Flexible, any-time fault tree analysis with component logic models

Marc Förster, Daniel Schneider

Fraunhofer-Institut für Experimentelles Software-Engineering (IESE)
Fraunhoferplatz 1
67663 Kaiserslautern, Germany

{marc.foerster | daniel.schneider}@iese.fraunhofer.de

Abstract—This article presents a novel approach to facilitating fault tree analysis during the development of software-controlled systems. Based on a component-oriented system model, it combines second-order probabilistic analysis and automatically generated default failure models with a level-of-detail concept to ensure early and continuous analysability of system failure behaviour with optimal effort, even in the presence of incomplete information and dissimilar levels of detail in different parts of an evolving system model. The viability and validity of the method are demonstrated by means of an experiment.

Embedded systems, fault tree analysis, model-based development, safety, software

ACRONYMS & ABBREVIATIONS

CFT Component fault tree
CLM Component logic model
FM(EA) Failure models (and effects analysis)
FT(A) Fault tree (analysis)
IS Interface specification
LOD Level of detail
PMF Probability mass function
RC Realisation composition
RI Realisation interface
SM Service mode
TF Transfer function
TS Transfer specification

I. INTRODUCTION

In the development of software-intensive and safety-critical embedded systems, fault tree analysis (FTA) plays an important role, which is emphasised by safety norms requiring the application of safety models and analyses for both hardware and software [14] [18]. FTA, however, is (in)famous for being labour intensive and inflexible and is therefore often used only in late phases where it is of little use for optimising a system’s design in a dependability-oriented manner.

Since every design decision on one level critically determines the range of choices available on all subordinated hierarchy levels, most of them have to be made in the face of uncertainty. In early phases, the component structure of a software system—its architecture—will not be definite; functions and the reliability required of them will be clear only on a coarse level of detail. Furthermore, a typical development process will mostly include a variety of approaches as well as changing models and artefacts on different levels of granularity and maturity. Parts of a system may be built bottom up from preexisting components, while others require more or less thorough adaptation and still others are completely new, top-down developments. Some failure models may be derived with tool support from functional models, while others are the result of purely human effort, their quality and detailedness highly dependent on the experience of the engineers involved. Fault tree models may state a generic value error as top or basic event; others may include a specification as detailed as “Signal value exceeds expected value by more than 10%”.

How, then, can we integrate such diverse and often uncertain and evolving information in one coherent dependability model? This article introduces a lightweight, FTA-like and component-oriented approach to performing this task, extending concepts that were presented in [10]. We combine them with a flexible dependability modelling framework featuring multimodal modelling on different levels of detail, the facilitation of quantitative analyses under uncertainty, and any-time analysability as soon as component and service failure modes have been determined. Instead of building a fault tree from scratch, we propose proceeding in the opposite direction: starting with an automatically generated, complete, worst-case characterisation of a system’s failure behaviour and pruning it until analysis shows it to be satisfying.

Section II defines basic aspects of the model that forms the basis for the proposed analyses. Section III explains principles of modelling and analysis. Section IV validates the approach by means of an experiment. Section V reviews related work, while Section VI concludes and gives an outlook on future work.

II. THE COMPONENT LOGIC MODEL

Component fault trees (CFT) [16] have been around for some time. Since they are still an evolving concept with emerging variants [6] [7], it is in order to take a snapshot and describe in more detail the concepts and some extensions of CFT which constitute the component logic model (CLM).

A. Basic CFT & CLM concepts

The rationale behind the introduction of CFT was the difficulty to manage fault trees in practice and the desire to associate fault tree models with components and artefacts of modern component-based software engineering (CBSE). Traditional fault tree models of complex systems tend to assume wallpaper dimensions and are not compositional. They lack support for separation of concerns, division of labour, and reuse, and are thus hard, if not impossible, to integrate with models and
workflows used in CBSE. Component-oriented failure models such as CFT or CLM aim at alleviating these problems.

Figure 1 represents an abstract view of the CLM for a traction control system (TCS) (Figure 6 has a structural view of the same system). Its building blocks are ports, components, specifications, realisations, and Boolean functions. In general, a CLM component has a specification (white rounded rectangle) and a realisation part (grey octagon). The specification has two basic constituents (white ovals): the interface specification (IS) containing specification ports (white square inports and black square outports) and the transfer specification (TS) with transfer functions (Boolean functions, fault trees) mapping input service and internal modes of a component to output SM. Every port represents a service either provided or required by the component.

