Making Implicit Safety Requirements Explicit
An AUTOSAR Safety Case

Thomas Arts, Michele Dorigatti, and Stefano Tonetta

QuviQ and FBK
thomas.arts@quviq.com, mdorigatti@fbk.eu, tonettas@fbk.eu

Abstract. Safety standards demand stringent requirements on embedded systems used in safety-critical applications such as automotive, railways, and aerospace. In the automotive domain, the AUTOSAR software architecture provides some mechanisms to fulfill the ISO26262 requirements. The verification of these mechanisms is a challenging problem and it is not always clear in which context the safety requirements are supposed to be met.
In this paper, we report on a case study developed in the SafeCer project, where we combined contract-based design and model-based testing. A contract-based approach has been used to formalize the safety requirements to detect communication failures. The formal specification shows under which assumptions the AUTOSAR protection mechanism fulfills these requirements. A model-based testing approach has been used to test the software implementing such protection mechanism. The model used for testing has been model checked against the contract specification ensuring that the system-level safety requirements are met.

Keywords: Formal Methods, Contract-Based Design, Testing, AUTOSAR

1 Introduction

The AUTOSAR standard [1] is a detailed architectural description of software components for the automotive industry. Modern cars contain many different computing units (ECUs) that are connected via several networks. The basic software running on all these units is written by different vendors, but it should be compatible. The AUTOSAR standard is created to "enforce" compatibility. The C software implementing AUTOSAR components is highly configurable and has in general a practically infinite number of possible internal states. Ensuring compatibility is in practice performed by testing and, in the setting of our work, by using Model-Based Testing (MBT): a formal model of the software is instantiated by the same configuration used for the C software. This model instance is then used to automatically generate thousands of random tests. Failing tests indicate a conformance deviation with the model.

Sensors and actuators often handle data for safety-critical applications such as an airbag or a parking brake. They can be located at different physical ECUs so that safety-critical data are communicated over one of the networks. For this reason, the ISO26262 standard [2] prescribes to implement measures to detect communication faults such as loss or corruption of messages. The AUTOSAR standard caters for a common set of these fault models by offering a solution called *End-to-End (E2E) Protection* [3] and is specified as a library with functions to protect a data item and to check it at the other end of the communication. In short, it adds a counter and identifier to the data, computes a checksum and sends the data and checksum over the bus instead of the raw data. At the other end, the checksum is used to see if the data got corrupted and if not, the data is compared to an earlier value to see if it can be trusted. By addressing a number of fault models once and for all with a library, the AUTOSAR software developers know what they can use when they are faced with specific safety requirements. However, what is achieved by using such libraries is not always clear: by just using the protection mechanism of AUTOSAR, the software developer is not guaranteed to obtain a fault-tolerant system, and it is critical to define the context in which the system is safe and level of tolerance that is guaranteed.

In this paper, we formalized the guarantees of the E2E Protection library components (with the simplest profile $P01$) and the assumptions on the communication failures into the input language of OCRA [4], a tool for Contract-Based Design (CBD) of embedded systems. Given that we have formalized the protection mechanism, we can combine this with system-level safety requirements of a system architecture. In particular, we modeled an airbag that inflates when the sensors send a message that we are in collision, whereas at the same time, the airbag never inflates if no such message arrives from the sensor. The contract-based refinement of the airbag using the AUTOSAR protection mechanism was proved correct with the OCRA support to contract-refinement checking. We finally verified with model checking techniques the contract specification on existing test models of the AUTOSAR library for QuickCheck [5,6], a tool for MBT of software. With QuickCheck, we can automatically generate arbitrary tests for the communication stacks. These tests are clearly enhanced by the formal proof that the test models correctly refine the safety requirements.

The contributions of the paper are manifold. First, we showed how to bridge the gap between system requirements and guarantees on the E2E Protection library. Second, we derived the guarantees on the communication stacks for a concrete example. Third, we showed how the properties of the AUTOSAR library used in the contract-based derivation can be verified on the models used for code testing. Finally, we provided a methodology to integrate CBD and MBT tools.

The paper is structured as follows: in Sec. 2, we give an overview of the AUTOSAR E2E Protection mechanism and the related ISO26262 safety requirements; in Sec. 3, we describe the background tools and techniques, which are CBD, OCRA, MBT, and QuickCheck; in Sec. 4, we present the main part of the paper, including the airbag example, the methodology integrating CBD and MBT, the formalization of the E2E Protection library, the contract-based derivation, and the verification of the code implementation; in Sec. 5 we give an overview of the related work, and finally in Sec. 6, we draw some conclusions.

