A Novel Approach Towards Model-Driven Reliability Analysis of Simulink Models

Padma Iyenghar[1], Stephan Wessels[1], Arne Noyer[1], Elke Pulvermueller[1], Clemens Westerkamp[2]
[1]Software Engineering Research Group, University of Osnabrueck, Germany
[2]Institute of Computer Engineering, University of Applied Sciences, Osnabrueck, Germany
{piyengha, stwessels, anoyer, elke.pulvermueller}@uni-osnabrueck.de, c.westerkamp@hs-osnabrueck.de

***Abstract*—In the recent decade, many formalisms and tools have emerged for model-driven Non-Functional Property (NFP) specification and assessment. In this direction, model-driven reliability and safety assessment of engineering systems developed using Matlab/Simulink is an emerging research challenge. However, a generic mechanism for NFP specification in the Simulink design model is not yet supported. Further, a completely automated model-based workflow for synthesis of a NFP analysis model and its subsequent analysis in a NFP analysis tool is missing. In this context, this paper proposes a novel approach and a fully automated workflow towards model-driven reliability analysis of Simulink models. An approach to annotate the Simulink design models with reliability attributes and subsequent synthesis of fault trees from the annotated design model is described. A prototype and initial experimental results are discussed.**

Keywords*—Reliability; Fault Tree Analysis (FTA); Matlab/Simulink; Embedded Software Engineering (ESE)

I. Introduction and Motivation

Model Driven Development (MDD) is a promising means for capturing key structural and behavioral requirements *before* implementing code. In addition, by changing the focus of software development from code to models, MDD also provides means to alleviate verification of Non-Functional Properties (NFPs) of the underlying software system such as timing, performance, reliability and safety. This leads to early detection of the potential problems in the underlying software design. Early avoidance or prevention of failures saves costs and improves the quality. Such a step is particularly beneficial for model-driven Embedded Software Engineering (ESE), as embedded software systems are often subject to critical NFPs. In this context, the Unified Modeling Language (UML) and Matlab/Simulink/Stateflow [1] can be considered as among the foremost of the commonly employed modeling languages/domains for MDD in ESE.

Let us consider a general approach for model-driven NFP analysis [2], as seen in Fig. 1. It comprises of the following steps (in brief): (a) add annotations to the design model to describe the NFPs (b) define model transformations from annotated software models to formalisms useful for NFP analysis (c) analyze NFPs and (d) assessment and feedback to the designers. An example for the above use case is the specification of the *failure occurrence probability* (in %) for the components in the Simulink design model, for reliability analysis (step (a) in Fig. 1). Fault Tree Analysis (FTA) is a well-known method for reliability and safety assessment of

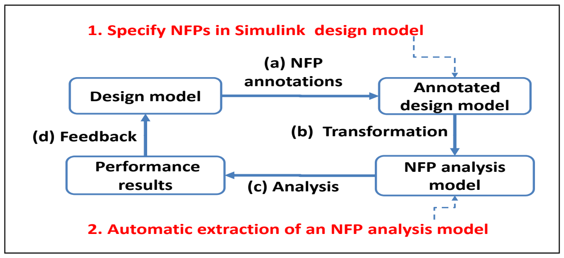


Fig. 1. A general approach for model-driven NFP analysis and challenges addressed (1, 2) in this paper

engineering systems [3]. In this context, the next steps (b)-(c) could deal with the synthesis of fault trees based on the annotated design model and subsequent analysis in a FTA tool. Finally, based on the assessment of the FTA results, feedback regarding the path(s) in the design model with higher failure occurrence may be provided (step (d) in Fig. 1). Such model-driven NFP analysis approaches are gaining inroads in ESE projects, for instance involving the UML [4]. Standardized, sophisticated and tailored extensions required for NFP-specific annotations (for step (a) in Fig. 1) are readily available in the UML domain. For instance, the UML profiles [5] provide such extensions to specify the NFPs.