![Figure 1. Conceptual view of a CLM](image)

A component realisation has a realisation interface (RI) (grey hexagon) containing realisation ports, and a realisation composition (RC) (grey polygon), which in turn consists of component specifications. These serve as placeholders for subcomponents that work together to fulfil the specification of their supercomponent. The RC of the TCS consists of a throttle delimiter (ThrD) and an enabler (Enab) subcomponent. The throttle delimiter is realized using a calculator (Calc) and a sanity checker (SChk); the enabler realisation has another sanity checker and an interpolator (Interpol). The sanity checkers share both their IS and TS, which is indicated by dotted grey lines meaning the nodes could actually be merged. In a similar way, subcomponents may share a realisation, but this is not the case for the TCS.

The specification part of the TCS is not of interest here, therefore it is greyed out. Likewise, the TS of the throttle delimiter and the enabler are left unspecified. The TS of those subcomponents that are not refined anymore, however, need to be defined for the CLM to be complete. Refinement, that is, the relation between a component specification and a realisation, is indicated by a wide line connecting the two. For a CLM, these are natural variation points, as a (sub)component specification may be realised in different ways.

Depending on the direction of development (bottom-up or top-down), the specification can be seen as belonging to the realisation of a subcomponent (as the publication of its interface) or to the supercomponent (as constraints on the realisation). Therefore, the TCS root component and the ThrD subcomponent share the ThrD specification, indicated by the intersection of their dotted lines.

![Figure 2. CFT connections and port relations](image)

One important aspect of the traction control system CLM that is not depicted in Figure 1 are the relations among ports, which symbolise the data flow. Inside the model of a component, some of these relations are represented by directed arcs, connecting sources (provided services) and targets (required services) with each other. Figure 2 shows a detail of the actual TCS CLM in greyscale view, with arcs. Note that ports on the outside of a wide, black component border belong to the interface specification, corresponding ports on the inside belong to the realisation interface. The enabler expects three services from its environment, with one of them coming from the throttle delimiter which, in turn, delegates the provision of the service to its sanity checker. The connection/data flow structure of a single CLM component \( X \) is defined by a relation

\[
R_X \subseteq S_X \times T_X ,
\]

where \( S_X \) is the union of the set of sources of the realisation interface of \( X \), \( RI_X \), with the sources of the specification interfaces of all subcomponents of \( X \), \( SI_{sub(X)} \), and where \( T_X \) is the union of the set of targets of the realisation interface of \( X \), \( RI_{out}(X) \), with the set of targets of the specification interfaces of all subcomponents of \( X \), \( SI_{out}(sub(X)) \). This relation is symbolised by directed arcs. To complement this, there are two refinement relations, \( R_{in}^H \) and \( R_{out}^H \) (\( H \) as a mnemonic for hierarchical), relating the specification ports of \( X \) to those of its realisation:

\[
R_{in}^H \subseteq IS_{in} \times RI_{in} ,
R_{out}^H \subseteq RI_{out} \times IS_{out} ,
\]
where $I_{inX}$ is the set of imports of the IS of $X$ and $R_{inX}$ is the set of imports of the RI of $X$, and where $R_{outX}$ is the set of outports of the RI of $X$ and $I_{outX}$ is the set of outports of the IS of $X$. Combined, the component hierarchy and port relations form a structural system model that can be directly mapped onto an architecture expressed with modelling and architecture description concepts from SysML, Matlab/Simulink, or KobrA [7], for example.

B. Mode logic

We propose associating a (sufficiently) complete specification of the service modes of a component with its interface, on different levels of detail. Therefore, it is useful to include the relationship between these modes and their hierarchy in the model.

In a CLM, a service (with its basic modes nominal and aberrant) is characterised by dimensions (corresponding to the domains of [3]), for example, timing and content (Figure 3). Dimensions are concretised by aspects, such as value, correctness, or veracity. These aspects correspond to SHARD concepts (the SHARD method [20] uses the HAZOP-related terms omission, commission, early, late, and value—detectable or undetectable—for characterising failures of a signal/service).

