A Reference Example on the Specification of Safety Requirements using ISO 26262

Jonas Westman[1] and Mattias Nyberg[2]

[1] Royal Institute of Technology (KTH)
[2] Scania

Abstract. ISO 26262 - *"Road vehicles-Functional Safety"* is a standard for the automotive industry, administered in an attempt to prevent potential accidents due to systematic and random failures in the Electrical/Electronic-system. In general, requirements in industry is often of poor quality and considering the strong emphasis on requirements management in ISO 26262, we believe that there is a strong need for guidance and experience-sharing on the specification of requirements in practice.
We therefore present a reference example on the application of ISO 26262 in practice, where we perform a breakdown of a Safety Goal of an industrial system down to Software Safety Requirements on the C-code implementation. As a basis for structuring and formulating the requirements, we use the concepts of contracts and port variables.

1 Introduction

ISO 26262 - *"Road vehicles-Functional Safety"* is a standard for the automotive industry, administered in an attempt to prevent potential accidents due to systematic and random failures in the Electrical/Electronic-system. For a particular system, ISO 26262 advocates a complete set of safety requirements covering both its top-level functionality as well as requirements on individual SW components. In general, requirements in industry is often of poor quality [1] and considering the strong emphasis on requirements management in ISO 26262, we believe that there is a strong need for guidance and experience-sharing on the specification of requirements in practice. In this paper, we therefore share our results and experiences from specifying safety requirements, as proposed by ISO 26262, for an industrial system.

The main contribution is a reference example on the application of ISO 26262 in practice, considering safety requirements from all requirement levels: from a Safety Goal down to requirements on SW components. In SW, we provide a wide range of safety requirements for real industrially written C-code covering both application- and basic SW. As a basis for structuring and formulating the requirements, we use the principles of contracts [2], and in particular, the notion of port variables [3].

In a literature search for references on the application of ISO 26262 and IEC 61508 with respect to requirements engineering, we find few other in-depth examples on how to apply the standard in practice. We encountered papers discussing

general approaches, such as Model-Based Development [4] or Contract-Based Design [5], to meet the demands on requirement management in ISO 26262 and IEC 61508. However, none of these papers went into any depth in their examples and would therefore provide limited support for an everyday engineer. Other papers, such as [6] and [7], go into more depth and provide experiences and insights of practical issues encountered when applying the standard. The scope in [6] is, however, limited to solely one of the development phases and the focus in [7] is on architectural design, rather than on requirements engineering. In contrast, in the present paper, we cover a wider scope as we consider a breakdown from a Safety Goal of an industrial system down to Software Safety Requirements on a C-code implementation.

In Sec. 2, we briefly describe the industrial case - the Fuel Level Display (FLD) system. In Sec. 3, we present the breakdown of the Safety Goal of the FLD-system down to requirements on individual SW components. In Sec. 4, we conclude our experiences collected while working with the FLD-system.

2 Industrial Case - The Fuel Level Display-system

The FLD-system is a safety-critical system present on all Scania vehicles and we will here consider the actual C-code flashed onto the produced vehicles. The basic functionality of the FLD-system is to provide an estimate of the fuel volume in the fuel tank to the driver. The functionality is distributed across three ECU (Electric Control Unit)-systems, i.e. an ECU with sensors and actuators, in the Electronic/Electrical (E/E)-system of the truck: Engine Management System (EMS), Instrument Cluster (ICL), and Coordinator (COO). The ECU-systems also interact with the fuel tank and the parking brake system that is outside of the E/E-system of the truck.

There are several architectural variants of FLDS, e.g. variability in fuel tanks, types of sensors, etc. Due to space restrictions, only one type of system variant is considered here. The considered variant is shown in Fig. 1.

