A Modular Safety Assurance Method considering Multi-Aspect Contracts during Cyber Physical System Design*

Peter Battram[1], Bernhard Kaiser[2], and Raphael Weber[1]

[1] OFFIS - Institute for Information Technology
Oldenburg, Germany
peter.battram|raphael.weber@offis.de
[2] Berner & Mattner Systemtechnik GmbH
Munich, Germany
bernhard.kaiser@berner-mattner.com

Abstract. Designing safety-critical cyber physical systems (CPS) was and remains a challenging task. CPS engineers are supposed to design solutions that are easy to modify, reusable, satisfy certification authorities, meet safety goals, separate between concerns, etc. With these partly contradicting demands it sometimes is even impossible to find a viable CPS design. The idea using contract-based design methods has been around for over two decades and enables automating the (re-)validation of the specification of CPS against the surrounding system or operational environment. In this work we extend the notion of contracts by component and interface contracts and give ideas on how to integrate them in a modular safety assurance approach. The explicit separation between these two types of contracts also better reflects the separation of concerns and reduces the overall modeling effort. We evaluate our approach with an automotive E-Drive case study.

Keywords: cyber physical systems, multi-aspect, contract based design, modular safety assurance

1 Introduction

When designing safety-critical cyber physical systems (CPS) we are dealing with a very difficult challenge. For a CPS, being safe means that the designer of that system must forsee everything that can lead to an unsafe situation. That being said, the designer not only has to install certain counter-acting mechanisms, increasing safety, he also must ensure that these mechanisms work correctly and that they actually increase the safety of that CPS. More precisely formulated, the designer of a safety-critical system has to guarantee its safety.

* This work was supported by the Federal Ministry for Education and Research (BMBF) under support code 01IS12005M

This fact is also mandatory when developing a system that has to be compliant to safety standards like the ISO26262 [8] or the DO178c [15]. These standards demand a safety case that evinces the absence of unreasonable risk for humans caused by the system under development. A vital part of these safety cases is a safety concept that depicts the architectural decisions concerning the fault detection as well as the fault mitigation. Besides an increasing use of components-off-the-shelf, the effective reaction to changes in the system design requires such safety concepts to be modelled in a modular way. Therefore the modular safety specification and analysis gain more and more importance.

Aside from the problem of ensuring the safety of embedded systems, there is another challenge in designing today's CPS: The ever-growing complexity, caused by the inclusion of yet another new feature, greatly impacting the rest of the system. The last few decades saw tremendous effort to cope with this complexity as well as safety issues.

In order to master this complexity, we need to break it down to the essentials. There are many formalisms to model different aspects of complex safety-critical systems. One single formalism in isolation only represents a fraction of the actual system under design. While some of these formalisms in isolation are well understood, the combination of them turns out to be another challenge. The combination of different modelling aspects allows for more realistic representations of the actual embedded system. However, that also makes it harder to analyze the systems to assure certain properties of interest.

Safety requirements are fundamental artifacts in the specification of safety-critical CPS, as they document how the hazards and failure of the system are mitigated. They are produced by the hazard and safety analyses and may therefore suffice in various viewpoints and levels of abstraction.

1.1 Cyber Physical Systems and Contracts

Embedded systems are small computing systems deeply embedded into their surroundings such that their existence is barely noticed. Since most embedded systems are small devices reacting on changes in their environment the notion cyber physical systems was coined to emphasize the necessity to take the environment into account during design [9]. CPS are also more networked due to the complex interdependencies of the environmental properties that are to be controlled by the CPS.

The intended control behavior and corresponding safety properties of CPS can be specified with various formalisms. In this regard, the use of contract-based modeling has proven to be beneficial [10]. To cope with the complexity of CPS, e. g. component-based design is applied during the development process. But when components or sub-systems are abstracted by their interfaces, such interfaces do not necessarily provide sufficient information for the correct and safe implementation of the other components [2]. Additional information about the interface in form of assumptions about the context conditions the component and the interface is needed. Contract-based design provides the concepts to handle these aspects.