2 AUTOSAR E2E Protection

2.1 AUTOSAR

AUTOSAR is a software standard for the automotive industry providing the specification of the basic software components, such as several protocol stacks, memory management, communication routing, etc. The AUTOSAR platform offers a variety of components to provide functionality, for example a component for E2E data protection that encodes and decodes a message in a standard way so that corruption or message loss can be detected. Each software component is specified by both a very specific programmer API as well as a behavioral description. The diversity of demands from different hardware platforms and car models are catered for by specifying the configurability of the software. Each component can be configured in many different ways. The actual software that is eventually put in a vehicle is partly generated from a confuguration file and partly statically provided.

The flexibility in configurations makes it difficult to test AUTOSAR software, since a test case typically consists of both a sequence of API calls with expected return values and side-effects as well as a configuration for which the code under test has to be generated. Manually creating test cases is tedious and error prone. Therefore, Quviq has developed a method to generate test cases from a formal model of one or more AUTOSAR components. Given a configuration, test cases, valid and meaningful for that configuration, are generated and executed.

Clearly, the correctness of software applications built on top of the AUTOSAR basic software components requires correctly implemented components and a correct use and configuration of these components. The latter is an engineering challenge due to the difficult in understanding from the set of requirements on software what guarantees the software components offers in a specific configuration and architecture. For example, the E2E Protection library is recommended, among others, as an implementation measure against *message loss during transmission* [3]. Clearly, message may get lost for several reasons and the engineering task is to handle that. Using the E2E protection library may seem the right thing to do, but the engineering challenge is to understand the context of this guarantee. If communication is safety-critical and many messages in a row are lost in few milliseconds, can one then still use the designed solution? What configuration of the E2E Protection library should be used in order to protect against this scenario? These questions are based on the use of the software component by an application outside the AUTOSAR basic software. Without knowledge about this application, such questions are difficult or impossible to answer. This paper shows how one can make the assumption of the application explicit in order to formulate precise guarantees on the E2E Protection library.

2.2 ISO26262 Requirements and E2E Protection

In order to implement effective measures against communication loss the ISO26262 standards prescribe to take into account a series of possible communication faults: loss of peer to peer communication; unintended message repetition due to the same message being unintentionally sent again; message loss during transmission; insertion of messages due to receiver unintentionally receiving an additional message, which is interpreted to have correct source and destination addresses; re-sequencing due to the order of the data being changed during transmission, i.e. the data is not received in the same order as in which it was been sent; message corruption due to one or more data bits in the message being changed during transmission; message delay due to the message being received correctly, but not in time; blocking access to data bus due to a faulty node not adhering to the expected patterns of use and making excessive demands for service, thereby reducing its availability to other nodes, e.g. while wrongly waiting for non existing data; and constant transmission of messages by a faulty node, thereby compromising the operation of the entire bus.

The faults described above are represented in AUTOSAR by the following fault models:

- repetition: a message is received more than once.
- deletion: a message or parts of it have been removed from the communication stream.
- insertion: an additional message or parts of it have been inserted into the communication stream.
- incorrect sequence: the messages of a communication stream are received in an incorrect order.
- corruption: the corruption data of a message or parts of it occurred.
- delay: a message is received too late.
- addressing faults: a message is sent to the wrong destination.

The protection measure provided by the AUTOSAR E2E library consists of using:

1. a counter modulo N ($N = 15$ in Profile 1) which is increased by one at every sent message;
2. a checksum provided by the AUTOSAR CRC library;
3. data ID to verify the identity of each transmitted safety-related data element.

In particular, the repetition, deletion, insertion, and incorrect sequence are addressed by the counter, corruption by the CRC checksum, addressing faults by the data ID. In addition to this, the real-time properties of AUTOSAR in combination with periodically sending messages enable detection of not receiving new data on the bus. Timeouts are therefore represented by either no new data available or by receiving new data with the same counter as the previously received valid data.