COO estimates the fuel volume `actualFuelVolume[%]` in the tank by a Kalman filter. The input signals to COO are: the position of a floater in the fuel tank `sensedFuelLevel[%]`, as sensed by the fuel sensor; and the CAN signal `FuelRate[l/h]` in the message `FuelEconomy-E`, transmitted on CAN from EMS. The CAN signal `FuelRate[l/h]` is an estimate of the current fuel consumption `injectedFuel[l/h]`. The estimated fuel volume is transmitted on CAN as the CAN-signal `FuelLevel[%]` in the CAN-message `DashDisplay`. The CAN message is received by ICL where a fuel gauge `indicatedFuelVolume[%]` in the display provides the information to the driver.

A development according to ISO 26262 revolves around an item, which is in [8] described as *"a system that implements a function at a vehicle level"*. For the analysis in this paper, COO is chosen to be the item. In Fig. 1, we can see the item boundary along with the SW architecture of the ECU of the COO, the ECU-HW, the fuel sensor, and its environment. We can also see the data flow and control flow between the different elements in the SW architecture.

The SW architecture of COO is structured as followed: the APPLication (APPL) SW consists of SW components that implement the high level functionality of the ECU; the MIDDleware (MIDD) SW contains the SW components in charge of controlling the I/O, e.g. sensors and actuators, connected to the ECU and the encoding/decoding of CAN-messages; and the Basic Input/Output System (BIOS) SW contains the SW components that manages the low-level interaction with the executing platform, i.e. the ECU HW.

3 Specification and Break-down of Safety Requirements

In this section, we specify the Safety Goal (SG) and the safety requirements for the FLD-system. Similar to our work in [9] and to the concepts of contracts as described in [3], we consider a requirement to be a relation on variables and also consider each requirement on an element to be applicable under explicitly identified assumptions on the environment to the element. Different hierarchical levels of requirements as described in ISO 26262, i.e. Safety Goals (SGs), Functional Safety Requirements (FSRs), Technical Safety Requirements (TSRs), and Hardware and Software Safety Requirements (HSRs/SSRs) are mapped to the type of variable refered to in the requirements. That is, if a requirement refers to variables that model properties at a vehicle level or those shared between ECU-systems, it is considered to be an FSR. If both variables with HW and SW properties are refered to, it is considered to be a TSR and if only variables with e.g. only SW properties are referenced, it is considered to be a SSR, and so on.

Concerning the format of the requirements/assumptions specifications, we refer to variables using the format '`name[unit]`', such as e.g. `actualFuelVolume[%]`. Furthermore, we write '`func`: ...' to indicate if a requirement is applicable only when the function `func` is called. To pair a requirement with assumptions, we make a reference to one, or a set of assumptions, by writing '($A^{\#}$)' besides the requirement. Similarly, for assumptions specifications, we write, '($SR^{\#}$)' or '(Element)' beside the assumption to either: refer to the requirements that implement the assumption; or to an element where a subset of its requirements implement the assumption. We use the term '*corresponds to*' when two variables are approximately equal, e.g. when they differ in type or deviate due to small delays.

As a limitation due to space restrictions, we only consider those requirements that are applicable when the ignition is on. In SW that corresponds to only including requirements applicable during run-time. We make a global assumption that every main function, identified as `xyz_20ms` in Fig. 1, is run every 20 ms and that the data-flow is exclusive to what is shown in Fig. 1, i.e. no other element can read or write to a variable other than what is shown.

3.1 Safety Requirements on COO and Application SW

In Table 1, the safety requirements allocated to COO and the SW component FUEL are presented, along with the assumptions on their respective environments. The Safety Goal SG^1 can be interpreted as: *the FLD-system does not*