Contracts represent a requirement in a structured way separating it into an assumption, stating the expected properties of the environment, and a guarantee, which describes the desired behavior of the component, provided the assumption is met by the environment. The introduced separation is the foundation for building a sound theory that allows the reasoning about the composition of systems in a formal way. Additionally, it reflects the very idea of CPS to explicitly consider the properties of the surrounding environment.

In this work we provide a method to distinguish between two contract types providing separation of concerns on the one hand and module decomposition and reusability on the other hand. This method may be applied in different domains allowing for the consideration of multiple aspects of CPS design and specification. In the context of safety-critical CPS design we describe how to integrate our proposed method into a modular safety assurance process.

1.2 Related Work

Using Contracts for better specification of critical systems and partly for improving safety analysis and verifications is currently a flourishing research domain. For example, Safety ADD (see [17]) helps to define and verify the safety contracts for software components in a graphical editor. The algorithm traverses all assumptions and guarantees to make sure they match.

A tool for checking the refinement between contracts called *OCRA* (Othello Contracts Refinement Analysis) was presented in [4]. It provides means for checking the contracts specified in a pattern based language called *Othello* which are later translated to a linear-time temporal logic for discrete and real-time constraints. The underlying engine allows to reason whether contract refinement is correct.

A full range of safety mechanisms, such as, definitions of faults and failures, fault containment, safety mechanisms, handling the degradation modes and safe states at multiple abstraction levels is proposed in [11, 12]. The interface between the safety view and the functional design is highlighted as well. Multiple safety patterns are provided in LTL [13] notation.

Temporal logic is also used in [3] for decomposing the system architecture with contracts. The framework automatically generates a set of proves.

1.3 Outline

The paper is organized as follows: In Section 2 we give a basic overview over contracts-based design. Section 3 details our approach to differ component and interface contracts and how to apply them in a modular safety assurance process. We evaluate this approach with an automotive E-Drive case-study in Section 4. Finally, we conclude our paper in Section 5 and give an outlook.

2 Fundamentals

In this section we will give a brief overview over contract-based design (informal and formal) and the pattern based RSL to specify contracts in a semantically sound way. In Subsection 2.3 we give a formal definition of the validation procedure that has to be performed in order to verify the correctness of contracts. Note that this procedure may differ in necessary effort or difficulty for different aspects.

2.1 Contract-based Design

Contracts are pairs consisting of an assumption (A) and a guarantee (G). The assumption specifies how the context of the component, i. e. the environment from the point of view of the component, should behave. Only if the assumption holds, then the component will behave as guaranteed. This kind of specification allows to replace components by more detailed ones, if they allow a less demanding environment, without re-validating the whole system. Thus, the system decomposition can be verified with respect to contracts without the knowledge of the concrete implementation. In this work, we use the RSL to specify the assumption and guarantee parts of contracts.

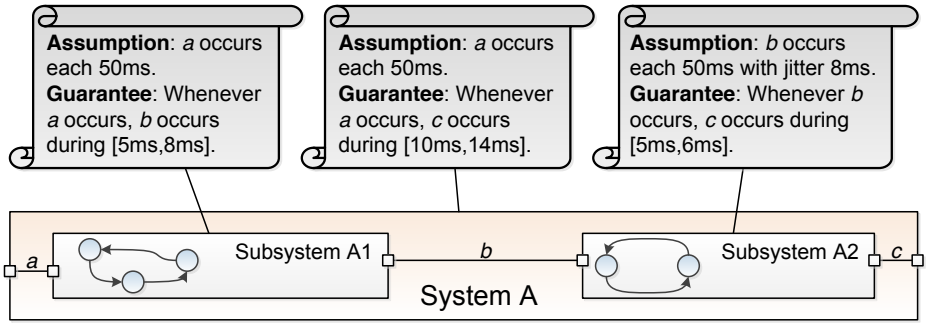

Fig. 1. Example for a contract specification.