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Timing</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Illegal</td>
<td>Illegal</td>
</tr>
<tr>
<td>Correctness</td>
<td>Illegal</td>
<td>Correct</td>
</tr>
<tr>
<td>Veracity</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>Mode</td>
<td>renormal</td>
<td>abnormal</td>
</tr>
<tr>
<td>(primary)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>renormal</td>
<td>abnormal</td>
</tr>
<tr>
<td>(secondary)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>+-----+-----+</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Excerpt from the service mode hierarchy

It makes sense to extend the SHARD set. Considering that the content of a piece of data of a service, especially for embedded systems, is taken to symbolise propositions regarding (physical) facts of the environment, it can deviate from its requirements in at least two ways: (a) the actual value may be illegal, which violates value bounds (intrinsic incorrectness) and (b) the value may be extrinsically incorrect, that is, the encoded fact is actually not the case.

If a digital fever thermometer, for example, is built to measure body temperatures from 30 to 42 °C, a temperature of 28 °C communicated by the sensor to the display driver is out of bounds but can be true—a serious case of hypothermia—as well as false. A temperature of 37 °C would comply with the specification but might either be sufficiently close to reality or not. Also, the actual value may influence not just the functional but also the failure behaviour of a component. Therefore, we use two different aspects in the content dimension to describe a service, correctness (with the same meaning as the SHARD term value) and value (for the actual, functional, data value delivered by a service).

For detectors (detection measures) [1], a special aspect, veracity, is introduced. If it is true, the detector works as it should, that is, it reports an error when there is one and does not when the monitored service behaves correctly. False veracity is the union of a false positive and a false negative reporting by the detector; accordingly, false veracity may be subdivided into type I and type II failures.

On the next level of detail, every aspect has two primary modes, one for nominal states/behaviours, one for aberrant states/behaviours. These primary modes can be further subdivided into secondary modes, if necessary (for example, false veracity as the union of false positives and false negatives). Underlying every aspect is a scale (Figure 3 indicates interval and nominal scales) which is partitioned by the modes such that modes of one aspect are mutually exclusive. When two ports are related in a model, their required and corresponding provided modes must either correspond on the underlying scale or be mapped if partitionings do not match exactly. For now, we postulate that the projection of modes of related services onto their scale is actually identical.

Assuming dimensions and aspects of a service to be essentially independent, the hierarchy of services, dimensions, aspects, and modes provides an implicit logic relating the described levels of detail. A service is in its nominal condition if all its dimensions are; it is aberrant if any one dimension is. Likewise, a dimension is nominal if all its aspects are; if any one aspect is aberrant, the dimension is, too. Modes, being equivalence classes, are combined by disjunction to yield their superordinated mode or aspect, respectively. Figure 4 shows the logical relations of service LOD, conceptualised as residing in a service port (conjunctions and disjunctions with one single argument are equivalent to identity). They are orthogonal to the data flow structure and relate the different LOD such that the transfer functions of two connected components can be combined even if they belong to different LOD (Section III.A).

C. Transfer functions

CLM specify nominal as well as aberrant behaviour, and they support multimodal modelling of service modes. For this reason, instead of the term fault tree associated with bimodal models and traditional basic events, we use the term transfer function (TF). In the field of signal processing, it describes the input-output relationship of linear, time-invariant systems such as filters. Unlike traditional FTA’s basic events, the basic causes (Section II.D) of CLM are not primarily connected to the idea of time-dependent probabilities of failure states, so the filter concept provides an appropriate image. TF map the modes of a
component’s required services and its internal modes to modes of provided services.

D. Basic causes

In CLM, the role of basic events of traditional FTA is played by basic causes (BC). BC are internal to transfer functions and represent those aspects and their modes (basic modes) pertinent to the component associated to a TF. For quantitative analysis, BC modes are assigned probabilities or, for second-order analysis, probability ranges (Section III.E).

E. Summary

CLM, as introduced here, present some differences to the original CFT concept [16]:

- CLM syntax (hierarchy) and semantics (transfer functions) are clearly separated (in CFT they could be mixed, meaning that subcomponents would be found “inside” fault trees, complicating the logical structure considerably);
- there is a strict association in CLM of modes with services (while in CFT, ports could be associated with arbitrary propositions and were not automatically assigned to a particular data flow);
- CLM are multimodal on every level and include non-failure (“success”) modes as well as modes for functional value ranges (CFT enabled multimodal Markov chains to be used like basic events, but were essentially bimodal otherwise).

Necessary criteria for (structural as well as semantical) consistence of a CLM are that

- for the relations $R^{i \rightarrow j}$, $R^{i \leftarrow j}$, $R^{k \rightarrow l}$, their respective inverse $R^{i \rightarrow j}$, $R^{i \leftarrow j}$, $R^{k \rightarrow l}$ is left total and right unique (or, equivalently, is a function);
- the relation of modes, aspects, and dimensions between a provided and a required service (port) is injective and left total.

