Functional Safety based on a System Reference Model

Manfred Broy
Institut für Informatik,
Technische Universität München
D-80290 München Germany
broy@in.tum.de

Abstract

Ensuring functional system safety comprises four major tasks. First, all possible hazards and risks of incidents with respect to functional safety have to be identified. Second, the system requirements specification must be shown to be valid in the sense that it excludes all the hazards with sufficiently high probability. Third, it has to be shown that the requirements are implemented correctly. Fourth, it must be demonstrated that for the implementation all possible failures of subsystems that could lead to violations of the functional safety requirements systems are excluded with a sufficiently high probability. This way it has to be shown that the specification and its implementation lead to an acceptable risk in terms of probabilities of violations of safety requirements. For a proper engineering of functional safety we suggest the use of a rigorous modelling framework. It consists of: a system modelling theory that provides a number of modelling concepts that are carefully related and integrated; a system reference model; and a reference architecture structuring systems into three levels of abstractions represented by views, including a functional view, a logical subsystem view and a technical view. It is demonstrated how, in this framework, all kinds of safety issues are expressed, analysed and traced; and how, due to the formalization of the framework, safety problems are formally analysed, specified and verified.

Keywords: Functional Safety, Hazards, System Modelling, Requirements, Specification, Design, Architecture.

1 Introduction

It is well accepted by now that software intensive systems - due to their functional power, their tight integration with human machine interaction, their safety critical functionality, and their additional complexity - bring in essential challenges to guaranteeing functional safety. Functional safety of systems addresses the general requirement that there is only a bounded, calculable, and acceptable risk that the usage of the system may result in harm for the health and life of people or other assets.

We suggest a systematic concept to categorize incidents and a comprehensive modelling approach to support functional safety.

Copyright © 2012, Australian Computer Society, Inc. This paper appeared at the Australian System Safety Conference (ASSC 2012), held in Brisbane 23-25 May, 2012. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 145, Ed. Tony Cant. Reproduction for academic, not-for profit purposes permitted provided this text is included.

1.1 System Development Steps and their Relation to Functional Safety

The development of systems follows a simple and clear structure:

- REQU: elicitation, analysis, and documentation of the requirements and their validation
- SPEC: functional specification of the system, verification of the specification w.r.t. the requirements
- ARCH: design of the architecture by decomposition into subsystems called components and their specification, verification of the architecture
- IMPL: implementation of the components and verification according to their specification
- VEIN: integration and system verification

This structure is reflected in the tasks to guarantee functional safety properties as follows:

- REQU: elicitation, analysis, and documentation of the safety requirements and their validation
- SPEC: verification of safety requirements on the basis of the functional specification
- ARCH: Failure-Modes-and-Effect Analysis (FMEA) on the basis of the architecture – identification of expected failures for components and their probability, analysis of the effects of failures, calculation of probabilities of failures and resulting violations of safety requirements
- IMPL: implementation of the components according to their specification, validation and verification of probabilities as requested in the FMEA
- VEIN: integration and system safety verification

This shows how tightly issues of functional safety are embedded into general system engineering steps, in particular model-based engineering

1.2 A Systematic Approach to Safety Issues

In this section we classify hazards and incidents along the lines of [Gleirscher 11].

1.2.1 Hazards and Incidents

A *hazard* characterizes a potential situation in the usage of a system that represents a degree of threat to life, health, property, or environment. A hazardous situation that has happened in the operation of a system is called an *incident*. A system is functionally safe if it is free of hazards and therefore there is no risk of incidents.

There are two basic ways to define functional safety for systems: empirical and analytical approaches. In an empirical approach, we consider the statistics of systems under operation with respect to incidents; in an analytic

approach we analyse a system with the goal to calculate the risk of incidents.

1.2.2 An Empirical View onto Functional Safety

There is obviously a clear pragmatic concept of functional safety in connection with systems and their operation. If we observe the operation of systems over a certain period of time we realize if and how often incidents happen and this way get an empirical assessment of hazards, risk of incidents, and functional safety.

In principle, in empirical approaches we do not need to identify hazards (possible situations that represent a degree of threat to life, health, property, or environment) in advance, but may identify, collect, and classify hazards as the result of empirical observations where hazards are identified via observed incidents. This is much more easy than to identify all hazards in advance, but cannot guarantee functional safety, but only monitor and evaluate functional safety during operation.