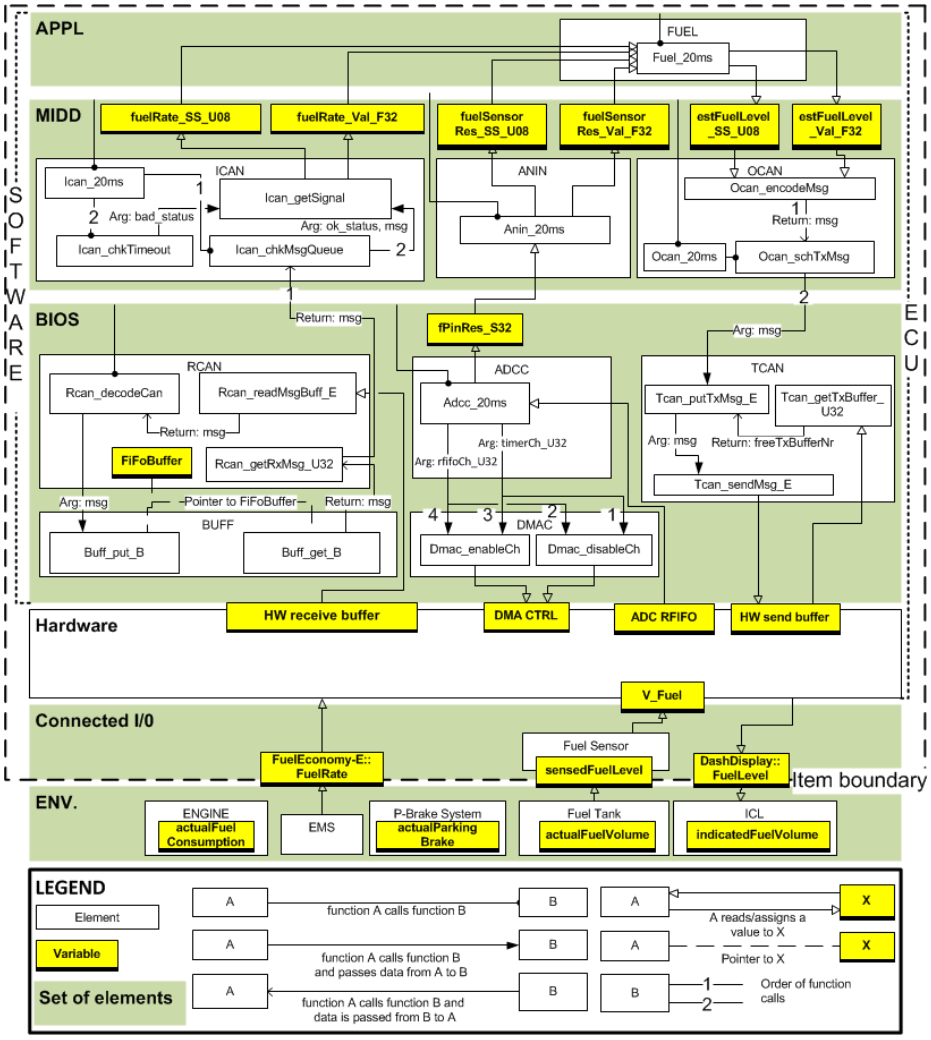

Fig. 1. System and SW architecture of the FLD-system. White blocks represent elements such as software components, C-functions, the HW of COO, the fuel sensor, ECU-systems, and the fuel tank. Yellow boxes with a bolded bottom line represent system properties, modelled as variables. Green boxes, without a black outline, represent a set of elements. Different relations, e.g. data and control flow, between the different elements and variables are explained in the legend at the bottom of the figure. The item boundary is indicated by the bolded dashed line. We consider the HW as a black box where the variables model its interfaces. *NOTE:* We modify the architecture and the naming to protect the integrity of the implementation. The original complexity and intent of the architecture is, however, sustained.

provide misguiding information to the driver while the parking brake is not applied. We hence consider the state when the vehicle is parked as a safe state of the vehicle. The assumptions A_{COO}^{1-5} on the environment to COO can be interpreted as the following, respectively: *the vehicle is not refueled while the parking brake is not applied*; *the fuel sensor is correctly installed*; *EMS provides an accurate estimate of the fuel consumption*; and *ICL displays the estimated fuel volume, provided by COO, to the driver*. COO hence imposes, by the use of the assumptions, the necessary requirements on its environment in order to be able to implement the Safety Goal. That is, if e.g. the fuel sensor is installed incorrectly, COO cannot guarantee that the Safety Goal is implemented.