One advantage of using contracts is that they can help to decrease the complexity of verifying the implementation against its specification. For example, consider the system in Fig. 1, to which a contract is assigned. The system contract states, that the input port 'a' is triggered each 50ms, and when it is triggered, the system has to respond by sending an event on port 'c' within a specific time interval. The system is decomposed into two subsystems each with one contract, and some internal behavior modeled by, e. g. state machines. Assume that the functionality on subsystem A2 depends on the output of subsystem A1. Further, assume that the subsystems would *not* be annotated with contracts. Thus, to validate the contract of the overall system A, the composed behavior of both

subsystems has to be computed, which generally leads to large state spaces. Using contracts for A1 and A2, we can omit the composition and validate the sub-contracts locally.

Formally, the semantics of a contract is defined as

$$[\![C]\!] := \ [\![A]\!]^{Cmpl} \ \cup \ [\![G]\!], \tag{1}$$

where $(X)^{Cmpl}$ defines the complement of a set X in some universe $\mathcal{U}$, and $[\![X]\!]$ is defined as the semantic interpretation of X. In our case, $[\![X]\!]$ is given in terms of sets of timed traces. A trace over an alphabet Σ is a sequence of events. Further, a time sequence τ is a monotonically increasing sequence of real values, such that for each $t \in \mathbb{R}$ there exist some $i \geq 1$ such that $\tau_i > t$. A timed trace is a sequence (ρ,τ) where ρ is a sequence of events and τ a time sequence. The set of all timed traces over Σ is denoted by $Tr(\Sigma)$.

The specification S of a component is given in terms of a set of contracts, i. e. $[\![S]\!] := \bigcap_{i=1}^{n} [\![C_i]\!]$. An implementation I of a component satisfies its specification S, if $[\![I]\!] \subseteq [\![S]\!]$ holds. The refinement relation between two contracts C and C' is defined as follows:

$$C' \text{ refines } C, \text{ if } [\![A]\!] \subseteq [\![A']\!] \text{ and } [\![G']\!] \subseteq [\![G]\!], \tag{2}$$

2.2 Requirements Specification Language

Natural language requirements are often accompanied by ambiguity, incompleteness or inconsistency; the reason for this resides in the very nature of a spoken language. These unintended byproducts may not be detected until late phases of the development process and therefore cause the realization to not fulfill the intended goals of the specification or cause incompatibilities to other system parts. A solution to this problem would be the use of formal languages to specify the requirements. But these are often hard to understand and therefore difficult to use. An alternative that bridges this gap is the pattern-based Requirements Specification Language (RSL) [14, 1]. This formal language provides a fixed semantic that enables an automated validation or verification but is still well readable compared to natural language.

Consisting of static text elements and attributes which are filled in by a requirements engineer, patterns have a well-defined semantic so that a consistent interpretation of the system specification between all stakeholders can be ensured. To cope with the needs of the different aspects of a design, various sets of patterns have been defined which are partly described in the following.

2.3 Validation of Contracts

When a component is decomposed into a set of sub-components, we have to check whether the overall contract $C = (A, G)$ (which also will be called global contract) and all subcontracts $C_i = (A_i, G_i)$ (also called local contracts) for

$i \in \{1, ..., n\}$ are consistent. We have to check the following virtual integration condition:

$$\begin{aligned} i)\ A \wedge G_1 \wedge ... \wedge G_n \ &\Rightarrow\ A_1 \wedge ... \wedge A_n \\ ii)\ A \wedge G_1 \wedge ... \wedge G_n \ &\Rightarrow\ G. \end{aligned} \tag{3}$$