On the other hand, automatic extraction of a NFP analysis model, based on the annotated design model with NFPs are still in the fledgling stages in the Simulink domain. A major contributing reason for this is the lack of such sophisticated, standardized mechanisms for annotation of the NFPs in the Simulink domain. In this context, the existing related work pertaining to synthesis of fault trees from Simulink models, in [6], [7], are neither completely automated nor fully model-driven. Further, a generic approach towards model-based annotation of the reliability attributes (e.g. *failure occurrence rate*) in the system design model (in Simulink), is not available. Thus, model-based specification of the reliability attributes in the Simulink design model and automatic extraction of a reliability analysis model, based on the annotated Simulink model (with NFPs) can be considered as emerging research challenges. These challenges are numbered as 1, 2 in Fig. 1.

In summary, a generic mechanism for model-based specification of NFPs (e.g. reliability attributes) in the blocks/subsystems in the Simulink design model has not been proposed so far. Further, an automated workflow (steps (a)-(d) in Fig. 1) for synthesis of fault trees and eventually performing a FTA in a reliability analysis tool is not yet available. These emerging challenges will be of interest to academia and also benefit the practitioners, esp. in the field of industrial automation, significantly. This paper aims to close the aforementioned gaps, with the following novel contributions:

(a) A generic mechanism for annotation of NFPs using *masks* in the Simulink design model (challenge 1 in Fig. 1).
(b) A generic workflow adhering to the general approach in Fig. 1, for model-driven synthesis of fault trees from the Simulink model (challenge 2 in Fig. 1) and subsequent analysis in a fault tree analysis tool (e.g. FaultCAT [8]).
(c) An experimental evaluation (of reliability analysis) using the proposed mechanism in a real-life, light weight ESE example implemented in Simulink.

The remaining of this paper is organized as follows. Next to this introduction, section II deals with related work. Section III and IV elaborate on the proposed approach and experimental evaluation respectively. Section V provides a conclusion.

II. Related work

A. NFP specification in UML and Matlab/Simulink

In the case of UML-based NFP analysis, the extensions required for NFP-specific annotations are defined as UML profiles. Some examples are using the Modeling and Analysis of Real-Time and Embedded Systems (MARTE) profile [5] for performance analysis [2], [9], using the MARTE-DAM profile for dependability analysis [10] and using a custom-defined profile for reliability analysis [11]. In [4], an example of constructing such analysis models, closely based on the semantics of the FTA tool FaultCAT [8] is presented.

In the Simulink domain, sophisticated standardized mechanisms such as profiles (as in UML) do not exist for annotation of the NFPs. For example, NFP (e.g. performance) analysis tools such as *TrueTime* and *Jitterbug* [12] are Matlab-based analysis tools which allow users to compute performance criterion for control loop under various timing constraints. However, direct support for NFP specification in Simulink (unlike UML) is lacking in the Simulink domain. To address this gap, this paper proposes a generic mechanism for specification of the NFPs (step (a) in Fig. 1) in the components (e.g. blocks/subsystems) of the Simulink design model. With the aid of this mechanism, the design model in Simulink can be annotated with reliability attributes (e.g. *failure occurrence probability*) which can be used for reliability analysis.

B. Model-Driven FTA of Simulink models

The *reliability* of a system is a measure of its ability to keep operating over time. Three key reliability measures are the *Mean Time to Failure (MTTF)*, *Mean Time Between Failures (MTBF)* and *failure rate* [10], [13]. An example to validate the proposed approach is presented in this paper based on the reliability attribute, *failure rate*. This attribute represents a probability that the component fails. It is also called "the rate of occurrence of failures".