3 Background Techniques

3.1 OCRA and CBD

In this paper, we adopt the CBD framework supported by the OCRA tool [4]. In particular, we use a finite-state discrete-time model of the system. Component interfaces are described with Boolean or bounded integer data ports and with events, which are instantaneous triggers of changes (from the formal point of view, they are Boolean labels on the state transitions). An execution trace of the component is therefore a sequence of states, which are assignments to the port variables. The transition from a state to another one can be labeled with an event. Assertions on the execution traces are specified by means of linear-time temporal logic. In particular, we use LTL [7] with past operators and predicates over current and next variables to represent state changes, as informally described in Table 1.

Syntax	Informal Description	Example	Example description
$P(V)$	atomic predicate over the port variables V	$m \geq 2$	m is greater or equal to 2
$P(V, V')$	atomic predicate over the current port variables V and their next value after a transition	$c' = c + 1$	c is increased by 1
$\neg\phi$	Boolean negation	$\neg f$	f is false
$\phi_1\{\wedge, \vee, \rightarrow\}\phi_2$	binary Boolean operators	$m = n \rightarrow c' = c$	if $m = n$ then c is not changed
$\mathbf{G}\ \phi$	ϕ holds in every future state	$\mathbf{G}\ (f \rightarrow \mathbf{G}\ f)$	when f is true, it remains true forever
$\mathbf{F}\ \phi$	ϕ holds in some future state	$\mathbf{G}\ (f \rightarrow \mathbf{F}\ e)$	f is always followed by e
$\mathbf{O}\ \phi$	ϕ holds in some past state	$\mathbf{G}\ (f \rightarrow \mathbf{O}\ e)$	f is always preceded by e
$\mathbf{X}\ \phi$	ϕ holds in the next state	$\mathbf{G}\ (f \rightarrow \mathbf{X}\ e)$	f is always immediately followed by e
$\mathbf{Y}\ \phi$	ϕ holds in the previous state (so, we are not in the initial state)	$\mathbf{G}\ (f \rightarrow \mathbf{Y}\ e)$	f is always immediately preceded by e

Table 1. Overview of the temporal logic used to specify the contracts.

The OCRA input language is a component-based description of the system architecture where every component is associated with one or more contracts. Each contract consists of an assumption and a guarantee specified as temporal logical formulas. The assumption represents a requirement on the environment of the component. The guarantee represents a requirements for the component implementation to be satisfied when the assumption holds.

When a component S is decomposed into subcomponents, the contract refinement ensures that the guarantee of S is not weakened by the contracts of the subcomponents while its assumption is not strengthened. This is checked independently from the actual implementation of the components and is verified by means of a set of proof obligations in LTL, which are discharged with model checking techniques [8,9].

OCRA allows to associate to a component a behavioral model representing its implementation. The language used for the behavioral model is SMV, the input language of the NuSMV model checker [10]. OCRA checks if the SMV model is a correct implementation of the specified component simply calling NuSMV to verify if the SMV model satisfies the implication $A \rightarrow G$ for every contract $\langle A, G\rangle$ of the component.

3.2 QuickCheck and MBT

QuickCheck is a testing technique and tool originating from the functional programming community [11] where one expresses properties of the software under test in a functional language. QuickCheck then computes test cases that should be satisfied if this property holds for the software. The original stateless and logic properties have been extended by Quviq into conformance properties, where the functional language is used to implement a kind of reference implementation from which test cases are automatically generated [5].

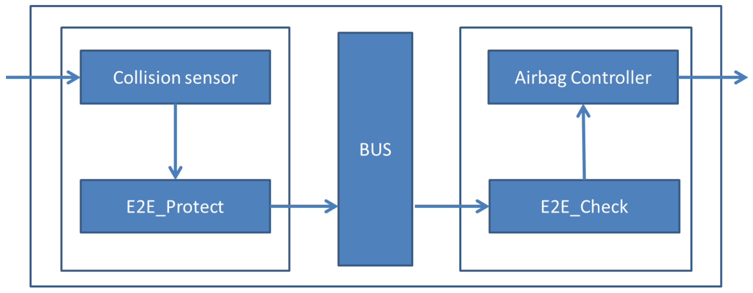

Fig. 1. A simple architecture of an airbag system communicating with a collision sensor.

Thus, rather than focusing on individual test cases to encapsulate the behavior of a system, this behavior is specified by properties, expressed in a logical form. The system is then tested by checking whether it has the required properties for randomly generated data, which may be inputs to functions, sequences of API calls, or other representations of test cases. In this way, large software systems, such as the AUTOSAR basic software or Radio Base Station Software, have successfully been tested.