An in-depth discussion about virtual integration can be found in [5]. In [5] contracts were extended by so called *weak* assumptions. Weak assumptions are used to describe a set of possible environments in which the component guarantees different behaviors. This separation is only methodological, and does not affect the semantics of the definition of the original contracts: Let $C = (A_s, A_{w_1}, ..., A_{w_n}, G_1, ..., G_n)$ be a contract consisting of a strong assumption A_s, a set of weak assumptions A_{w_i}, and a set of corresponding guarantees G_i for $i \in \mathbb{N}$. Semantically, we map C to a standard contract of the form $C' = (A_s, G)$, where $G = (G'_1 \wedge ... \wedge G'_n)$ $G'_i = (A_{w_i} \Rightarrow G_i)$. For timing and a special subset of safety properties the validation has already been done [6, 7]. For other aspects this may, however, be more difficult which is one reason to also consider a methodological approach. We propose the methodological separation between component and interface contracts described in the following section.

3 Multi-Aspect Modular Safety Assurance

This section gives informal definitions of component and interface contracts. Furthermore, we describe how to transform them into each other and how to integrate them into a modular safety assurance process. The term multi-aspect thereby refers to the possibility to specify required properties concerning multiple views on the CPS (e. g. signal quality, timing, or reliability).

3.1 Component Contracts

In contract-based design, contracts are used to systematically derive and document requirements and to conduct contract validation on the systems architecture. Such validation can serve as evidence for the correct allocation of functions which in turn constitute the overall function of a system, i. e. the allocation of the functions to a given CPS architecture and environment is correct and provably valid.

Component contracts are contracts between any system of consideration and its operational context. Therefore, they are a key artifact in order to make requirements allocated to a component modular and reusable, and to make formerly tacit assumptions explicit. In the context of safety, contract-based specifications aim at reasoning about the fault containment properties of components in a modular, reusable way [12].

Fig. 2 depicts an excerpt of the architecture of an E-Drive system (more details in Section 4) and two exemplary contracts linked to the Microcontroller component. The functional contract e. g. states that under the assumption that

the components power supply is sufficient and the temperature of the components environment is suitable, the component will start the calculation of the pWM signal within 5 milliseconds after the acceleration pedal has been pressed. A safety contract, focusing on failure propagation, states that the outputs of the Microcontroller component (shutOffSignal and pWM) will not fail, if the components input (phaseCurrentValue) or the component itself do not fail. These contracts are represented in a formal way using the predefined RSL (see Section 2.2).

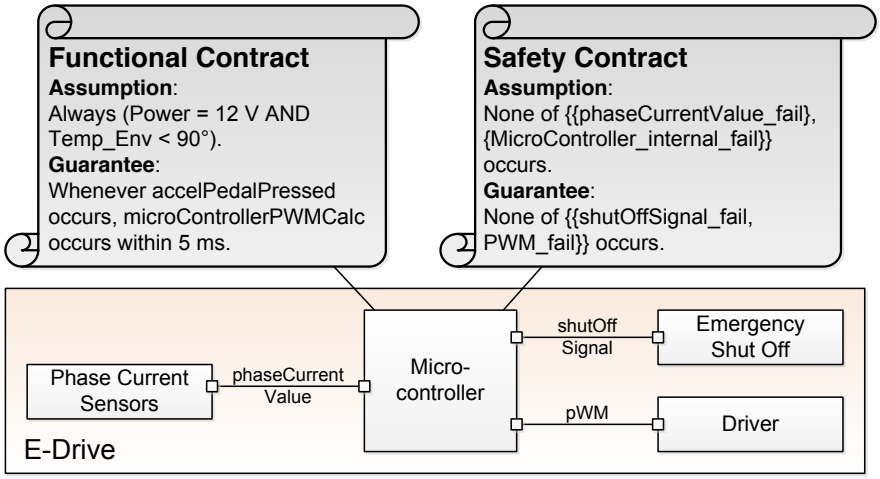

Fig. 2. Functional and safety examples of component contracts.