A popular analysis technique for reliability, among others, is the FTA. It is a classical, standardized and well-established method applied to the safety and reliability analysis of systems [3]. It identifies the conditions that may cause or contribute to the occurrence of an undesired top event and represents them in a graphical form. As limitations, many researchers name the usually high manual efforts and experience which are needed to perform the FTA [14]. Automation in FTA can be viewed from different perspectives, as a series of steps, namely: (1) specification of reliability attributes in the design model, (2) automatic generation of fault trees from the (annotated) design model and (3) performing FTA in a FTA tool. As the manual activity of generating fault trees usually requires immense efforts, during the last years various approaches for steps (2), (3), i.e., generation of fault trees based on system models and subsequent FTA have been published [14], [15]. A method to automatically generate fault trees from Simulink models is proposed in [7]. Using hierarchical system structure and dependencies between components, their algorithm generates the system's fault trees. To identify failures, they use a form of computer termed *HAZOP* and examine each system element in detail. Another method to synthesise fault trees from Simulink models is proposed in [6]. The Simulink model is enhanced with failure behavior and a fault tree is automatically derived. Both [6], [7] deal with the steps (2) and (3) and not step (1) mentioned above. Their main aim being reducing the efforts to generate fault tree. However, for the method proposed in [7] the modeling and preparation efforts are very high. Similarly, the approach in [6] requires significant manual efforts during the analysis of dependencies of the models. In the approach proposed in this paper, a novel perspective about automating FTA covering all three steps mentioned above is presented.

III. Proposed approach

In this section, an integrated model-driven workflow for reliability analysis of Simulink models is proposed. The workflow, illustrated in Fig. 2, closely adheres to a general approach for model-driven NFP analysis, which was introduced in section I. The workflow consists of four main steps (steps 1-4 in Fig. 2). The individual steps involve one or more actions, denoted by steps (a)-(d) in Fig. 2. Each step and its corresponding activities in the proposed workflow are explained in detail below.

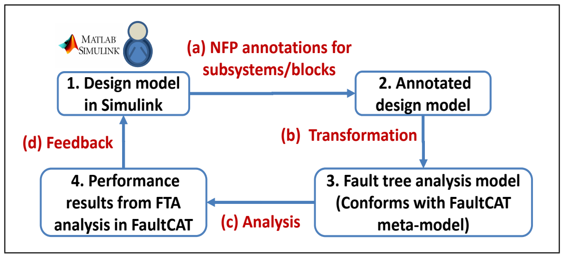

MATLAB SIMULINK
(a) NFP annotations for subsystems/blocks
1. Design model in Simulink
2. Annotated design model
(d) Feedback
(b) Transformation
4. Performance results from FTA analysis in FaultCAT
(c) Analysis
3. Fault tree analysis model (Conforms with FaultCAT meta-model)

Fig. 2. Model-driven workflow for reliability analysis of Simulink models

A. Specification of design models in Simulink (step 1 in Fig. 2)

Let us consider that the overall system topology of an engineering software system is implemented in the Simulink tool. Then, the following guidelines should be taken into consideration to employ the workflow.

(a) Related blocks which represent one functionality, should be kept together with the aid of a subsystem (a type of block which can comprise of a set of blocks).

(b) N distinct functions should be implemented using N distinct subsystems/blocks. This is because, in the proposed workflow, during the extraction of the NFP analysis model, the subsystems performing more than one distinct function are not considered as N separate components. For instance, when a component is implemented with two distinct functions A and B, the failure of such a component might be the result of the failure of either A or B or both. Therefore, also for the sake of simplicity, when N distinct functions are implemented using N distinct subsystems/blocks, the whole software system can be defined as a collection of single failure components.

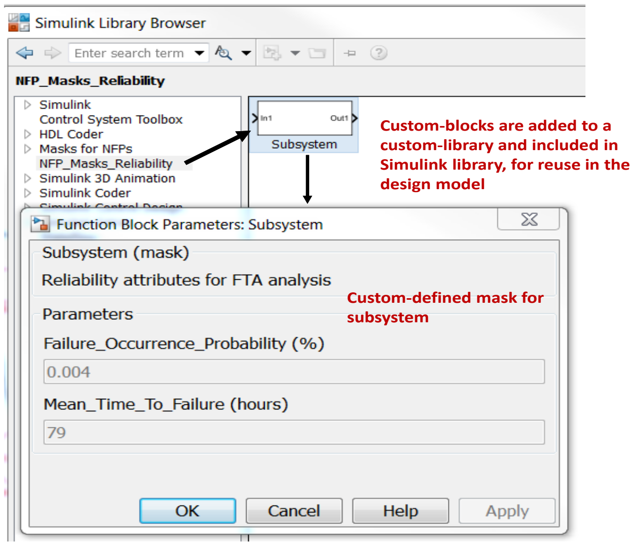
B. NFP annotations in the design model (step (a) and step 2 in Fig. 2)