Some versions of QuickCheck can systematically inflate the complete state space of a model. However, the models in general have an infinite or at least practically infinite state space. Therefore, the Quviq QuickCheck version is optimized for random walks through the state space, not for guaranteeing that all paths have been walked through.

4 Making Implicit Safety Requirements Explicit

4.1 The Airbag Example

As an example of usage of the E2E Protection mechanism, we consider a simplified version of an airbag system. An informal picture of the system architecture is shown in Figure 1. In real systems the airbag software is different, but we use this example to show how we handle systems similar to real industrial ones. The formalization of the real E2E library is faithful and the method we present for deriving the constraints of using the E2E library is generally applicable to any similar system.

The Airbag System communicates with a Collision Sensor to know when to trigger the airbag. The communication runs on a bus which may fail. In this example, the Collision Sensor is split into two components: the actual sensor that senses the collision and the E2E Protection which enriches the message with the E2E Protection mechanism before sending it over the bus. Similarly, the Airbag System is split into the actual airbag controller and the software component that checks the message to detect communication failures.

After making a design like this, one wants to reason about it in the presence of the given fault models. In theory, when the world is perfect, as soon as the sensor detects a collision, a message is sent and the airbag inflates. In practice, the sensor may be broken, the bus may experience one of the faults described above and the controller may be disfunctional. In other words, there is no guarantee that this is going to work.

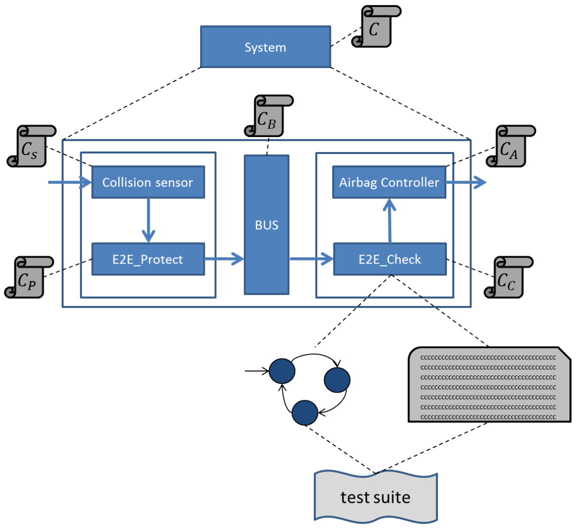

Fig. 2. Integration of CBD and MBT

Since it is important to know whether the sensor is working, it will continuously send messages to the controller. In that way, a broken sensor is detected. We do not want to report sensor failure if only one message is lost and therefore it is important that the E2E Protection detects message deletion.

It is also important to only inflate the airbag if the car is in a real collision. We therefore want to detect data corruption at the same time as that we do not just send a one bit zero and one message, but use over engineering to send a byte with two different bit patterns to distinguish normal and collision situation.

4.2 Integration of CBD and MBT

We propose a methodology that integrates Contract-Based Design (CBD) and Model-Based Testing (MBT). CBD is used at the system level in order to ensure a correct derivation of the properties of the components. In particular, in case of safety-related requirements, it forces to explicitly state the assumptions on the failures that are necessary in order to guarantee the requirements. MBT is used at the code level to ensure the compliance of the code with the model.

The proposed methodology integrates CBD and MBT as depicted in Figure 2: the system architecture is enriched with a contract refinement, which is proved correct with formal methods; the software components are implemented with a state machine and with a concrete program; the state machine is proved to satisfy the component contracts with model checking and is used to generate a test suite for the program.

The airbag system has been formalized with OCRA. Each component is enriched with contracts to specify the assumption and guarantee of the component. The contract of the Bus component include the possibility to have failures (either a message corruption or loss). The system component explicitly specifies the assumptions on such failures in order to ensure its guarantee. The contract refinement has been proved correct. The state machine of the E2E_Check component has been taken from the AUTOSAR specification and modeled with QuickCheck. This model has been automatically translated into an SMV model and verified with OCRA to check that the state machine correctly implements the E2E_Checks contract. Finally, the QuichCheck state machine has been used to generate tests for a real C implementation of the E2E_Check.