Condition (1) means that a requisition must always be met (there must be an arc pointing to a required-port, and a provided-port of the specification/realisation must be mapped to by a required-port of the corresponding realisation/specification). Condition (2) ensures that the modes of a provided service can, in principle, be handled by the component requesting it. The exact mapping of modes and ports is not in our focus here; thus, we assume required and provided modes as well as ports to be completely compatible/identical and to have bijective relations.

CLM offer three well-defined degrees of freedom for system modelling: (1) structure change, by substituting one realisation with another or by altering the connection structure/service flow in the hierarchy graph (relations $R^{i \rightarrow j}$, $R^{i \leftarrow j}$, $R^{k \rightarrow l}$); (2) change of transfer functions; (3) adaptation of probabilities and probability ranges assigned to basic causes and services, their dimensions, aspects, and modes. This opens up systematic variation possibilities for the application of optimisation algorithms as well as for the integration with variation models, such as decision models in product line engineering.

III. PRINCIPLES OF MODELLING & ANALYSIS

Before describing the parameters and results of our experimentation, we introduce properties that are of interest for understanding the modelling and analysis of CLM.

A. Default transfer functions & design patterns

As soon as a preliminary architecture has been defined and all service modes have been determined (for example, by interface-focussed FMEA—IF-FMEA [19]), it should be possible to perform an analysis of the system, without first conducting a complete fault tree analysis. For this purpose, at least those components that do not (yet) have a realisation composition will have to get a default transfer specification. The default TS can be generated automatically and combines the modes of the internal basic causes and those of required services as follows:

1. all provided nominal (success) modes are assigned a TF consisting of a conjunction of all internal and required primary nominal modes (which, in turn, are a disjoint combination of their secondary modes);
2. all provided aberrant (failure) modes are assigned a TF consisting of a disjunction of all internal and required primary aberrant modes (which also are a disjoint combination of their secondary modes). For quantitative analysis, modes are assigned a $[0, 1]$ interval as their probability range, which is sampled during second-order analysis by Monte-Carlo simulation (Section III.E).

The resulting default TS is equivalent to a worst-case assumption: if there are no aberrant modes, the system is working as it should; if anything goes wrong, the system will fail in every specified way. Likewise for mode probabilities: they might be 1 or 0, or anything in between. It is now up to the fault tree analysts to improve upon the default logic. This improvement can be made incrementally, TF by TF and probability range by probability range (reducing them from the unity interval to smaller ranges), while the overall system model stays analysable at every moment. In the beginning, of course, analysis results will hardly be exciting; there are nothing but single points of failure, and the probability mass functions of provided top-level aberrant modes (equivalent to FTA’s top events) show a near-perfect bell curve over the $[0, 1]$ interval [10]. Nevertheless, the point here is that model goodness can be appreciated with a glance at the probability mass functions of top-level modes. Tradeoff analyses between design alternatives are possible at any time.
Default transfer specifications can also be used to capture design patterns in a logical way. As an example, consider the case of redundancy when a component receives two or more equivalent services of which only one needs to be correct. Figure 5 shows a default TS for simple redundancy: the component’s output is nominal when its internal workings are correct and one of the redundant required services is. The complementary TF for the aberrant top mode now needs both of these required services to fail. This places a constraint on the TF on higher, more detailed LOD (Section III.B), which helps to concretise them. Depending on the exact functioning of the redundancy mechanism, a more detailed analysis may reveal a behaviour that contradicts the abstract TS. For example, a specific failure of just one required service could still result in a problem. This can be pointed out by qualitative analysis, and can be accommodated accordingly.

B. Derivation & constraint of transfer functions between LOD
At the beginning, default transfer functions are propagated from their original level of detail (derived from FMEA) to all other LOD. Propagation rules are the same as for asserted functions (see below), but for default TF they have a trivial result: aberrant modes on all levels are assigned the same default function.