In the proposed workflow for reliability analysis, the reliability attributes must be specified in the blocks/sub-systems in the design model. As seen in section II, sophisticated standardized extension mechanisms such as profiles (as in UML) do not exist for annotation of the NFPs in the Simulink domain. To address this gap, an existing feature in Simulink is used to propose a simple but effective mechanism (using masks) for the specification of the NFPs in the Simulink design model. This addresses the challenge 1 introduced in section I.

In the graphical block diagram tool in Simulink, a rich customizable set of block libraries is available. The blocks are the main elements available for the user to build models in Simulink. For instance, block libraries include continuous/discrete function blocks, ports, subsystems and so on. An advantage is that the blocks are customizable and can be added to a custom-defined block library, for reuse. Another advantage is the availability of customizable *masks* for blocks. A *mask* is a custom user interface for a block. By *masking* a block, one can encapsulate the block to have its own parameter dialog box (i.e., a user-defined interface) with its own customized appearance, block description and parameter prompts.

Making use of these two advantages/features in Simulink, the components (e.g. blocks/ subsystems) in the design model can be annotated with the NFPs using custom-defined masks. Such customized blocks can now be included as a user-defined custom library for reuse. This simple but effective methodology for NFP specification in the design model, which makes use of the existing features in Simulink, can be considered as a close equivalent of the extension mechanisms such as profiles available in UML.

In the proposed workflow, N distinct functions are considered to be implemented using N distinct blocks. Related blocks which represent one functionality are represented using a subsystem (refer section III-A). Thus, the software design model (from step (1) in Fig. 2) is defined as a collection of single failure components. Now, these components can be annotated with the NFPs (step (a) in Fig. 2), for instance the reliability attributes using custom-defined masks. These blocks can now be included as a user-defined custom library, for reuse. An example of a custom-defined mask which is created for a Simulink component is shown in Fig. 3. The reliability attributes for this component (a subsystem), which will be used for reliability analysis using the proposed workflow (Fig. 2), are now specified in the user-defined interface for the mask,



Custom-blocks are added to a custom-library and included in Simulink library, for reuse in the design model

Subsystem (mask)

Reliability attributes for FTA analysis

Custom-defined mask for subsystem

Parameters

Failure_Occurrence_Probability (%)

0.004

Mean_Time_To_Failure (hours)

79

Fig. 3. An example of a custom-defined mask for a subsystem

as shown in Fig. 3. In this case, an example of a mask to specify the *failure occurrence probability* and the *MTTF* of the subsystem is illustrated. To avoid repeated manual effort to create these masks for blocks, a custom-defined Simulink library is created and the blocks (now with special masks) are added to this Simulink library. Simulink supports the addition of such custom-defined libraries to its core Simulink libraries, thereby facilitating the reuse of the custom-defined blocks in the custom-defined libraries. As seen in Fig. 3, the block with customized masks is added to a custom library (*NFP_Masks_Reliability*) and included along with the core Simulink libraries. The newly added custom library can be included to appear along with the other core libraries in the Simulink library browser [1], thereby facilitating its (re-) use in the design model. Upon including this custom-defined subsystem in the design model to represent a functionality, the mask parameters can be edited to specify the required NFPs (in this case the *failure occurrence probability* and/or the *MTTF*).

C. Creation of a FTA model (step (b) and step 3 in Fig. 2)

In the steps 1 and 2 of the proposed workflow in Fig. 2, the design model is constructed and annotated with reliability attributes respectively. The annotated design model needs to be transformed to an analysis model in step 3, which can then be subject to a NFP analysis (in step 4 in Fig. 2). In the following, the activities involved in model-driven transformation of the annotated design model to a reliability analysis model (i.e., step 3 and step (b) in Fig. 2), which can be used for analysis in a reliability analysis tool (e.g. FaultCAT [8]) are described.