1.2.3 Analysing and Guaranteeing Functional Safety

When designing systems with the potential for hazards we have to exclude any unacceptable risk to come to the conclusion that there does not exist a safety problem with the system in operation. Clearly functional safety has the goal to avoid unacceptable risk and hazardous situations.

A systematic approach in avoiding unacceptable risk always consists of the following steps for a system under development:

1. Specification of the operational context (as part of domain modelling)
2. Identification of hazards
3. Specification of the system's functional behaviour excluding hazards
4. Analysis of possible defects and failures in the system and its subsystems leading to hazards
5. Measures to reduce the unacceptable risk in the system and its subsystems

If we follow such a systematic approach, we work with the following views:

1. Context behaviour as specified
2. System behaviour as specified
3. System behaviour as realized with defects both of systematic or probabilistic nature
4. Context assumptions and defects due to violations of the assumptions about the operational context

We consider the following classification of hazards (and related potential incidents) and their relation to specifications:

	Specification	Realization
Context	Hazard not identified and recognized in context specification	Violation of specification of operational context
System	Hazard not excluded by system specification	Violation of specification of system behaviour

All together we get the following classifications of reasons for incidents due to hazards:

Classification of incident	Cause of hazard
Hazard not identified in context specification	Errors in the analysis of the set of hazards and potential incidents; hazards that were not recognized in the elicitation of safety requirements
Hazard not excluded by system specification	Errors in the specification, either of the system or of the operational context, since the composition of the ideal system behaviour and the ideal operational context behaviour still allow for hazards
Violation of specification of system behaviour	Hazards and risk of incidents due to systematic or probabilistic failures in the system and its subsystems
Violation of specification of operational context	Hazards and risk of incidents due to violations of the idealistic assumptions about the context

A result of safety analysis should be probabilistically formulated bounds for the risk of hazards and incidents – bounds sufficient for the given safety requirements. If hazards and incidents happen during the operation of systems, we have to distinguish between: hazards and incidents, that are a result of remaining risks, just as analysed in the safety process; and hazards and risk of incidents, that have to be seen as a result of faults in the functional safety analysis.

Modelling techniques can help to analyse, in a systematic manner, functional safety issues. However, as we will show, we need a careful modelling of the system, its possible defects, the operational context, the assumptions about the operational context, and possible violations of assumptions about the operational context. The better such an approach is, the more reliable the safety analysis is.

What we demand and describe is along the lines of ISO 26262 which emphasizes:

> NOTE 2 There is a difference between
> to perform a function as required (stronger definition, use-oriented) and
> to perform a function as specified, so a failure can result from an incorrect specification.

This citation taken from ISO 26262 underlines the fact that a safety analysis falls short if it only shows the risk of hazards due to violations of the behaviour in terms of a function as specified; in contrast, a safety analysis also has to guarantee the absence of hazards in the empirical general sense as defined above. Note that there have been a number of serious incidents, for instance in air traffic, where systems reacted as specified, but the specifications were not adequate for functional safety since they did not match with the expectations of the pilots.

2 Modelling and Structuring Systems

In the following we introduce a short overview of system modelling techniques and architectural views. Fig. 1 gives a schematic illustration of a system and its operational context.

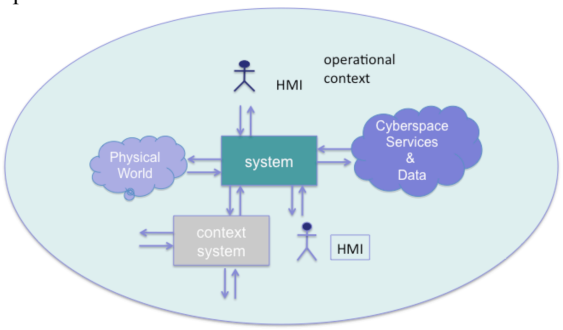

Figure 1: System and its Context

We use basically two frameworks for structured views onto systems

- modelling theory
- structuring of systems into adequate levels of abstraction

A key starting point is the fundamental concept of a system. The modelling and architectural framework has two parts:

- a family of mathematical and logical system modelling concepts for systems, addressing the notion of interface, state and architecture with two models of behaviour:
  - logical model: a system described in terms of interface, architecture and state – we distinguish between the interface view (black box view) and a glass box view
  - probabilistic model: a system described in terms of probabilities for its behaviours – more precisely probability distributions on sets of possible behaviours.
- a structured set of views – sometimes called comprehensive system architecture; it comprises the following views:
  - context
  - functional view (structured system interface behaviour):
    - hierarchy of system functions with modes of operation to capture their dependencies and their context
    - probability distribution on behaviours,
  - subsystem architecture view: hierarchical structure of subsystems (in terms of "logical components"),
  - technical and physical view: electronic hardware, software at design and runtime, mechanical and physical hardware, and their connections.