Starting from such a default TS, the component logic is concretised step by step, replacing default transfer functions by more specific ones on each level. These TF are asserted functions. Asserted TF are propagated in two directions: towards the more generic or towards the finer-grained LOD. On the failure side of the model, TF on higher LOD are nested inside lower-level TF they are related to by mode logic. A TF \( f_i \) is nested inside another TF \( f_j \) if, for all argument vectors \( v \), \( f_j(v) \leq f_i(v) \); in other words, wherever \( f_i \) is false, \( f_j \) is false, and wherever \( f_j \) is true, \( f_i \) is true (with \( false < true \)). When \( f_i \) subsumes \( f_j \) (that is, \( f_i = f_j \cup f_i \)), and \( f_j \) is replaced by \( f_i \) such that \( f_i < f_j \) (or \( f_i > f_j \), respectively), then \( f_i \) is reduced (widened) accordingly. In the other direction, towards higher LOD, when \( f_i \) is replaced by \( f_j \) such that \( f_i \preceq f_j \) (or \( f_j \preceq f_i \) respectively), then \( f_j \) and \( f_i \) are reduced (widened) and replaced by \( f_i \) and \( f_j \), where

\[
\begin{align*}
    f_{i*} &= f_i \cap f_j \quad \text{(reduction)}, \\
    f_{j*} &= f_i \cup f_j \quad \text{(widening)}.
\end{align*}
\]

Note that, if \( f_j \) and \( f_i \) had not before “overlapped”, \( f_{i*} \) and \( f_{j*} \) will after widening: \( f_{i*} \cap f_{j*} \neq \emptyset \). Thus, the new transfer functions are derived.

In a similar way, TF of nominal output modes can be specified as asserted: instead of including combinations of causes that could have an unwanted effect, from the set of all possible combinations one excludes those that are certain to be impossible or irrelevant. Together with the use of default TF, such a “pruning” approach helps to improve the reliability of the FTA process itself, and of the resulting models, because relevant combinations of failure causes are less likely to be overlooked.

TF of one aspect of a provided service (including the success functions of nominal modes) might, however, not completely cover unity. This means that there are argument vectors which do not get mapped to any provided mode because no transfer function evaluates them to \( true \). Such “holes” can be found by analysis and should be closed by uniting them with one or more existing TF or by adding a new one.

C. Nondeterminism
Given that, in principle, modes of the same aspect of the same service are mutually exclusive, it may seem counterintuitive at first to let them overlap such that the transfer functions of two modes can assume the value \( true \) for the same combination of required (incoming) modes. Furthermore, we actually force them to overlap by widening a superordinated TF. This is not a bug but a feature of CLM.

When a combinational model, such as fault trees/Boolean functions or even multivalued functions with multivalued variables, is used to describe the behaviour of a complex system in an abstract fashion, it would be rather surprising if this model were able to capture system (failure) behaviour perfectly. In this case, the model is not an abstraction anymore but, in a sense, equivalent to the system, which is hard to imagine for all but the simplest devices. In reality, the fact that two TF intersect means that we cannot decide, for certain variable assignments and a provided service, whether it will result in one or the other mode. This is very natural, and analysing the intersection can give valuable information about model and system. Taking this further, the degree of overlap can serve as a measure of the ability of the model to discriminate between different service and system states: if it is total, that is, if the values of TF are the same for all input vectors, it is impossible to distinguish the associated provided modes; on the other hand, if the intersection is empty (but there are no holes), we have perfect discrimination at least for a certain aspect of a service.

D. Transfer function synthesis by BDD
Transfer functions are kept separate per component in CLM and have to be combined for the analysis of a top-service mode. This can essentially be done in two ways: (1) all transfer functions that contribute to the top mode in focus are put together, and the resulting monolithic function is then transformed into a binary decision diagram (BDD) and analysed; (2) the relevant TF are separately converted to BDD, which are then merged to yield the global BDD of the entire function. We have opted for the second solution because (a) TF can be transformed into BDD offline whenever they change, and the result can be stored along with the model; (b) the separate BDD are smaller and may be reduced in size by variable reordering much easier than the global BDD. Both reasons help to reduce analysis effort and have been successfully put to use in an existing CFT tool [13][15].

E. Aspects of analysis
Often, we do not know how reliable exactly a service could or should be. Thus, uncertainty (present especially at the beginning of system design) as well as multimodality determine our approach to CLM modelling and analysis.

To successfully navigate the design space in a dependability-oriented way, we would like to know, at any point, the answer to
the question ‘If I take this decision (and not the alternative), what is the likelihood of the system turning out “good”?’. If goodness is the probability of the system delivering its intended service(s), the question asks for the probability of probabilities, in other words, for a second-order probability.

Randomly sampling every mode’s interval many times and evaluating the TF based on those values results in the approximation of such a second-order probability distribution. The distribution shows the probability mass function of the probability of the failure mode of interest [10]. With the proposed concept of default transfer functions, a system can be probabilistically analysed even when its (failure) behaviour has been only partially modelled by FTA.