1) FaultCAT: A fault tree analysis tool: Among the several FTA tools available, FaultCAT [8] is selected for FTA in this paper. It is an open source fault tree creation and analysis tool which directly supports XML input/output. This feature enables straightforward means to import/export of fault tree data between the MDD tools and FaultCAT. On the other hand, there is no standard meta-model for fault trees defined in advance, which is required for example during model transformations. In this paper a fault tree meta-model is built which closely adheres with the semantics used by the FaultCAT tool.

2) A meta-model for fault trees: In a general approach towards NFP analysis discussed in section I (Fig. 1), the annotated design model is often converted directly to a tool-specific NFP analysis model. However, in the proposed approach in Fig. 2, the annotated design model is converted to an instance of an intermediate FTA meta model. This meta model closely adheres to both the semantics of the *reliability analysis method* under consideration (e.g. FTA in our case) and the *reliability analysis tool* used for prototype evaluation (i.e., FaultCAT). Having an intermediate fault tree analysis meta model implies that the annotated design model is converted to an instance of the fault tree meta model. Some advantages of this step are:

(a) The resulting FTA model can be used for analysis across several FTA tools.
(b) Coupling of reliability properties of two sub-design models based on their resulting instances of FTA models.
(c) Merging/coupling of related reliability NFPs of design models created based on completely different modeling domains (e.g. one each from Simulink and UML)

The meta-model in Fig. 4, created using the Eclipse Modeling Framework (EMF) [16], describes fault trees. From

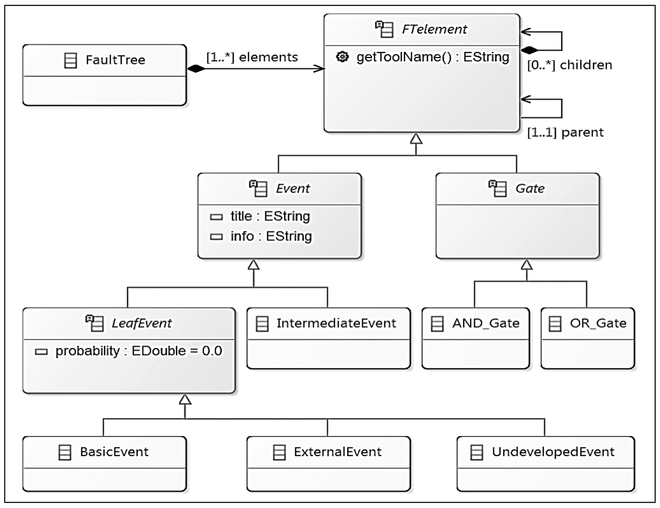


Fig. 4. FTA meta model closely based on semantics of FaultCAT tool

Fig. 4, it is seen that a fault tree model consists of elements named *FTelement*, which can be either *Events* or *Gates*. Both events and gates can have child elements. For instance, an event can have a child gate and a gate can have several children events. However, there are constraints such as a gate cannot have another child gate and an event cannot have more than one child gate. Every event has two attributes: *Title* and *Info* (which can be used to store extra useful information). Some specialized event types (*BasicEvent*, *UndevelopedEvent* and *ExternalEvent*) have a *Probability* attribute, which is the occurrence probability of the event (e.g. *failure rate*).

3) Model-to-model (M2M) transformation: Based on the annotated design model, a FTA model needs to be created. M2M transformations may be employed (implemented using a model transformation language, namely the Atlas Transformation Language (ATL) [17]) to convert the annotated design model to an instance of the fault tree meta model (i.e., annotated Simulink design model $\xrightarrow{M2M\_transformation}$ instance of fault tree meta-model). The resulting FTA model should conform with the meta model in Fig. 4

D. Reliability analysis in FaultCAT tool (step 4 and step (c) in Fig. 2)

Once the steps 1-3 are completed in the proposed workflow (Fig. 2), the annotated design model from Simulink is converted to a FTA model (e.g. with reliability attributes for analysis). This FTA model is exported automatically to the FTA tool under consideration for analysis. Based on the analysis results from the reliability analysis tool, feedback regarding the design model (e.g. the paths in the design model with higher failure occurrence) may be provided to the design model (step (d) in Fig. 2). In the proposed workflow, the only manual steps required for performing a model-driven FTA of a design model in Simulink in a FTA tool are:

(1) A one-time creation of a custom-defined mask for block/subsystem to specify the reliability attributes. Such custom-defined components (with masks) can be included in a custom library and included in the Simulink browser. Then, the components in this library can be (re-)used in the design model, similar to the blocks in any other existing Simulink library.
(2) Specification of the reliability attributes in the masks for FTA, for each single failure component (block/subsystem) in the design model in Simulink.