The two modelling frameworks are related and described in the following. We start by briefly introducing the modelling theory; for details see [Broy 12].

2.1 The System Modelling Theory

Our approach uses a specific notion of discrete system with the following characteristics and principles:

- A discrete system has a well-defined boundary that determines its interface.
- Everything outside the system boundary is called the system's environment. Those parts of the environment that are relevant for the system's operation are called the system's operational context.
- A system's interface describes the means by which the system interacts with its context. The syntactic interface defines the set of actions that can be performed in interaction with a system over its boundary. In our case syntactic interfaces are defined by the set of input and output channels together with their types. The input channels define the input actions for a system while the output channels define the output actions for a system.
- We distinguish between syntactic interface, also called static interface, which describes the set of input and output actions that can take place over the system boundary and interface behaviour (also called dynamical interface), which describes the system's functionality; the interface behaviour is captured by the causal relationship between streams of actions captured in the input and output histories. We give a logical behaviour as well as a probabilistic behaviour for systems.
- The interface behaviour of systems is described by: logical expressions, called interface assertions; by state machines; or it can be further decomposed into architectures.
- A system has an internal structure. This structure is described by in a state view by its state space with state transitions and/or by its decomposition into subsystems forming its architecture in case the system can be decomposed correspondingly. The subsystems interact and also provide the interaction with the system's context. The state machine and the architecture associated with a system are called its state view and its structural or architectural view respectively.
- In a complementary view, the behaviours of systems can be described by sets of traces, which are sets of scenarios of input and output behaviour of systems. We distinguish between finite and infinite scenarios.
- Moreover, systems operate in time. In our case we use discrete time, which seems, in particular, adequate for discrete systems. Subsystems operate concurrently within architectures.

This gives a highly abstract and at the same time comprehensive model of systems. This model briefly is formalized in the following.

2.1.1 Data Models – Data Types

Data models define a set of data types and some basic functions for them. A *(data) type* T is a name for a data set. Let TYPE be the set of all data types.

2.1.2 Interface Behaviour

Systems have syntactic interfaces that are described by their sets of input and output channels attributed by the type of messages that are communicated over them. Channels are used to connect systems to be able to transmit messages between them. A set of typed channels is a set of channels with a type given for each of its channels.

Definition. Syntactic interface

Let I be a set of typed input channels and O be a set of typed output channels. The pair (I, O) characterizes the syntactic interface of a system. The syntactic interface is denoted by $(I \blacktriangleright O)$. ❑

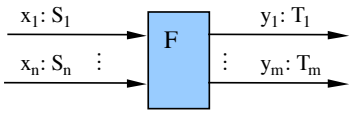

Figure 2: Graphical Representation of a System F as a Data Flow Node

Fig. 2 shows the syntactic interface of a system F in a graphical representation by a data flow node with its syntactic interface consisting of the input channels $x_1, \ldots, x_n$ of types $S_1, \ldots, S_n$ and the output channels $y_1, \ldots, y_m$ of types $T_1, \ldots, T_m$.

Definition. Timed Streams

Given a message set M of data elements of type T (M is also called the carrier set of type T), we represent a *timed stream* s of type T by a mapping

$$s: \mathbb{N}\backslash\{0\} \rightarrow M^*$$

In a timed stream s a sequence s(t) of messages is given for each time interval $t \in \mathbb{N}\backslash\{0\}$. In each time interval an arbitrary, but finite number of messages may be communicated. By $(M^*)^\infty$ we denote the set of timed infinite streams. ❑

A (timed) channel history for a set of typed channels C assigns to each channel $c \in C$ a timed stream of messages communicated over that channel.

Definition. Channel history

Let C be a set of typed channels; a (total) *channel history* x is a mapping (let $\mathbb{M}$ be the universe of all messages)

$$x : C \rightarrow (\mathbb{N}\backslash\{0\} \rightarrow \mathbb{M}^*)$$

such that x(c) is a timed stream of messages of the type of channel $c \in C$. $\vec{C}$ denotes the set of all total channel histories for the channel set C. ❑

For each history $z \in \vec{C}$ and each time $t \in \mathbb{N}$ the expression $z{\downarrow}t$ denotes the partial history (the initial communication behavior on the channels) of z until time t. $z{\downarrow}t$ yields a finite history for each of the channels in C represented by a mapping

$$C \rightarrow (\{1, \ldots, t\} \rightarrow \mathbb{M}^*)$$

$z{\downarrow}0$ denotes the history with empty sequences associated with each of its channels.