In order to correctly deal with multimodality in quantitative analysis with BDD, we use the Boolean algebra with restricted variables by Caldarola [5], further developed in [24]: every mode is associated with a BDD variable, and the mutual exclusiveness of variables belonging to modes of the same aspect is accounted for by an adapted way of traversing and evaluating the BDD. The only prerequisite is that mutually exclusive variables have to be consecutive in the variable order.

### IV. EXEMPLARY MODEL & EXPERIMENT

This section describes the execution of an experiment intended to validate the modelling and analysis concepts presented so far, using a model from the automotive domain. We will demonstrate that the combination of Boolean concepts with multivalued variables, second-order probability analysis, and the successive replacement of default transfer functions with more specific ones in CLM lead to meaningful results that can be used to optimise the effort spent on FTA.

#### A. The traction control system

A traction control system (TCS) is used in modern cars to keep the vehicle stable and to ensure tyre grip by preventing the wheels from slipping. For this purpose, an ECU monitors wheel slip and reduces the throttle requested by the driver until both tyres have grip. In cars with a “sportive” marketing image, the slip is often not altogether prevented but allowed at low speed. In this way, drivers can still perform a traditional racing start.

Figure 6 (representing the same system as the conceptual view of Figure 1) shows the CLM of a simple but realistic TCS consisting of a throttle delimiter and an enabler. Please note that every source and target of an arrow symbolises just one service port, subdivided into several service modes and/or aspects; here, white and black subdivisions signify nominal/correct and erroneous modes, respectively. The delimiter receives the measured slip of the left and right driven wheels, the driver’s throttle request, and the measured car velocity, and passes the (potentially) delimited throttle on to the enabler. The enabler receives the delimited throttle, the requested throttle, and the car’s velocity and delivers either the delimited (car moves faster than 20 km/h) or the requested throttle (up to 20 km/h) to the motor controller, depending on the current speed of the car.

To prevent the driver from experiencing a sudden jump at 20 km/h, the enabler interpolates the provided throttle between requested and delimited throttle, starting somewhere below and ending somewhere above the threshold. In addition to their calculator and interpolator subcomponents, both enabler and

---

On the modelling side, lack of certainty is captured by facilitating the combination of different levels of detail by means of mode logic (Sections II.B, III.A). For quantitative analysis, alongside fixed values, probability intervals are assigned to service modes. In a first approximation, their distribution is assumed to be uniform, which is consistent with a least-knowledge approach (but any distribution is conceivable if there is a reason for it). If no statement at all is possible about the likelihood of a mode, even the complete probability range of [0, 1] can be used. If the interest lies within a certain range of resulting probabilities, for example, if a specific safety integrity level has to be attained, the subset of samples within that range can be taken as exemplars of possible system realisations. They typically differ considerably with respect to required-service and internal reliabilities. A sample subset may be subjected to further analysis, for example with a cost function, to find out how to achieve a given safety goal most efficiently [10].
throttle delimiter have a sanity checker that can detect erroneous output values and correct them before they are passed on.

The regarded aspect of the sanity check is veracity (Section II.B), with the modes OK, false positive (F+), and false negative (F–). We assume, for the sake of conciseness, that the sanity checker corrects wrong throttle values perfectly. Therefore, the nominal as well as the false positive mode of the internal sanity check always lead to a nominal output value (both are thus represented by white shapes). A false negative, however, together with an input value of the required throttle which is too high, leads to an output that is too high; similarly, a false negative with an input throttle value that is too low leads to an output value that is too low.

The interpolator (Figure 9) outputs a provided throttle whose value is too high when (a) the internal interpolation fails and yields a value that is too high, or (b) the incoming values of both the delimited and the requested throttle are too high, or (c) the value of the delimited throttle is too high and the requested throttle is correct or too high, or (d) the value of the delimited throttle is correct or too high and the value of the speed of the car (v_carRef) is too low (aspect: correctness, with mode: –) and the actual value of the v_carRef signal is between zero and 20 km/h, or (e) the value of v_carRef is out of range (else) (we leave out the other modes of the outgoing provided throttle because the following analysis will focus on the +-mode of the provided throttle).

B. Quantitative analysis by simulation
To test the viability and validity of the concept of incremental, any-time FTA with default transfer specifications and second-order probability simulation, service modes are assigned a
variable and a probability range from which their probability values are sampled for analysis. The distribution of samples for failure modes over intervals is log uniform; the probability of the remaining mode (usually the OK mode) is derived such that the probabilities of all modes of an aspect sum up to 1. The interval range of $[10^{-6}, 10^{-5}]$ is intended to mirror the probabilities of all modes of an aspect.