4.3 Formal Model of the E2E Protection Mechanism

We modeled in OCRA an abstraction of the E2E Protection mechanism described in Sec. 2.2. The OCRA components of the E2E have a counter that is incremented by the E2E_Protect component at every transmitted message. The E2E_Check component checks if the received counter is exactly the last received valid counter incremented by one or if there was a communication fault. The CRC checksum is abstracted with a Boolean ValidCRC variable that represents the return value of the CRC Library. Thus, if ValidCRC is false, the message was corrupted. The data IDs instead are not modeled since we have only one destination of messages. The availability of new data on the bus is represented by a Boolean variable.

The interface of the E2E_Check component has therefore the following ports:

– NewDataAvailable: a Boolean input port that represents if there is new data available on the transmission medium;
– ReceivedCounter: a 4-bit input port that represents the value of the counter on a new received message;
– ValidCRC: a Boolean input port that represents if the received message is valid;
– MaxDeltaCounter: a parameter used to define acceptable number of lost messages;
– Status: an output port that defines the status of the received message, which can be: NONEWDATA, WRONG_CRC, INITIAL, REPEATED, OK, OKSOMELOST, WRONG_SEQUENCE;
– LastValidCounter: an output port that stores the counter of the last received message.

The main guarantees of the component are formalized by the following formulas (a complete version of the specification can be found in
`https://es.fbk.eu/people/tonetta/tests/safecomp14`):

– $\mathbf{G}\ (((\mathit{NewDataAvailable} \wedge \mathit{ValidCRC} \wedge 1 \leq \mathit{DeltaCounter} \leq \mathit{MaxDeltaCounter}) \wedge \mathbf{Y}\ \mathbf{O}\ (\mathit{NewDataAvailable} \wedge \mathit{ValidCRC})) \rightarrow (\mathbf{X}\ \mathit{status\_ok}(\mathit{Status})))$; in the OCRA syntax this is written as:

```
always (((NewDataAvailable and ValidCRC and
          1<=DeltaCounter and DeltaCounter<=MaxDeltaCounter)
 and previously in the past (NewDataAvailable and ValidCRC))
  implies (then status_ok(Status)));
```

this means that whenever the component receives a new valid message with a valid counter and previously another valid message was received, then the status is set to OK or OKSOMELOST depending on the value of the counter (this is encoded by the macro “status_ok”).

– $\mathbf{G}$ $(\textit{status\_ok}(\textit{Status}) \rightarrow (\mathbf{Y}\ ((\textit{NewDataAvailable} \wedge \textit{ValidCRC} \wedge 1 \leq \textit{DeltaCounter} \leq \textit{MaxDeltaCounter}) \wedge \mathbf{Y}\ \mathbf{O}\ (\textit{NewDataAvailable} \wedge \textit{ValidCRC}))))$; in the OCRA syntax this is written as:

```
always (status_ok(Status) implies
 (previously ((NewDataAvailable and ValidCRC and
  DeltaCounter>=1 and DeltaCounter<=MaxDeltaCounter) and
  previously in the past (NewDataAvailable and ValidCRC))));
```

this means that the status is set of OK or OKSOMELOST only if the component just received a valid message with a valid counter and in the past another valid message was already received.

4.4 Contract-Based Refinement of the Airbag Requirements

The top-level requirements of our airbag example are that every collision must be followed by the inflation of the airbag and vice versa, the airbag inflates only after a collision. This is formalized in the LTL formulas $\mathbf{G}\ (collision \rightarrow \mathbf{F}\ inflate)$ and $\mathbf{G}\ (inflate \rightarrow \mathbf{O}\ collision)$. This is accomplished by the five components of the system as follows: the sensor senses the collision and generates a message; the message is enriched with the counter by the E2E_Protect component; the message is transmitted from the sender to receiver by the bus; on the receiver side, the message is checked by the E2E_Check component and then passed to the airbag controller that makes the airbag inflate. Each of these functions is formalized by an LTL formula.

The bus contracts include the fault models of message corruption and deletion. In particular, the bus component receives in input two events, fault_deletion and fault_corruption. Whenever fault_deletion happens, the NewDataAvailable output is set to false. Whenever fault_corruption happens, the ValidCRC output is set to false.

If we do not specify any assumption on the system inputs, the contract refinement fails, confirming that some assumptions are implicit. There are three assumptions that we added in order to obtain a correct refinement:

1. $\mathbf{G}\ (fault \rightarrow \mathbf{X}\ \neg fault)$: we assume that there cannot be two consecutive faults, either corruption or deletion of a message;
2. $MaxDeltaCounter \geq 2$: we assume that at least one lost message is acceptable and the E2E_Check component will consider the new message valid even if the counter has been increased by 2;
3. $\mathbf{G}\ (collision \rightarrow \mathbf{G}\ collision)$: whenever there is collision, this will be permanent, so that collision messages will be continuously sent to airbag controller.