The behavior of a system with syntactic interface $(I \blacktriangleright O)$ is defined by a mapping that maps the input histories in $\vec{I}$ onto output histories in $\vec{O}$. This way we get a functional model of a system interface behavior.

Definition. I/O-Behaviour

A causal mapping $F: \vec{I} \rightarrow \wp(\vec{O})$ is called an *I/O-behaviour*. By $\mathbb{F}[I \blacktriangleright O]$ we denote the set of all (total and partial) I/O-behaviours with syntactic interface $(I \blacktriangleright O)$ and by $\mathbb{F}$ the set of all I/O-behaviours. ❑

Interface behaviours model system functionality. For systems we assume that their interface behaviour is total. Behaviours F may be deterministic (in this case, the set F(x) of output histories has at most one element for each input history x) or nondeterministic.

2.1.3 State Machines by State Transition Functions

State machines with input and output describe system implementations in terms of states and state transitions. A state machine is defined by a state space and a state transition function.

Definition. State Machine with Syntactic Interface $(I \blacktriangleright O)$

Given a state space Σ, a state machine (Δ, Λ) with input and output according to the syntactic interface $(I \blacktriangleright O)$ consists of a set $\Lambda \subseteq \Sigma$ of initial states as well as of a nondeterministic state transition function

$$\Delta: (\Sigma \times (I \rightarrow \mathbb{M}^*)) \rightarrow \wp(\Sigma \times (O \rightarrow \mathbb{M}^*)) \quad ❑$$

For each state $\sigma \in \Sigma$ and each valuation $a: I \rightarrow \mathbb{M}^*$ of the input channels in I by sequences of input messages every pair $(\sigma', b) \in \Delta(\sigma, a)$ defines a successor state σ' and a valuation $b: O \rightarrow \mathbb{M}^*$ of the output channels consisting of the sequences produced by the state transition. (Δ, Λ) is a *Mealy machine* with possibly infinite state space. If in every transition the output b depends on the state σ only but never on the current input a, we speak of a *Moore machine*.

2.1.4 Systems and their Functionality

Systems interact with their contexts via the channels of their interfaces. We identify both systems by names. A system named k has an interface, consisting of a syntactic interface $(I \blacktriangleright O)$ and interface behaviour

$$F_k: \vec{I} \rightarrow \wp(\vec{O})$$

The behaviour may be a combination of a larger number of more elementary sub-function behaviours. Then we speak of a *multifunctional* system.

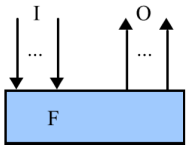

Figure 3 Graphical Representation of a Function Interface with the set of input channels I and the set of output channels O

Let SID be the set of system names. A system named $k \in$ SID is called *statically interpreted* in a system model or in an architecture if only a syntactic interface $(I_k \blacktriangleright O_k)$ is

given for k and *dynamically interpreted* if an interface behaviour $F_k \in \mathbb{F}[I_k \blacktriangleright O_k]$ is specified for component k.

2.1.5 Architectures

In the following we assume that each system used in an architecture as a component has a unique identifier k. Let K be the set of identifiers for the components of an architecture.

Definition. Set of Composable Interfaces

A set of component names K with a finite set of interfaces $(I_k \blacktriangleright O_k)$ for each identifier $k \in K$ is called *composable*, if the following propositions hold:

- the sets of input channels Ik, $k \in K$, are pairwise disjoint,
- the sets of output channels Ok, $k \in K$, are pairwise disjoint,
- the channels in $\{c \in Ik: k \in K\} \cap \{c \in Ok: k \in K\}$ have consistent channel types in $\{c \in Ik: k \in K\}$ and $\{c \in Ok: k \in K\}$. ❑

If channel names and types are not consistent for a set of systems to be used as components we simply may rename the channels to make them consistent.

Definition. Syntactic Architecture

A syntactic architecture $A = (K, \xi)$ with interface $(I_A \blacktriangleright O_A)$ is given by a set K of component names with composable syntactic interfaces $\xi(k) = (I_k \blacktriangleright O_k)$ for $k \in K$.