The experiment, calculating the second-order PMF for the “+” failure mode of the provided throttle (“too high”) of the TCS (mode A1 in Figure 6), consisted of four stages, with each stage using different transfer functions to be combined in order to yield the global function evaluated by Monte-Carlo simulation:

(a) in the first stage, all transfer functions had the default form;

(b) for the second stage, the default TS of the sanity checker and the interpolator of the enabler component were replaced by those depicted in Figures 8 and 9;

(c) in the third stage, the TF of the enabler subcomponents stayed the same as in the second stage, the default TS of the calculator was replaced by the one from Figure 7, and that of the throttle delimiter’s sanity checker assumed a form similar to the one in Figure 8, but modelled on the LOD of primary modes (merging the TF of the secondary modes “+” and “-”);

(d) eventually, in the fourth stage all TF were used in their most detailed forms as shown in Figures 7, 8 and 9.

Stage (c) demonstrates that transfer functions on lower, more generic LOD can actually be used to provide input to functions requiring a more detailed mode (in terms of object-oriented programming this is akin to downcasting an object of a class to a subclass) (Section III.B).
interval (histograms are normalised; x-axes, denoting sample probability, are logarithmic).

The probabilities of PMF (a) (which in linear presentation is a slim and slightly skewed bell curve) range (logarithmically) between -1.698 and -0.722, have a mean of -1.160, a standard deviation of 0.206, and a maximum (linear) y value of 0.008. PMF (b) ranges from -3.999 to -2.001, has a mean value of -2.999, a standard deviation of 0.408, and the same maximum y as (a). PMF (c) brings a significant change in range, which now lies between -9.818 and -4.965, has a mean value of -6.820, a standard deviation of 0.714, and a maximum y of 0.010. PMF (d) features almost the same range with -9.847 to -4.983; its mean is -7.031, with a standard deviation of 0.858 and a peak on the y-axis at 0.009. Note that we could determine the expectation of a function (=sample mean) by just feeding it with the expected values of every variable. Since, for subset analysis and reliability allocation, we are interested in the actual variable values producing a sample, this is not an option. Likewise, the function range cannot be determined using simple interval algebra, bypassing simulation, because Boolean functions with multivalued variables are generally nonmonotonous [23].

While the PMF of the first two stages of the experiment come nowhere near a failure probability satisfactory for the safety-critical TCS, (c) and (d) both hit ASIL ranges, and intervals of $[10^{-5}, 10^{-4}]$ (ASIL A) and $[10^{-3}, 10^{-2}]$ (ASIL D) are indicated in the diagrams. If the analysed FM had a safety goal of ASIL A, we see that (c) already would yield a sufficient number of samples in the sample subset for ASIL A, such that further detailing, as used in stage (d), of the FTA of the throttle delimiter’s sanity checker would not be necessary. If, however, the safety goal is ASIL D, then it makes sense to analyse the sanity checker in greater detail, as has been done in (d). In this way, the FTA effort is reduced to an optimal level. The ASIL-A- or D-related sample subset can now be analysed further to arrive at a better picture of the reliability of the system, for example, by applying cost functions to find the best reliability allocation to internal and required services (see [10] for examples of second-order analyses with varying probability intervals and for a detailed description of the application of cost functions to sample subsets).

V. RELATED WORK

The taxonomy proposed in [17] places the CLM approach in the failure logic modelling (FLM) class. We therefore briefly review selected FLM methods, restricting the selection to combinational techniques.

An early technique in this area was the fault propagation and transformation notation (FPTN) [9]. Wallace [22] describes it as “a bridge between” FMEA and FTA, enabling deductive (FTA-like) as well as inductive (FMEA-like) progress. FPTN models failure propagation as a (combinational) relationship between incoming and outgoing failures of a component but does not include the possibility of a failure originating from internal component failure.

The dynamic flowgraph methodology (DFM) [11] includes the notion of temporal transitions between system states and employs multivalued logic to model software and embedded systems. For analysis, the prime implicants of the system function are determined. A particularity with DFM is that it captures behavioural dynamics via annotation of fault tree variables with information about discrete time steps. Thus, a conjunction of mutually exclusive states may still be valid, as long as they bear different timestamps. Consistence rules for temporal dynamics are used to narrow down analysis results. DFM, however, lacks full compositionality.