The FSR FSR_{FUEL}^{1} allocated to FUEL, derived from the SG SG^1, states that COO estimates the fuel volume in the tank and transmits it on CAN. Assumptions A_{FUEL}^{1-3} are equal to A_{COO}^{1-3}. Assumption A_{FUEL}^{4} states that the input signal `fuelSensorRes_Val_F32[%]` corresponds to the position of the floater `sensedFuelLevel[%]`, or the status signal `fuelSensorRes_SS_U08[Enum]` has value ERR. Assumption A_{FUEL}^{5-6} states that the input signal `fuelRate_Val_F32-[litres/h]` to FUEL corresponds to the CAN signal `FuelRate[l/h]` in the message `FuelEconomy-E` in case it does not have the value $0xFE$ (error). In case `FuelRate[l/h]` has the value $0xFE$ (error) or if the signal was expected sooner, then `fuelRate_SS_U08[Enum]` has the value ERR. Assumption A_{FUEL}^{7-8} states that output signal `estFuelLevel_Val_F32[%]` to FUEL is transmitted on CAN as the signal `FuelLevel[%]` in the CAN-message `DashDisplay` if `est-FuelLevel_SS_U08[Enum]` does not have the value ERR. In case `estFuelLev-el_SS_U08[Enum]` has the value ERR, then `FuelLevel[%]` has the value $0xFE$.

3.2 Safety Requirements on Middleware SW

In Table 2, the safety requirements allocated to the SW components ANIN, ICAN and OCAN are presented, along with the assumptions on their respective environments. The SW components in the MIDD SW provide the APPL SW with SW-signals that correspond to readings from sensors and CAN-signals and also encodes SW-signals from the APPL SW into CAN-messages.

The TSR TSR_{ANIN}^{1} allocated to ANIN is equal to the assumption A_{FUEL}^{4} in Table 1, which means that TSR_{ANIN}^{1} implements A_{FUEL}^{4}. The assumptions A_{ANIN}^{1} and A_{ANIN}^{2} on the environment to ANIN state that the SW-signal `fPinRes_s32[mV]` corresponds to a voltage value `V_Fuel[mV]` at one of the input pins of the ECU and also that the fuel sensor either provides the intended values or values that are out-of-range. The SSRs SSR_{ANIN}^{1-2}, derived from TSR_{ANIN}^{1}, state that `fuelSensorRes_Val_F32[%]` corresponds to a filtered value of `fPin-Res_s32[mV]`, or `fuelSensorRes_SS_U08[Enum]` has the value ERR.

Note that the TSRs TSR_{ICAN}^{1-2} allocated to ICAN are equal to, and hence also implement, the assumptions A_{FUEL}^{5-6} in Table 1. The assumption A_{ICAN}^{1} on the environment to ICAN states that if the oldest message `msg` in the queue `FiFoBuffer` has the Parameter Group Number (PGN) number $OxFEF2$, then `msg` is equal to `FuelEconomy-E`. As stated in the assumption A_{ICAN}^{2} on the