$IA = \{c \in Ik: k \in K\} \setminus \{c \in Ok: k \in K\}$ denotes the set of input channels of the architecture,

$DA = \{c \in Ok: k \in K\}$ denotes the set of generated channels of the architecture,

$OA = DA \setminus \{c \in Ik: k \in K\}$ denotes the set of output channels of the architecture,

DA\OA denotes the set of internal channels of the architecture

$CA = \{c \in Ik: k \in K\} \cup \{c \in Ok: k \in K\}$ denotes the set of all channels

By $(I_A \blacktriangleright D_A)$ we denote the *syntactic internal interface* and by $(I_A \blacktriangleright O_A)$ we denote the *syntactic external inter*face of the architecture. ❑

A syntactic architecture forms a directed graph with its components as its nodes and its channels as directed arcs. The input channels in I_A are ingoing arcs and the output channels in O_A are outgoing arcs for that graph.

Definition. Interpreted Architecture

An interpreted architecture (K, ψ) for a syntactic architecture (K, ξ) associates an interface behavior $\psi(k) \in \mathbb{F}[I_k \blacktriangleright O_k]$ for the syntactic interface $\xi(k) = (I_k \blacktriangleright O_k)$, with every component $k \in K$. ❑

An architecture can be specified by a syntactic architecture given by its set of subsystems and their communication channels and an interface specification for each of its components.

2.1.6 Probabilistic Interface View

We provide a probabilistic model for systems along the lines of [Neubeck 12]. Given a set of typed channels C we define a probability distribution for a set $H \subseteq \vec{C}$ by the function

$$\mu: H \rightarrow [0:1]$$

Let $\mathcal{M}[\vec{C}]$ denote the set of all probability distributions over sets $H \subseteq \vec{C}$.

Given a behaviour

$$F: I \rightarrow \wp(\vec{O})$$

its probabilistic behaviour is defined by a function

$$D_F: I \rightarrow \mathcal{M}(\vec{O})$$

where for every input history $x \in I$ by

$$D_F(x)$$

we get a probability distribution for every input history $x \in I$

$$\mu_x: \wp(F(x)) \rightarrow [0:1]$$

We get a probability $\mu_x(Y)$ by the function μ for every measurable set $Y \subseteq F(x)$ of output histories. This shows that μ defines a probability distribution μ_x for every input history $x \in I$ on its set F(x) of possible output histories.

2.2 Overall Structuring of Systems into Levels of abstraction

We choose a systematic structuring of systems and their contexts using the following categories.

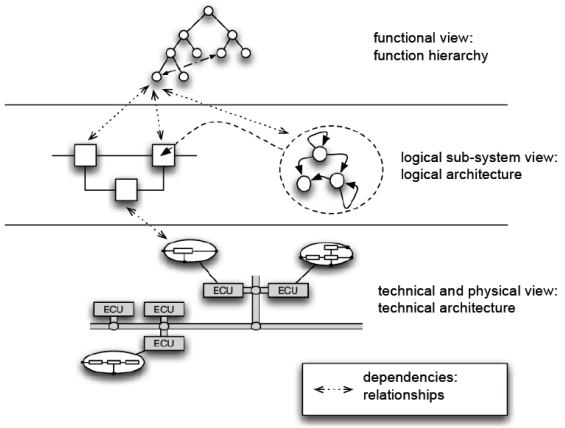

Figure 4 Levels of Abstraction Taken from [Broy et al. 08]

We structure the properties of systems into a number of views that are the result of viewpoints. We use three fundamental views:

- usage: function and context
- design: (logical) subsystem structure
- implementation: technical, physical, syntactical representation and realisation

Each view uses modelling concepts taken from a basic set of modelling elements

- interface and interface behaviour in terms of the interaction over the system boundaries
- architecture and architectural behaviour in terms of structuring a system into a set of subsystems

and their connection by communication channels and its interaction between the components and over the system boundaries
- state and state transition behaviour in terms of describing the state space of a system, its state transitions triggered by interaction.

For behaviour we distinguish

- logical behaviour in terms of the correct patterns of interaction
- probabilistic behaviour in terms of the probability of certain patterns of interaction.

These different aspects of behaviour apply to all three modelling concepts interface, state, and architecture.

3 Key Challenges for Functional System Safety

As we can see from the categorization of incidents, in hazard classification it is essential to analyse what can go wrong at the level of the specification and design, and what are the effects of failures of subsystems (as identified by FMEA). In particular, it is essential to make sure that, first of all, no potential hazards are overlooked in domain modelling and that the functional specification excludes all the hazards with sufficiently high probability. In particular, a very difficult task is to find out to what extent a particular system design may lead to failures in its operational context; this includes especially errors of humans operating the system.