3.2 Interface Contracts

The relationship between assumptions of specific component to guarantees of its neighbors (and vice versa) can be defined by an *interface contract*. The difference to component contracts is mainly that assumptions refer to signal qualities provided at input ports of a component and promises refer to signal qualities provided at the output ports of the neighbor components. Interface contracts and component contracts can be transformed into each other canonically.

An interface contract is a combination or a pair consisting of an assumption and a guarantee attached to the interfaces. Each contract is negotiated between two respective neighbor components. Assumptions and guarantees may refer to signal characteristics at a given port, as seen by an omniscient external observer, e. g. the accuracy of a signal with respect to the real physical quantity, the integrity of a signal, the delay of a port signal with respect to an event in the outside world. It is obvious that some reusable component, taken out of its environment, shall not refer to such context knowledge that is not directly related to

quantities observable at its direct interfaces - therefore the modularization to its full extent is only achieved by later transforming interface contracts into component contracts. Yet, interface contracts help verifying the architecture when decomposing the system, because it can easily be checked along the signal flow links whether or not the assumptions at the input of some component are fulfilled by the guarantees at the linked output of its predecessor component. Unfulfilled contracts, i. e. assumptions that are not satisfied by corresponding guarantees, can be detected and highlighted automatically, which has been demonstrated by a prototype tool in [16]. Fig. 3 illustrates the concept of the interface contract.

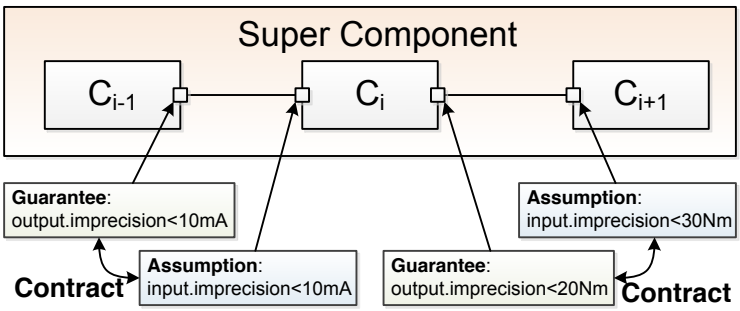

Fig. 3. Signal quality examples of interface contracts.

3.3 Using Component and Interface Contracts

In order to use both types of contracts during the design of CPS, we propose the following steps:

1. Let S be some CPS to be developed with its external interfaces, some requirements (e. g. some signal to be provided with a required accuracy and safety integrity, or an event to be triggered with a specified maximum delay counted upon some internal condition, e. g. some failure condition) and some assumptions about its environment. The job at hand is to decompose the requirements and to allocate them onto the designated (sub-)components of the system, thereby verifying that the refinement is correct.
2. The requirements engineer specifies the informal (textual) requirements formally, or at least in a semi-formal notation, which allows to specify the target properties unambiguously. A parametrized template language (like the RSL) may be a semi-formal way to do so, temporal logics like LTL [13] may be formal notations that are suitable. Thereby, the requirements to the system are stated as guarantees that the system must fulfill, while it can rely on assumptions, which are the formalized assumptions about the technical usage environment. These assumptions and guarantees constitute a *component contract* with the system to be developed being the subject component.