Apart from the aforementioned activities, the user has the advantage to work at a higher abstraction level and fully automate the reliability analysis of Simulink models using the proposed approach. This is a significant advantage for reliability analysis of large and/or industrially relevant examples, esp. during the early design stages.

IV. Experimental evaluation

Some of the requirements of an electric screwdriver are realised as a screwdriver software system using the MDD approach. The core functionalities of such a screwdriver system considered in the prototype are (a) decoding incoming commands for basic operations and (b) protection mechanisms indicating respective safety hazards (e.g. torque/overheat/undervoltage protection). An excerpt (of the main components) of the screwdriver software system implemented in Matlab/Simulink is shown in Fig. 5.

A. Design model in Simulink

The core functionalities of the screwdriver system considered in the prototype, namely items (a) and (b) mentioned above, are managed by a *ScrewDriverControlLogic* subsystem as seen in Fig. 5. This subsystem decodes the incoming commands for screwdriver operation(s) and the components implementing the respective functionality are invoked. In accordance with the guidelines mentioned in section III-A, for specification of the design model, each functionality envisaged for the screwdriver software system is implemented in one subsystem. For instance, the incoming commands are decoded for basic operations (such as mode right/left, stop) using the *ModeController* subsystem as seen in Fig. 5. Similarly, the torque/speed/overhead/undervoltage protection

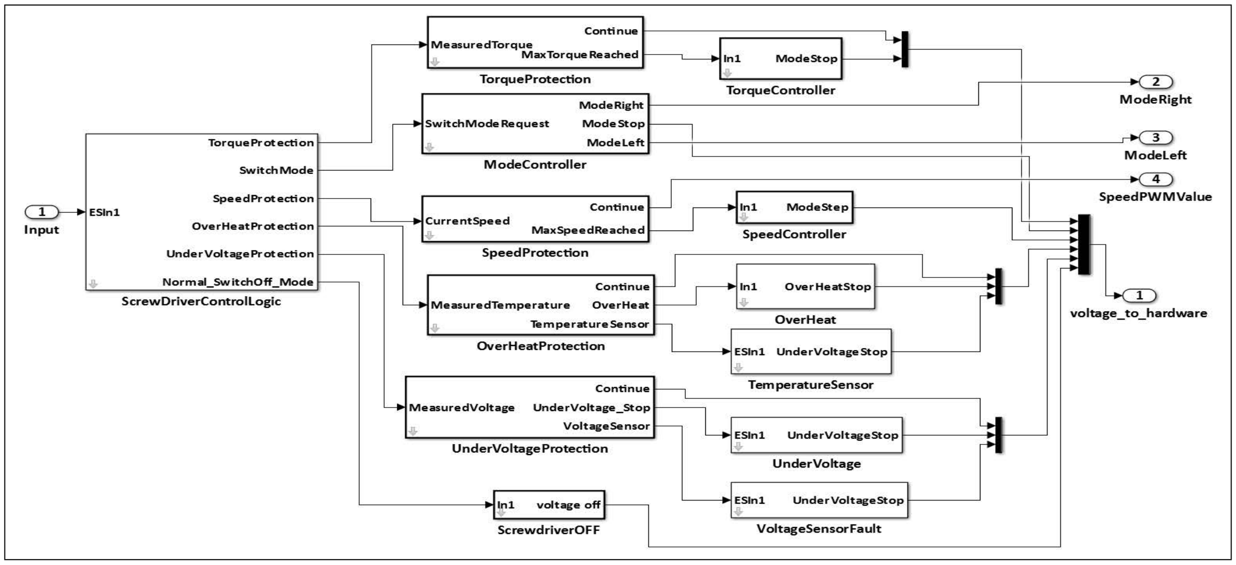
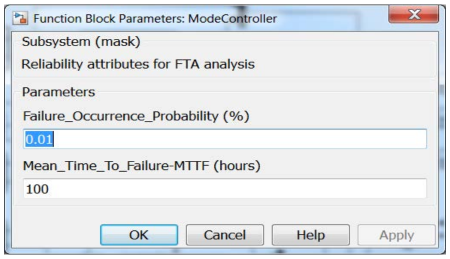