Table 1. Safety requirements allocated to COO and APPL SW components

SG^1	IF `actualParkingBrake[Bool]` is not applied (false), THEN `indicatedFuelVolume[%]`, shown by the fuel gauge, is less than `actualFuelVolume[%]`; OR `indicatedFuelVolume[%]` has a value below 0%. (A^{1-5}_{COO})
A^1_{COO}	IF `actualParkingBrake[Bool]` is not applied (false), THEN `injectedFuel[l/h]` is equal to the derivative of `actualFuelVolume[%]`. (Driver)
A^2_{COO}	The position of the floater `sensedFuelLevel[%]`, sensed by the fuel sensor, does not deviate more than ±10% from `actualFuelVolume[%]` in the fuel tank. (Fuel tank)
A^3_{COO}	The CAN signal `FuelRate[litres/h]` in the CAN message `FuelEconomy-E` does not deviate more than ±1% from `injectedFuel[l/h]`; OR `FuelRate[litres/h]` has the value $0xFE$ (error); OR `FuelEconomy-E` is delayed more than 0.3s. (EMS)
A^4_{COO}	IF `actualParkingBrake[Bool]` is not applied (false) AND the CAN signal `FuelLevel[%]` has the value $0xFE$ (error) THEN `indicatedFuelVolume[%]`, shown by the fuel gauge, has a value below 0%. (ICL)
A^5_{COO}	IF `actualParkingBrake[Bool]` is not applied (false) AND the CAN signal `FuelLevel[%]` does not have the value $0xFE$ (error) THEN `indicatedFuelVolume[%]` corresponds to `FuelLevel[%]`. (ICL)
FSR^1_{FUEL}	IF `actualParkingBrake[Bool]` is not applied (false), THEN the CAN signal `FuelLevel[%]` in the CAN message `DashDisplay` is less than `actualFuelVolume[%]`; OR `FuelLevel[%]` has the value $0xFE$ (error). (A^{1-8}_{FUEL})
A^1_{FUEL}	See A^1_{COO}
A^2_{FUEL}	See A^2_{COO}
A^3_{FUEL}	See A^3_{COO}
A^4_{FUEL}	See TSR^1_{ANIN} in Table 2 (TSR^1_{ANIN})
A^5_{FUEL}	See TSR^1_{ICAN} in Table 2 (TSR^1_{ICAN})
A^6_{FUEL}	See TSR^2_{ICAN} in Table 2 (TSR^2_{ICAN})
A^7_{FUEL}	See TSR^1_{OCAN} in Table 2 (TSR^1_{OCAN})
A^8_{FUEL}	See TSR^2_{OCAN} in Table 2 (TSR^2_{OCAN})

Table 2. Safety requirements allocated to MIDD SW components

TSR^1_{ANIN}	`fuelSensorRes_Val_F32[%]` corresponds to the floater position `sensedFuelLevel[%]`, sensed by the fuel sensor; OR `fuelSensorRes_SS_U08[Enum]` has the value ERR. (A^{1-2}_{ANIN})
SSR^1_{ANIN}	On `Anin_20ms()`: IF 200 $\leq$`FuelPin_s32[mV]`$\geq$ 3000[3], THEN `fuelSensorRes_Val_F32[%]` is set to the interpolated value of `fPinRes_s32[mV]` according to table X[3].
SSR^2_{ANIN}	On `Anin_20ms()`: IF 3000 $<$`fPinRes_s32[mV]`[3] OR `fPinRes_s32[mV]`$<$ 200[3], THEN `fuelSensorRes_SS_U08[Enum]` has the value ERR.
A^1_{ANIN}	See TSR^1_{ADCC} in Table 3 (TSR^1_{ADCC})
A^2_{ANIN}	The fuel sensor converts the floater position `sensedFuelLevel[%]` into a voltage value `V_Fuel[mV]` according to table Y[3]; OR 3000 $<$`V_Fuel[mV]` OR `V_Fuel[mV]`$<$ 200[3] (Fuel sensor)
TSR^1_{ICAN}	IF the CAN signal `FuelRate[l/h]` in the CAN-message `FuelEconomy-E` does not have the value $0xFE$ (error) AND has been received within 0.3s[3] THEN `fuelRate_Val_F32[l/h]` corresponds to `FuelRate[l/h]`. (A^{1-2}_{ICAN})
TSR^2_{ICAN}	IF the CAN signal `FuelRate[l/h]` has the value $0xFE$ (error) OR is delayed more than 0.3s[3] THEN `fuelRate_SS_U08[Enum]` is set to the value ERR. (A^{1-2}_{ICAN})
A^1_{ICAN}	See SSR^1_{RCAN} in Table 3 (SSR^1_{RCAN})
A^2_{ICAN}	See TSR^1_{RCAN} in Table 3 (TSR^1_{RCAN})
TSR^1_{OCAN}	IF `estFuelLevel_SS_U08[Enum]` has the value ERR, THEN the CAN signal `FuelLevel[%]` in the CAN-message `DashDisplay` is transmitted with the value $0xFE$ (error) every 1s[3]. (A^1_{OCAN})
TSR^2_{OCAN}	IF `estfuelLevel_SS_U08[Enum]` does not have the value ERR, THEN the CAN signal `FuelLevel[%]` is transmitted with a value that corresponds to `estFuelLevel_Val_F32[%]` every 1s[3]. (A^1_{OCAN})
A^1_{OCAN}	See TSR^1_{TCAN} in Table 3 (TSR^1_{TCAN})