3. The architect decomposes the system into components, specifying their purposes and their interfaces at which (intermediate) signals are provided. Each signal gets some meaning assigned and all signals are listed in a signal dictionary. The architect states his ideas about how to decompose the requirements into sub-requirement and how to assign the budgets on quality properties (such as timing, accuracy, safety integrity) among the components. He does so by tentatively specifying assertions that are meant to hold at different "probing points" (signals) in the architecture.
4. The assertions can be interpreted as *interface contracts* as follows: The output of the component upstream in the signal flow must guarantee the properties at the given point, whereas the input of the component downstream in the signal flow may rely on the property as being granted, i. e. this is an assumption. Interface contracts are made between neighbor components. Where an input comes directly from an input port of the surrounding system or an output direct feeds an output port of the surrounding system, the stated assertions are propagated one level up in the system hierarchy (or matched with the systems assumptions and guarantees, when we are on the top level of the hierarchy).
5. The interface contracts are not yet suitable for specifying the sub-components as standalone components with the potential of being designed from some external supplier who does not know about the rest of the system. This is due to the fact that the assertions were stated from the architects point of view (who is in the position of an omniscient observer, which is obviously not true for the component supplier). For instance, the manufacturer of a car airbag controller cannot guarantee that the airbag inflates at max 100 microseconds after the crash occurs, simply because he has no knowledge about the crash occurrence. All he knows is the signal at the input port of his own component, provided for example by some sensor subsystem, and therefore all requirements must refer to this behavior at the ports. The solution to this challenge is to transfer the interface contracts into component contracts, which is a canonical step (i. e. can be fully automated).
6. The resulting component contracts only consist of propositions that refer to the observable behavior at the outer interface of the component of interest. Therefore, the guarantees that the component will have to fulfill can be transferred into requirements to the component (if the supplier is unfamiliar with the specific formal notation, a pattern generator could be used to create boilerplate language in English or any other natural language out of it). The other way around, the assumptions granted to the component can also be exported to the supplier and give him some certainty about properties of the usage environment he can rely on when designing the component.

3.4 Integration into a Modular Safety Assurance Method

For the usage of component and interface contracts in an integrated safety process, we propose a three-step approach:

– During conventional system design, functional requirements are analyzed and an initial system architecture is derived from these requirements. The architectural view allows for specifying system contracts, which cover in this step the decomposition and verification of the nominal behavior of the system. The system architecture is hence enriched by functional contracts.
– The next step in the system engineering process is the safety analysis. The purpose of this analysis is to find failures, i. e. component behavior observable at the component interfaces that violates the specification of the nominal behavior and that might lead, if not handled appropriately, to some hazard or accident. The contracts defined in the previous step formalize the nominal behavior of the system, which aids in finding deviations. This means, that the failure mode definitions needed to aggregate different types of safety analysis, can be interpreted as contract violations: If, for instance, an output interface of some sensor component provides the value for a measured temperature and an interface contract promises for this output an accuracy of $\pm 2°C$, then every value actually provided at the output which deviates from the true value by more than $2°C$ is obviously considered as failure of the sensor component (to be precise, corresponding to failure modes "too high" or "too low" respectively, when applying the failure mode keywords known from the HAZOP analysis technique). The existing functional contracts together with the architecture design and the results of safety analysis serve as a basis to define safety contracts and thereby derive the functional safety requirements.
– The final step in the safety process is to create the technical safety concept. The systems architectural view is now updated to include additional safety components and mechanisms in addition to the system design, which may become necessary to detect and mitigate the identified component failures, or to achieve the required safety integrity level. To these new components, but also to the existing ones, additional contracts can be assigned, this time relating to safety properties. Examples for those properties can be promises for safety-relevant properties that hold with a defined safety integrity level or a minimum guaranteed probability, or the guarantee that despite certain failures at a component input, there will be no failures at its output (e. g. Assumption: Input can be any well-formed bus message; Guarantee: Output is an integer software variable either corresponding to the value transmitted by the bus message or the accompanying Boolean valid flag being set to FALSE.) Transforming such kind of safety-related interface contracts into component contracts leads to the formulation of the related technical safety requirements. As a result, the system architecture showing the nominal behavior design is enriched towards the final system safety architecture.

4 E-Drive Case Study

We evaluated our approach by applying the component and interface contract modeling to an automotive E-Drive case study that has been introduced in [16]. An E-Drive is a system responsible for actuating a vehicle. It consists of an

electric machine, a controller and power electronics. An excerpt of the E-Drive system architecture is depicted in Fig. 4. This case study qualifies for the evaluation of our approach, because of its safety-critical properties and its modular character, meaning that an E-Drive could for example replace a conventional combustion engine in the future.

