle$**B1:** no fault in *Monitor*;
H1: *SubBSCUi.Valid* $\rangle$;
$\langle$**B2:** (*Monitor* developed to DAL A);
H2: *SubBSCUi.Valid* with high confidence$\rangle$;
$\langle$**B3:** Platform==x and Compiler==y;
H3: Delay between ($Change$(*Pedal1*,*Pedal2*), $Change$(*Valid*,*CMD_AS*,*AS*)) $\leq$ 4ms$\rangle$;$\}$

TABLE V. SUBBSCUI.MONITOR CONTRACT

A: *Pedal1==Pedal2*
G:*Monitor* developed according to DAL A;

$\{\langle$**B1:** Platform==x and Compiler==y;
H1: Delay between ($Change$(*Pedal1*,*Pedal2*), $Change$(*Valid*)) $\leq$ 2ms$\rangle$;$\}$

TABLE VI. SUBBSCUI.COMMAND CONTRACT

A: *Pedal1==Pedal2*
G: *Command* developed according to DAL B;

$\{\langle$**B1:** Platform==x and Compiler==y;
H1: Delay between ($Change$(*Pedal1*,*Pedal2*), Change(*CMD_AS*,*AS*)) $\leq$ 1ms$\rangle$;$\}$

In the previous works done on this system by [7] and [6], the safety requirement that "no single failure within *BSCU* shall cause inadvertent braking" is specified as "No_Double_Fault" variable meaning that always at least three out of four components within *BSCU* (two *Monitors* and two *Commands*) work correctly. This assumption is a direct representation of the requirement that no single failure shall cause BSCU to fail, by imposing too strict requirement on the system. The BSCU can handle if more than one component of the four within *BSCU* fails, so for example if both *Command* components fail, the *Monitors* can still report the error and provide correct *Valid* signal. Another issue with the way safety contracts have been captured is separation of concerns. By

TABLE VII. SUBBSCUI.SELECTSWITCH CONTRACT

A: -
G: always terminates;

$\{\langle$**B1:** (Platform==x and Compiler==y) AND Valid1==TRUE;
H1: Delay between ($Change$(*CMD_AS1*,*AS1*), $Change$(*CMD_AS*,*AS*)) $\leq$ 0,25ms$\rangle$;$\}$
$\langle$**B2:** (Platform==x and Compiler==y) AND Valid1== FALSE;
H2: Delay between ($Change$(*CMD_AS1*,*AS1*), $Change$(*CMD_AS*,*AS*)) $\leq$ 1ms$\rangle$;$\}$

using the "No_Double_Fault" variable on *WBS* level, authors are not respecting the encapsulation of *SubBSCUi* level by making assumptions about its internal structure on higher levels.

Our specification of the above-mentioned safety requirement can be seen through Tables II, IV, V andVI. We assume that no external fault is propagated through the pedal signals and that faults in this context refer to internal faults. In that case we can guarantee that the correct *Valid* signal will be provided by the *BSCU* if either of its subcomponents *SubBSCU1* or *SubBSCU2* provide the correct *Valid* signal, as assumed in *BSCU* contract in Table II. For the reasoning to hold, the subcomponents *SubBSCU1* and *SubBSCU2* guarantee the *Valid* signal only when the corresponding *Monitor* subcomponent is fault-free, Tables IV and V. To be able to guarantee *Valid* signal with a certain confidence, we capture the required DAL of *Monitor* within a weak pair. This way by respecting the separation of concerns and making assumptions only on assumed externally visible properties of interconnected and subcomponents, we allow for extra flexibility of the system, hence better reuse possibilities.

On the primitive components level, that are not composed of subcomponents, we sometimes must make guarantees that are based on the external evidence i.e., guarantees that do not follow from assumptions. For example, for components *Monitor* and *Command* we make a statement/guarantee about the component that says that the component is developed according to a specific DAL. This information is essential when it comes to completing the safety contract structure. Some components can be considered reliable, such as *SelectSwitch*, where we specified that it always terminates as its strong guarantee in Table VII. The timing contract of *SelectSwitch* component specifies its behaviour for two alternative situations in which the component has different timing behaviour. It performs much faster when it just forwards the values by default from *SubBSCU1* (*Valid1* $== TRUE$), than when it must switch to the redundant component *SubBSCU2* (*Valid1* $== FALSE$).

B. Discussion on benefits of the extended formalism

Developing a new or moving an existing component to an out-of-context setting implies capturing increased number of assumptions and guarantees that are used to describe the behaviour of the component in different contexts, as we can see on the timing contracts examples from Section IV-A. Accordingly, the introduction of the additional constructs with the extended formalism offers us with possibility to specify conditions that are out-of-context i.e., that must be satisfied by any environment in order for component to operate or offer any kind of reuse in that environment (strong assumptions).

Then, within all of those environments in which the component can operate and offer reuse, we have the possibility to specify conditions using weak contracts that describe behaviour of the component in some of the correct environments. For example, we use weak contracts i.e., weak assumption/guarantee pairs, to specify timing behaviour in different environments, or safety behaviour under different failure conditions, but all of these information can only be reused after the strong assumptions are met by the environment in which we use the component. As can be seen through contract examples in Section IV-A, this way of capturing context-specific information allows for extra flexibility of the system that enhances reuse possibilities.