If we remove one of these assumption, the OCRA tool gives us a counterexample showing an execution trace that violates the top-level contract. Note however that these assumptions are not guaranteed to be the weakest conditions.

4.5 Formal Model and Verification of the E2E Check Implementation

The E2E Check function is informally presented as a state machine description in the AUTOSAR specification. In our case, we took that informal presentation as a start for a

QuickCheck model. This QuickCheck model has a formal semantics and has been used to generate test cases. On turn, these were used to verify a C implementation of the E2E Check function.

The QuickCheck model was also translated into an SMV model. This translation is to a high extent automated. Erlang [12] is a functional language, with single assignment. In a few steps, we transform the program by partially evaluating all occurrences of local variables. The result is a program that only has the input variables present. For technical reasons we rewrite if-statements into case statements and make all case statements into Boolean choices by nesting them. The resulting transformation is a completely equivalent Erlang program. The QuickCheck model uses records to store data in. The input record is processed and a similar output record is returned. For each field in the output record, we backtrace in the source code what conditions should hold to change its value into the value that it can get in the return record. This then results in the SMV next state function for each input variable.

Finally, the SMV model has been model checked with OCRA to verify that it is a correct implementation of the E2E_Check component, in other words that it satisfies the contracts of the component. We can conclude that the Erlang model satisfies such contracts. Moreover, we can be confident that the C implementation satisfies such contracts, since the state space is finite and by collecting the model state space while running a few hundred thousand tests (which takes about an hour), we can show to have covered all model states during testing. We also performed other sanity checks provided by OCRA such as the checking the receptiveness of implementation [4].

4.6 Discussion

In this section, we discuss the results that we achieve giving also a critical overview of the pros and cons of the approach. We start highlighting the importance of the feedback on the requirements that we achieved. Protection mechanisms such as the one implemented in AUTOSAR are fundamental to detect failures and to fulfill the safety requirements prescribed by the safety standards. However, it should be clear that they do not allow to achieve a complete tolerance to failures and that the system requirements are fulfilled only under some assumptions. The contract-based approach allows to formalize a context in which this holds.

These conclusions should also suggest to improve the safety standards accordingly. In fact, requiring that the system is robust to any communication failures in absolute terms is not realizable and pose on the designer an arbitrary choice on the assumptions under which the requirements should be fulfilled. It would be desirable instead that such assumptions are defined by the standard itself.

Another important aspect of the case study is that the E2E_Check component that has been used and verified can be reused in another case study as is. Using it in another system architecture will require to perform a different contract refinement of the top-level contract, but the contracts of the E2E components should not be changed.

The main weakness of the case study is that is a small example and many details are abstracted. This has the advantage that the example is suitable to explain and understand the approach. We focused on the protection mechanism disregarding a more realistic representation of the airbag controller or of the bus. The E2E_Check component is faithfully represented and we also consider a real C implementation of it.

In a contract-based approach the analysis is compositional so that the verification of the contract refinement would remain the same even if we added a full description of the airbag controller. In future work, we plan to model a more complex system involving more components communicating on the same bus. We used only two types of faults, but these were sufficient to elicit the modeled protection mechanisms. Finally, we used a discrete model of time, but this seems acceptable due to the real-time properties of AUTOSAR and the periodic sending of messages via an external scheduling algorithm.

5 Related Work

Contracts have been previously used in the context of safety-critical systems and in particular in relation to standards such as ISO26262. Many of these works focused on processes and methodologies [13,14]. In [15] and [16], example of contracts are elaborated and it is shown that they can be used to reason about safety. The relation between contracts and ISO26262 is made stronger in [17] where contracts are used to structure the safety requirements of the standard.

Our paper differs from previous ones in a number of distinguishing features: first, we take a critical look at the implicit assumptions on safety requirements and use contracts to make such assumption explicit; second, we focus on the usage of concrete tools to support contracts and testing; to the best of our knowledge, it is the first contribution that concretely integrates a tool that supports contracts with a tool for testing in a way that the same model is used by both tools.