There are quite a number of incidents, in particular in avionics and perhaps less spectacular and less well analysed also in the operation of other systems such as cars, boats and trains that are due to wrong reactions by their users, such as pilots. In these cases, the system functions were specified and implemented in such a way that users get confused and could not operate systems properly as expected in particular situations and as required by the identified user groups.

3.1 System Boundaries and Hazards

A hazard is due to certain critical events inside a system or in its operational context. Therefore we distinguish two categories of hazards for a system under safety analysis:

- Intrinsic hazards are hazards that result in incidents inside a system; as an example take a battery together with its control unit, which is a system that might explode or catch fire
- Extrinsic hazards are due to incidents that happen in the operational context of a system that are under the control of the system; for example, the explosion of a battery due to a fault in the control system is an extrinsic hazard from the perspective of the control system (where the battery is part of its operational context).

In safety analysis we have to capture and analyse and exclude both intrinsic and extrinsic hazards. Extrinsic hazards are related to the interface behaviour and the functionality of systems. Intrinsic hazards become extrinsic if we change the scope and focus the system under analysis such that the critical events are no longer part of the system. An example is the shift of the focus from a battery together with its control unit to the control unit with the battery as part its operational context. The change of scope is typically a result of design and system decomposition.

3.2 Domain Modelling

One particular important issue to find out about hazards is a very precise understanding, analysis, and modelling of the system's operational context. Typically incidents happen in the operational context. There are two difficulties that have to be mastered in domain modelling as basis and part of safety analysis.

3.2.1 Identifying and Modelling Hazards

First of all we have to understand what potential hazards are. So we have to carry out a careful analysis of the environment and operational context to find out about hazards. This is very much related to the task of requirements engineering.

The similarities between hazard analysis and identification and requirement analysis and identification are obvious. It is a difficult issue to find out about all the actual requirements. Forgetting a requirement leads to a system that does not fulfil the user expectations in some respects. In analogy, overlooking a possible hazard leads to a safety analysis in which no measures are undertaken to ensure that this overlooked hazard is not happening or that the probability of it happening is low enough. There are quite a number of practical examples where such a problem has happened (example: Titanic).

In both cases of requirements engineering and hazard analysis the completeness of a specification cannot be verified but has to be checked by validation. A careful validation of the result of the hazard analysis and identification is mandatory. [Gleirscher 11] discusses environment modelling for hazard analysis and for hazard-oriented derivation of scenarios for specification validation and system testing.

3.2.2 Relating Domain Specific Levels of Abstraction

A particular difficulty results from the different levels of abstraction for the formulation of safety requirements. Fig. 5 shows schematically four chunks of system properties from an example inspired by [Kondeva 12].

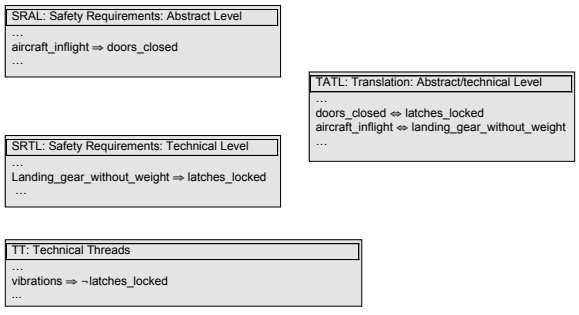

Figure 5 Safety Requirements: From Abstract to Technical Level and Threats at Technical Level

At the abstract level safety requirements are formulated in application domain oriented language addressing key concepts and notions of the application domain. At the

technical level the same safety requirement is expressed in technical terms. This "translation" is not as simple as the one in the illustration. As stated above, it is an inference, based on the architectural structure and the behaviour of the subsystems in the structure. There is no chance to produce this manually (see [Struss, Fraracci 11], [Struss 11]). This has to be generated, and this is exactly what model-based prediction for the physical components has to deliver. We need a translation of the abstract safety requirements into the technical ones in terms of logical assertions that formalize this relationship. This relation is part of the domain model. We have to show

$$SRTL \wedge TATL \Rightarrow SRAL$$

(see the example in Fig. 5). Then, on the technical level, additional technical threats have to be and can be identified that are hard or even impossible to find at the abstract level. In the example in Fig. 5 we get some inconsistency and thus a contradiction to safety requirements in SRTL if we assume that there may vibrations while the aircraft is in flight that they in term might unlock the latches. Such inconsistencies can be checked and found by SAT solvers.