Besides the architecture of the E-Drive, Fig. 4 also shows interface contracts (located on the left and right hand side above the architecture) as well as a component contract for the Microcontroller component (located in the center above the architecture).

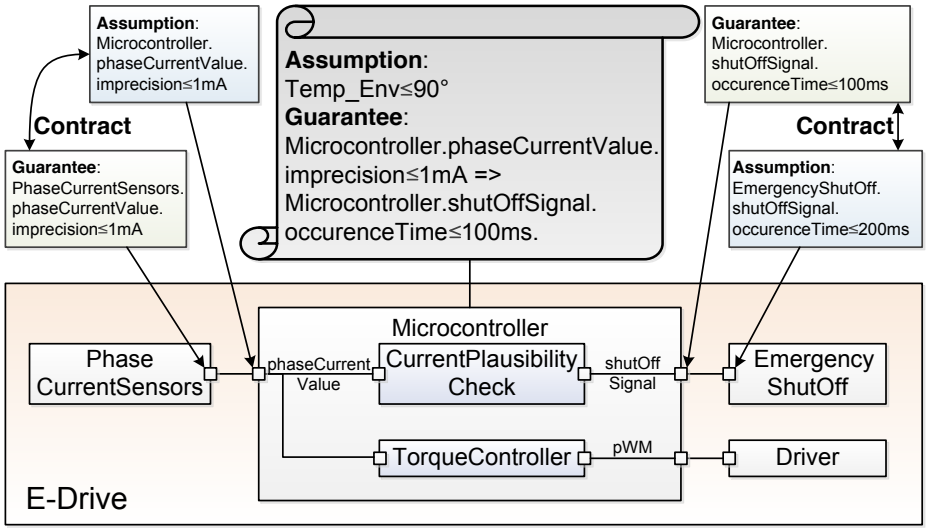

Fig. 4. Interface and component contracts linked to the corresponding architecture entity of an E-Drive system.

For this case study the interface contracts have been modeled along the signal flow from the Phase Current Sensors via the Microcontroller to the Emergency Shut-Off component. They specifically state which condition the signal property does fulfill or has to fulfill like for example the guarantee of the Phase Current Sensors output. This guarantee states that the imprecision of the phase current value is not allowed to exceed one milliampere. Following the signal flow, the next component to process the signal is the Microcontroller. This component itself has requirements regarding the input signals, which are stated as an assumption. In this case the component also demands the phase current value's imprecisions to not exceed one milliampere. This pair of guarantee and assumption constitutes an interface contract between the two components regarding the phase current value's imprecisions.

The same procedure was performed for the shut-off signal sent from the Microcontroller to the Emergency Shut-Off component (right hand side of Fig. 4).

For this case, analyses like a compatibility check can be performed. Applying this kind of analysis to our case study showed that the compatibility between the components along this signal flow was given.

In the next process step the interface contracts were used to derive the component contracts for the components of the E-Drive system. The result, the component contract for the Microcontroller is depicted in the top center of Fig. 4. As can be seen the guarantee was derived by the previously defined assumption and guarantee of the Microcontroller, stating that if the left hand side of the implication is fulfilled, the right hand side can be guaranteed. The assumption in the example was added to state that the Microcontroller can just guarantee its correct functionality for an environmental temperature not exceeding 90°Celsius.

The evaluation of the approach showed that besides the fact that forcing the engineer to capture requirements in a more structured and ultimately a more formal way decreases the ambiguity of the specification. Interface as well as component contracts enable analyses to validate the specification against the system context or the operational environment. Additionally, these approaches support a modular safety assessment by encapsulating the assumptions and guarantees relevant for the component that shall be changed or replaced.