[3]For integrity reasons, we either modify the values or choose not provide this information.

Table 3. Safety requirements allocated to BIOS SW components

TSR^1_{ADCC}	`fPinRes_s32[mV]` corresponds to the voltage value `V_Fuel[mV]`. (A^{1-3}_{ADCC})
A^1_{ADCC}	IF the DMA channels `timerCh_U32` AND `rfifoCh_U32` are enabled for approx. 20ms, THEN a RAW value of `V_Fuel[mV]` is available in `ADC RFIFO`. (ADC (HW))
A^2_{ADCC}	See SSR^1_{DMAC} (SSR^1_{DMAC})
A^3_{ADCC}	See SSR^2_{DMAC} (SSR^2_{DMAC})
TSR^1_{RCAN}	IF `DashDisplay` is received within 20ms THEN it is available in `FiFoBuffer`. (A^{1-3}_{RCAN})
SSR^1_{RCAN}	On `Rcan_getRxMsg_U32()`: IF the oldest message in `FiFoBuffer` has PGN $0xFEF2$, THEN `msg=FuelEconomy` is returned. (A^{2-3}_{RCAN})
A^1_{RCAN}	On `Rcan_decodeCan`: a new CAN message is available in `HW receive buffer` (CAN-Controller (HW))
A^2_{RCAN}	See SSR^1_{BUFF} (SSR^1_{BUFF})
A^3_{RCAN}	See SSR^2_{BUFF} (SSR^2_{BUFF})
TSR^1_{TCAN}	On `Tcan_putTxMsg_E(msg)`: IF `msg` has PGN $0xFEFC$, THEN `DashDisplay=msg` is transmitted onto CAN. (A^1_{TCAN})
A^1_{TCAN}	The messages put in `HW send buffer` is transmitted onto CAN. (CAN-Controller (HW))
SSR^1_{DMAC}	On `Dmac_enableCh(ch_U32)`: the DMA channel `ch_U32`[4] is enabled.
SSR^2_{DMAC}	On `Dmac_disableCh(ch_U32)`: the DMA channel `ch_U32`[4] is disabled.
SSR^1_{BUFF}	On `Buff_put_B(msg)`: Adds msg to `FiFoBuffer`.
SSR^2_{BUFF}	On `Buff_get_B()`: returns the oldest message `msg` from `FiFoBuffer`.

[4] `ch_U32`=(`timerCh_U32` OR `rfifoCh_U32`)

environment to ICAN, if `FuelEconomy-E` has arrived within 20ms, i.e. since the latest execution tick, then it has been placed in the queue `FiFoBuffer`.

Note that the TSRs TSR^{1-2}_{OCAN} allocated to OCAN are equal to, and hence also implement, the assumptions A^{7-8}_{FUEL} in Table 1. The assumption A^1_{OCAN} on the environment to OCAN states that if the function `Tcan_putTxMsg_E` is called with an argument *msg* that has a PGN number $OxFEF2$, then the CAN-message `DashDisplay` is transmitted on CAN.

3.3 Safety Requirements on Basic Input/Output System SW

In Table 3, the safety requirements allocated to the SW components TCAN, RCAN, ADCC, DMAC and BUFF are presented, along with their assumptions on their respective environments. The SW components in the BIOS-layer provide the MIDD-layer with SW-signals that correspond to voltage values at the input pins for analogue sensors and manages the HW/SW interaction.