Of course, this change of levels of abstraction typically continues. At the technical level, there does not exist the signal "Landing_gear_without_weight". There exists: "no signal at the pin connected to the weight sensor". This technical view is essential in order to analyse the impact of a broken weight sensor, open wires and connectors between sensor and ECU, shorts of the wires, etc.

3.3 Modelling Context, HMI and Safety

Hazards can only be caused by a system in the interaction between the system and its operational context.

3.3.1 Hazards as Result of the Interaction between Systems and their Operational Context

Another issue is to understand how such hazards in the operational context are triggered by the system. This leads to the necessity to have a kind of a formalisation of the interaction between the operational context and the system.

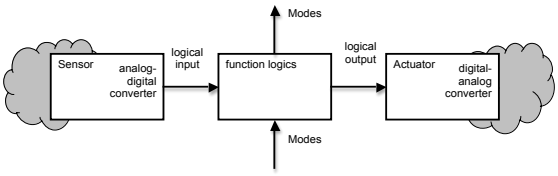

Figure. 6 Schematic split of a function in hybrid pre- and post-processing

More precisely we have to model the operational context including extrinsic hazards and incidents as they may appear in the operational context. Only if such modelling is done in a sufficiently formal way, we can start to get estimations of the bounds on the risk of incidents and hazards (see [Struss, Fraracci 11]).

3.3.2 Hazards as Result of the Interaction between Systems and Their Users

Clearly, such a modelling can be very difficult because the operational context, in particular, has to deal with the user interface and the way users operate a system. In principle, we have to look at issues such as: "what is the probability that a user presses a wrong button in a particular situation?" and to find out what is the psychological analysis that is needed to speak about those probabilities.

This shows that in the analysis of functional safety the logical and the technical user interface have to be looked at and analysed very carefully.

Second we have to deal with issues in approaches like use case analysis. One way to do this would be to identify intrinsic and extrinsic hazards and then to develop a number of anti-use-cases that describe scenarios of hazards happening and to analyse what are the possibilities to avoid these hazards. All this activity can be and must be done quite independently of the question of the necessary and additional FMEA to make sure that the system as specified with an operational context as modelled does the right thing. Today, in practice, functional safety analysis is often too much focused onto FMEA and vulnerability impact analysis with the danger to miss hazards that are not due to defects of subsystems.

3.4 Functional Modularity and Extrinsic Hazards

Note that strictly speaking, in terms of extrinsic hazards, it is not the system that is safety critical but its functions. In fact, in a safety analysis we have to identify the safety criticality of the functions. This goes hand in hand with modelling the operational context; we can see how the functions are connected to the operational context and which of the functions may cause hazards. In addition, we have to consider a number of failure assumptions for the functions that have to be related to FMEA and then find out which are the functions and the output provided by those functions as safety critical aspects. Then we can analyse which error deviations of functions we can tolerate and which error deviations we cannot tolerate and where we have to be sure that they can happen only with a certain sufficiently low probability.

Today we typically deal with so-called multi-functional systems. These are systems that introduce and offer a large number of different functions as pointed out in [Broy 10]. These functions have to be specified in a modular way, in spite of the fact that they are usually not logically independent. There are behavioural dependences between these functions. When mastering the specification of systems from a safety point of view, we have to deal with the different functions that are part of the functionalities.

As shown in [Broy 10] it is possible to identify and specify dependences between functions. If there is a function F that depends on another function F' and if the dependency of these functions may lead to hazards then the function F' (which, considered in isolation, is not safety critical) has to be treated as a safety critical function, if the dependency may lead to extrinsic hazards. More precisely, using these dependencies we can

introduce a directed dependency graph with functions as nodes. Then we identify the functions that are safety critical. This way we to stick to a kind of propagation and inheritance of safety criticality levels such that a highly safety critical function F may pass on its safety level to functions F' that show dependencies to F.

3.5 Tracing and Safety#

Finally, in standards for functional safety, tracing is required for safety critical functions. Unfortunately, what we see as a foundation of tracing in the scientific literature so far is not sufficient. Based on the proposed modelling framework, a very rigorous approach to tracing is possible by representing all the properties of systems within a formalised logical framework. This way we can introduce a completely formalised concept of tracing.