5 Conclusion

Coping with specifications for CPS and validating them is a regular task that needs to be performed during system design. This paper described a new method to apply contract-based design in the context of a modular safety assurance process. We therefore distinguished between component and interface contracts to better support the system architect in validating the correctness of his design (including the specification and the structural composition). We evaluated our approach using an automotive E-Drive case-study which also showed an improvement in the usability and applicability of contracts in general. Future work will be conducted in the field of multi-aspect validation and verification of inseparable properties of CPS. There are still a lot of design concerns that have not yet been formalized or addressed at all. Optimizing one or more properties of interest (e. g. reduce costs, maximize safety, ...) is another topic which we will look into.

References

1. Baumgart, A., Böde, E., Büker, M., Damm, W., Ehmen, G., Gezgin, T., Henkler, S., Hungar, H., Josko, B., Oertel, M., Peikenkamp, T., Reinkemeier, P., Stierand, I., Weber, R.: Architecture modeling. Tech. rep., OFFIS (March 2011), http://ses.informatik.uni-oldenburg.de/download/bib/paper/OFFIS-TR2011_ArchitectureModeling.pdf
2. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T., Larsen, K.: Contracts for systems design. In: Inria Research (2012)

3. Bozzano, M., Cimatti, A., Katoen, J.P., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended aadl models. Comput. J. 54(5), 754–775 (May 2011), http://dx.doi.org/10.1093/comjnl/bxq024
4. Cimatti, A., Dorigatti, M., Tonetta, S.: Ocra: A tool for checking the refinement of temporal contracts. In: ASE IEEE. pp. 702–705 (2013)
5. Damm, W., Hungar, H., Josko, B., Peikenkamp, T., Stierand, I.: Using contract-based component specifications for virtual integration testing and architecture design. In: DATE Exhibition. pp. 1–6 (2011)
6. Gezgin, T., Weber, R., Girod, M.: A refinement checking technique for contract–based architecture designs. In: Fourth International Workshop on Model Based Architecting and Construction of Embedded Systems (10 2011)
7. Gezgin, T., Weber, R., Oertel, M.: Multi-aspect virtual integration approach for real-time and safety properties. In: International Workshop on Design and Implementation of Formal Tools and Systems (DIFTS14). IEEE (Oct 2014)
8. ISO: Road Vehicles - Functional Safety. International Standard Organization (November 2011), iSO 26262
9. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach. Lee and Seshia, 1 edn. (2010), http://chess.eecs.berkeley.edu/pubs/794.html
10. Meyer, B.: Applying "design by contract". Computer 25(10), 40–51 (1992)
11. Oertel, M., Kacimi, O., Böde, E.: Proving compliance of implementation models to safety specifications. In: Bondavalli, A., Ceccarelli, A., Ortmeier, F. (eds.) Computer Safety, Reliability, and Security, Lecture Notes in Computer Science, vol. 8696, pp. 97–107. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-10557-4_13
12. Oertel, M., Mahdi, A., Böde, E., Rettberg, A.: Contract-based safety: Specification and application guidelines. In: Proceedings of the 1st International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC 2014) (2014)
13. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 1977., 18th Annual Symposium on. pp. 46–57 (Oct 1977)
14. Reinkemeier, P., Stierand, I., Rehkop, P., Henkler, S.: A pattern-based requirement specification language: Mapping automotive specific timing requirements. In: Reussner, R., Pretschner, A., Jähnichen, S. (eds.) Software Engineering 2011 Workshopband. pp. 99–108. Gesellschaft für Informatik e.V. (GI) (2011)
15. RTCA: DO-178C: Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics (RTCA) (2011)
16. Sonski, S.: Contract-based modeling of component properties for safety-critical systems. Master's thesis, Hochschule Darmstadt University of Applied Sciences (2013)
17. Warg, F., Vedder, B., Skoglund, M., Soderberg, A.: Safetyadd: A tool for safety-contract based design. In: International Symposium on Software Reliability Engineering (ISSRE) 2014 Workshops. to appear (2014)