V. Related Work

Contract-based design has been a research topic of many works in the recent years [3]–[8]. These works are largely based on developing a theoretical foundation for contract-based framework and creating verification techniques for contract-based design. These works mainly focus on an Original Equipment Manufacturer (OEM)-supplier relationship. Through the examples provided in [4], [6], [7] and the formalisms [3]–[5], [8] we could notice the lack of focus on specifying contracts for out-of-context components that are planned to be instantiated or used in different contexts. When an OEM is developing a component for a supplier, OEM usually has some requirements and demands about the component it needs to develop, which means that the context in which the component is supposed to operate is not completely unknown and many assumptions can be omitted since they are implied. When we want to actually move an in-context component to an out-of-context setting, or develop an out-of-context component, the number of properties that need to be captured increases and a more expressive way of capturing them is needed.

Comparing to the related work, in this paper we build on our previous work [14] and further extend the contract-based formalism [3] and [6] to provide more expressiveness for specifying contracts for out-of-context components. We additionally use an example that was used by [7] and [6] to show the usage of the strong and weak contracts for specifying out-of-context component contracts.

VI. Conclusion and Future Work

We have presented our extended contract formalism for specifying strong and weak contracts to support reuse of safety-related information within safety-critical systems. We specify strong and weak contracts for components that are developed or moved to out-of-context setting, where very little or no information is known about the contexts of the component. We distinguish between properties that must hold for all contexts and properties that are more context-specific and are specified as additional or optional properties. The introduced additional constructs provide us with the possibility to capture context-specific information within contracts but still retain system flexibility that is needed to offer better reuse possibilities. Moreover, we define relations of satisfaction for implementations and environments in terms of strong and weak contracts, as well as relations of refinement and dominance between contracts. Finally, we use a wheel braking system as an example of a safety-critical system to demonstrate the usage and expressiveness of the extended formalism.

In our future work we plan to extend one of the existing contract languages such as Requirement Specification Language or Othello System Specification to support the presented extended formalism with strong and weak contracts. Further on, we see possibilities to establish a closer relation between the contracts in the presented form and safety argumentation used within the certification process of safety-critical systems.

Acknowledgements

This work is supported by the Swedish Foundation for Strategic Research (SSF) project SYNOPSIS and the EUs Artemis-funded SafeCer project.

References

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

[2] AC 20-148. *Reusable Software Components*. Federal Aviation Administration, December 2004.

[3] S. S. Bauer, A. David, R. Hennicker, K. Guldstrand Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from specifications to contracts in component-based design. In *Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering*, FASE'12, pages 43–58, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] I. Ben-Hafaiedh, S. Graf, and S. Quinton. Reasoning about safety and progress using contracts. In *Proceedings of the 12th international conference on Formal engineering methods and software engineering*, ICFEM'10, pages 436–451, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple Viewpoint Contract-Based Specification and Design. In *Proceedings of the Software Technology Concertation on Formal Methods for Components and Objects (FMCO'07)*, volume 5382. Springer, October 2007.

[6] A. Cimatti and S. Tonetta. A property-based proof system for contract-based design. In *38th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2012, Cesme, Izmir, Turkey, September 5-8, 2012*, pages 21–28. IEEE Computer Society, 2012.

[7] 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), 2011*, pages 1–6. IEEE, 2011.

[8] S. Graf and S. Quinton. Contracts for bip: Hierarchical interaction models for compositional verification. In *Proceedings of the 27th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems*, FORTE '07, pages 1–18, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] ISO 26262-10. *Road vehicles — Functional safety — Part 10: Guideline on ISO 26262*. International Organization for Standardization, 2011.

[10] O. Kath, R. Schreiner, and J. Favaro. Safety, Security, and Software Reuse: A Model-Based Approach. In *Proceedings of the 4th International Workshop on Software Reuse and Safety*, RESAFE '09, Washington, D.C., US, September 2009.

[11] E. Kesseler. Assessing cots software in a certifiable safety-critical domain. *Information Systems Journal*, 18(3):299–324, 2008.

[12] L. Latour, T. Wheeler, and B. Frakes. Descriptive and predictive aspects of the 3cs model: Seta1 working group summary. In *Proceedings of the first international symposium on Environments and tools for Ada*, SETA1, pages 9–17, New York, NY, USA, 1991. ACM.

[13] F. Redmill. The COTS Debate in Perspective. In *Proceedings of the 20th International Conference on Computer Safety, Reliability and Security*, SAFECOMP '01, pages 119–129, London, UK, 2001. Springer-Verlag.

[14] I. Sljivo, J. Carlson, B. Gallina, and H. Hansson. Fostering Reuse within Safety-critical Component-based Systems through Fine-grained Contracts. In *International Workshop on Critical Software Component Reusability and Certification across Domains*, June 2013. http://www.es.mdh.se/publications/2970-.