Model based testing in combination with fault-injection has been presented in [18] on exactly the same airbag example, but without CBD. Fault injection is a technique to demonstrate that even in the presence of faults, the system behaves in a safe way. This is highly valuable from a testing point of view, but like testing in general, it shows weaknesses rather than that it provides guarantees. In best case one obtains knowledge on what guarantees are not fulfilled.

6 Conclusions and Future Work

In this work, we experimented with a new integration of CBD supported by the OCRA tool and MBT supported by QuickCheck. The goal of the integration is to exploit the contract-based refinement to make explicit the assumptions of the model used for testing. The approach has been applied to the AUTOSAR measures to protect safety-critical communication from communication failures as specified by the ISO26262 standard. The assumption of the AUTOSAR protection mechanism has been formalized with OCRA on an airbag example.

In the future, we want to link the proposed approach to the development and certification process defined in the SafeCer project; we may also integrate both tools in a common design environment, develop further the case study introducing real-time aspects and generalizing the assumption using parametrized properties.

Acknowledgments We like to thank Hans Svensson from Quviq for writing the Erlang QuickCheck model for the end-to-end protection library. The research leading to these results has received funding from the ARTEMIS JU under grant agreement n^o 295373, from National funding, and from the Swedish research foundation Vinnova for its support of the Acsäpt project (ref. 2012-00943).

References

1. AUTOSAR: Software architecture specification www.autosar.org.
2. ISO 26262: Road vehicles Functional safety (2011)
3. AUTOSAR. In: Specification of SW-C End-to-End Communication ProtectionLibrary. AUTOSAR consortium (2008-2013)
4. Cimatti, A., Dorigatti, M., Tonetta, S.: OCRA: A tool for checking the refinement of temporal contracts. In: ASE. (2013) 702–705
5. Arts, T., Hughes, J., Johansson, J., Wiger, U.: Testing telecoms software with Quviq QuickCheck. In: ACM SIGPLAN Workshop on Erlang. (2006)
6. Svenningsson, R., Johansson, R., Arts, T., Norell, U.: Formal methods based acceptance testing for AUTOSAR exchangeability. SAE Int. Journal of Passenger Cars Electronic and Electrical Systems **5**(1) (May 2012) 209–213
7. Pnueli, A.: The Temporal Logic of Programs. In: FOCS. (1977) 46–57
8. Cimatti, A., Tonetta, S.: A Property-Based Proof System for Contract-Based Design. In: EUROMICRO-SEAA. (2012) 21–28
9. Cimatti, A., Tonetta, S.: Contracts-refinement proof system for component-based embedded systems . Sci. Comput. Program. To appear.
10. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: CAV. (2002) 359–364
11. Claessen, K., Hughes, J.: QuickCheck: a lightweight tool for random testing of Haskell programs. In: ACM SIGPLAN ICFP. (2000) 268–279
12. Armstrong, J.: A history of erlang. In: HOPL. (2007) 1–26
13. Blanquart, J.P., Armengaud, E., Baufreton, P., Bourrouilh, Q., Griessnig, G., Krammer, M., Laurent, O., Machrouh, J., Peikenkamp, T., Schindler, C., Wien, T.: Towards Cross-Domains Model-Based Safety Process, Methods and Tools for Critical Embedded Systems: The CESAR Approach. In: SAFECOMP. (2011) 57–70
14. Baumgart, A., Reinkemeier, P., Rettberg, A., Stierand, I., Thaden, E., Weber, R.: A Model-Based Design Methodology with Contracts to Enhance the Development Process of Safety-Critical Systems. In: SEUS. (2010) 59–70
15. Damm, W., Josko, B., Peikenkamp, T.: Contract Based ISO CD 26262 Safety Analysis. In: Safety-Critical Systems. In: SAE. (2009)
16. Damm, W., Hungar, H., Josko, B., Peikenkamp, T., Stierand, I.: Using contract-based component specifications for virtual integration testing and architecture design. In: DATE. (2011) 1023–1028
17. Westman, J., Nyberg, M., Törngren, M.: Structuring Safety Requirements in ISO 26262 Using Contract Theory. In: SAFECOMP. (2013) 166–177
18. Vedder, B., Arts, T., Vinter, J., Jonsson, M.: Combining fault-injection with property-based testing. In: Proc. of Int. Workshop on Engineering Simulations for Cyber-Physical Systems. ES4CPS '14, New York, NY, USA, ACM (2014) 1–8