Notably, the TSR TSR^1_{ADCC} allocated to ADCC is equal to, and hence also implement, the assumption A^1_{ANIN} in Table 2. As stated in A^1_{ADCC}, a RAW AD value is available if the ADC is allowed to sample for 20ms, i.e. an execution tick. ADCC controls the ADC by enabling/disabling Direct Memory Access (DMA)-channels by calling functions in DMAC. Hence, as stated in the assumptions A^{2-3}_{ADCC} on DMAC, the DMA-channels are enabled/disabled by calling the functions `Dmac_disableCh(ch_U32)` and `Dmac_enableCh(ch_U32)`, respectively, by passing the appropriate DMA channels with the argument `ch_U32`. The assumptions A^{2-3}_{ADCC} are implemented by the SSRs SSR^{1-2}_{DMAC} allocated to DMAC.

The TSR TSR^1_{RCAN} allocated to RCAN implement A^2_{ICAN} since TSR^1_{RCAN} and A^2_{ICAN} are equal. In the same manner, the SSR SSR^1_{RCAN} implement

A^1_{ICAN}. As stated in A^{2-3}_{RCAN}, the put and get functions in BUFF manages the queue `FiFoBuffer`. Note that the assumptions A^{2-3}_{RCAN} are implemented by SSR^{1-2}_{BUFF} allocated to BUFF. Furthermore, as stated in assumption A^1_{RCAN}, RCAN is notified whenever a new message has arrived in `HW receive buffer`.

The TSR TSR^1_{TCAN} allocated to TCAN implement A^1_{OCAN} since TSR^1_{TCAN} and A^1_{OCAN} are equal. The assumption A^1_{TCAN} on the environment to TCAN states that the messages placed in `HW send buffer` are transmitted on CAN.

4 Conclusions

We have, in Sec. 3, presented a reference example on the specification of safety requirements in ISO 26262 in practice: from a safety goal down to software safety requirements. The reference example is intended as a source of guidance and experience-sharing on the application of requirements engineering within ISO 26262. As a basis for structuring and formulating the requirements, we have used the concept of assume-guarantee contracts, and the concept of port variables. Thus, the example provided in the paper is not only a reference example on ISO 26262 and requirements engineering in general, but also serves as an example of using assume-guarantee contracts and explicit port references in the context of requirements specification. Although further practical validation is needed, the presented example indicates that the concepts are fully possible to use in an industrial context.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. 1st edn. Cambridge University Press, New York, NY, USA (2010)
2. Meyer, B.: Applying "Design by Contract". IEEE Computer **25** (1992) 40–51
3. Benveniste, A. et al.: Multiple Viewpoint Contract-Based Specification and Design. In Boer, F. S. et al., eds.: Formal Methods for Components and Object. Springer-Verlag, Berlin, Heidelberg (2008) 200–225
4. Boulanger, J.-L., and V. Q. Dao: Experiences from a Model-Based Methodology for Embedded Electronic Software in Automobile. In: Inf. and Comm. Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd Int. Conf. on. (2008) 1–6
5. Damm, W., B. Josko, and T. Peinkamp: Contract Based ISO CD 26262 Safety Analysis. In: Safety-Critical Systems, 2009. SAE (2009)
6. Ellims, M., H. Monkhouse, and A. Lyon: ISO 26262: Experience Applying Part 3 to an In-Wheel Electric Motor. In: Sys. Safety, 6th IET Int. Conf. on. (2011) 1–8
7. Sinha, P.: Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives. Reliability Engineering & System Safety **96**(10) (2011) 1349 – 1359
8. ISO: 26262 - "Road vehicles-Functional safety" (2011)
9. Westman, J., M. Nyberg, and M. Törngren: Structuring Safety Requirements in ISO 26262 using Contract Theory. In: To be published at SAFECOMP 2013. (2013)