CFT [16] were introduced to associate fault tree models with system components directly, by introducing FT in- and outports. Ports are connected according to the failure flow (communicated through data, energy, or matter) between system components. The CFT concept has led to the development of a number of variants and extensions, among them safety concept trees (SCT) [25] [6], safe component model (SCM) [7], as well as CLM. SCT formalise the notion of a safety concept for safety-critical systems and support its development and assessment, as well as the derivation of a safety argument for certification. SCM integrate functional and safety models of a system and focus on supporting distributed development by abstracting between component hierarchy levels.

The hierarchically performed hazard origination and propagation studies (HiP-HOPS) method [19] is component oriented as well and aims at partial automation of the FTA process by deriving fault trees from component design models, associating them with their corresponding modules in the system architecture. Recently [21], DFT-like [8] extensions to HiP-HOPS were proposed that would take it in the direction of DFM.

Wallace’s fault propagation and transformation calculus (FPTC) [22] takes FPTN further by requiring, like CLM, a complete analysis of the failure responses of a component in isolation, independent of a specific context. Accordingly, FPTC models are based on a complete architecture and improve the reuse of component models by requiring no global rework upon context changes. Applying fixpoint analysis/simulation, the system model is evaluated as a “token-passing network”. The result is, for every connection between components, a stable set of tokens that describes potential failures occurring on that connection. Cyclic dependencies can be handled.

The fault propagation and transformation analysis (FPTA) proposed in [12] builds on FPTC and aims at probabilistic analysis of failure behaviour. Tokens are annotated with the occurrence probability of their associated failure, and a component’s output token/failure probabilities are calculated from input and internal failure probabilities until the fixpoint is reached (which, as in FPTC, is guaranteed to exist). There is a caveat here, though: since FPTA calculations are strictly local, the model has difficulty handling common causes: when a component puts out tokens on different connections for the same combination of inputs, output probabilities are stochastically dependent. If such tokens (or their descendants, respectively) reconverge in a component further downstream, their shared ancestry has been “forgotten”, and resulting probabilities will be incorrect, typically not in a conservative way. Considering that
common causes are critical and just very common in real systems, the applicability of FPTA is limited.

With all of the described approaches, CLM shares some commonalities, but it also differs from them. Unlike FPTN and FPTC, CLM offers probabilistic analysis (in the vein of [22], describing FPTC as a token-passing network, the CLM could be labelled as a “function-passing” network, which is emphasised by the compositional way of building the BDD used for evaluation); unlike DFM, CLM is strictly component oriented; while HiP-HOPS and SCM have focussed on semi-automata failure tree generation and their hierarchical abstraction, respectively, CLM introduces the level-of-detail concept and features, like DFM, the inclusion of functional values for behaviour modelling.

VI. CONCLUSION & FUTURE WORK

Common problems in the early and continuous application of fault tree analysis in the development of software-controlled systems are the presence of uncertainty and the coexistence of information on different levels of detail. In this article, we have described and experimentally validated an approach to alleviating this situation. The concept enhances previous work by introducing a way to automatically produce a worst-case, combinational failure mode (including consideration of non-failure modes) by way of default failure specifications which can be continuously adapted as system design proceeds. It was demonstrated that CLM features any-time analysability, which facilitates the combination of safety concepts for safety-critical systems. Diploma thesis, TU Karlsruhe.

Analyses with second-order probabilities were realised in a dedicated tool [4], whose algorithms have now been extended to deal with multimodality. Since CLM includes modes for functional values, it is especially suited to being used for modelling software behaviour, and environment conditions or usage profiles can be accounted for in analyses.

At the moment, we are conducting trials to evaluate further ways of analysing and characterising sample subsets fulfilling a certain safety goal, besides the existing cost trade-offs. We are also looking at ways to combine our ideas with concepts of related techniques, such as SCM and DFM, and to produce a completely graphical CLM editor/analysers. With increased tool support, we expect that the component logic model and its analyses will be a powerful means for the dependability-oriented exploration and navigation of the design space of software-controlled systems, helping to build safer systems more efficiently.

ACKNOWLEDGEMENT

This work was funded by the German Federal Ministry of Education and Research (BMBF) in the context of the ViERforES project (01 IM08003).

The author thanks Lars Grunsk for his valuable feedback.

REFERENCES

[19] Papadopoulos, Y (1999). Hierarchically performed hazard origin and propagation studies. LNCS 1698, 139-152.