In doing so we get a precise concept of what tracing is. In particular, we can study traces between general requirements, functional specification and architectural decomposition. Such an approach provides a firm framework for defining what traces are but at the same time it addresses the question of how dense traces are and how many traces we need. By the approach we see how difficult and complex tracing is. Here we need more research and also empirical studies.

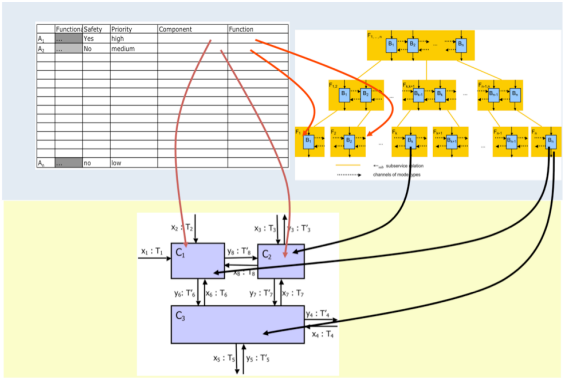

Figure 7 Tracing between Requirements, Functional Hierarchy, and Logical Subsystem Architecture

Recently, we have performed a number of empirical studies about dependencies between functions in trucks to find out about how many dependencies we can expect between those functions. Similar numbers are not available for dependencies between requirements, functional specification, and architecture.

4 Summary and Outlook

We have introduced and sketched a rigorous framework of modelling that allows us to capture logical and probabilistic properties at different levels of abstraction. We believe that such a rigorous framework allows for modelling that can be used both for system specification, design and implementation, for verification including test case generation, for safety analysis as well as for diagnoses.

Using a rigorous modelling approach we model the system as well as its operational context. We recommend to distinguish and to model intrinsic as well as extrinsic hazards. We, in particular, recommend validating the specification carefully to make sure that hazards are not implied by it. Doing so, we can apply all kinds of automatic analysis and verification techniques to deal with functional safety. In any case, the quality of functional safety analysis depends on the expressive power and the adequate application of the modelling techniques and methods.

Acknowledgements

It is a pleasure to acknowledge helpful and stimulating discussions with Mario Gleirscher, Antoaneta Kondeva, and Peter Struss.

5 References

M. Broy: The ‚Grand Challenge' in Informatics: Engineering Software-Intensive Systems. IEEE Computer, Oktober 2006, 72–80

M. Broy, I. Krüger, M. Meisinger: A Formal Model of Services. TOSEM - ACM Trans. Softw. Eng. Methodol. 16:1 Feb. 2007

M. Broy: Model-driven architecture-centric engineering of (embedded) software intensive systems: modelling theories and architectural milestones. Innovations Syst. Softw. Eng. 3:1, 2007, 75-102

M. Broy: Multifunctional Software Systems: Structured Modeling and Specification of Functional Requirements. Science of Computer Programming 75 (2010), S. 1193–1214

M. Broy: Software and System Modeling: Structured Multi-view Modeling, Specification, Design and Implementation. In: Conquering Complexity, edited by Mike Hinchey and Lorcan Coyle, Springer Verlag, January 2012, S. 309-372

M. Broy, M. Feilkas, J. Grünbauer, A. Gruler, A. Harhurin, J. Hartmann, B. Penzenstadler, B. Schätz, D. Wild: Umfassendes Architekturmodell für das Engineering eingebetteter Software-intensiver Systeme. Technische Universität München, Institut für Informatik 2008, TUM-I0816 (Technical Report)

M. Gleirscher: Hazard-based Selection of Test Cases. In: Proc. 6th ICSE Workshop on Automation of Software Test (AST'11), May 2011

M. Gleirscher: Behavioural Safety of Software-controlled Physical Systems. Ph.d. Thesis forthcoming 2012

A. Kondeva: Safety-based Requirements Engineering: Systematic refinement and specification of safety requirements in the avionic domain. Ph. D. Thesis forthcoming 2012

P. R. Neubeck: A Probabilitistic Theory of Interactive Systems. Ph. D. Thesis forthcoming 2012

P. Struss, A. Fraracci: FMEA of a Braking System - A Kingdom for a Qualitative Valve Model. In: 25th International Workshop on Qualitative Reasoning, Barcelona, Spain, 2011

P. Struss: Automated Failure-modes-and-effects Analysis of Embedded Software (Extended Abstract). In: 2nd International Workshop on Software Health Management, SHM-2011/4th IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT), Palo Alto, 2011