Fig. 5. An excerpt of the main components in the design model of the screwdriver software system implemented in Simulink

mechanisms are implemented in the subsystems named as *TorqueProtection*, *SpeedProtection*, *OverHeatProtection* and *UnderVoltageProtection* respectively, in the design model.

B. Specification of reliability attributes for the subsystems

To subject the design specification to a FTA, the design model needs to be annotated with quantitative reliability attributes (refer workflow in Fig. 2). In this section, examples pertaining to the specification of reliability attributes for the screwdriver software system described above are elaborated. The custom-defined mask available for subsystems from the library *NFP_Masks_Reliability* in Fig. 3 is used to specify the *failure occurrence probability* attribute in the design model. For example, specification of reliability attributes for the *ModeController* subsystem in the screwdriver design model in Fig. 5 is shown in Fig. 6. The next step is to create the reliability


Function Block Parameters: ModeController

Subsystem (mask)

Reliability attributes for FTA analysis

Parameters

Failure_Occurrence_Probability (%)

0.01

Mean_Time_To_Failure-MTTF (hours)

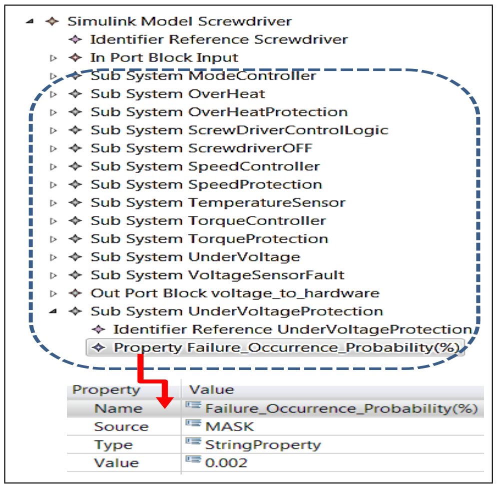
100

Fig. 6. Custom-defined mask to specify *failure occurrence probability* reliability attribute for the *ModeController* subsystem in Fig. 5

analysis model (step 3 and step (b) in Fig. 2) for the annotated design model (in Fig. 5) using model transformations.

C. Creation of FTA model

To subject the design model in Simulink to a model transformation, a straightforward way is to represent (i.e., to convert by model transformations) the design model in Simulink based on a standardized and structured data/meta model, such as the EMF [16]. In the prototype, an open source

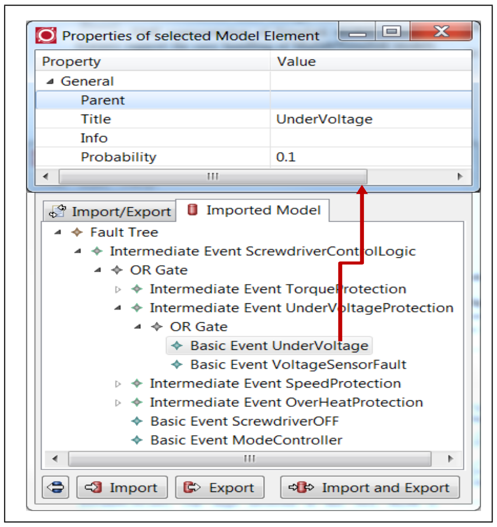

Simulink Model Screwdriver
Identifier Reference Screwdriver
In Port Block Input
Sub System ModeController
Sub System OverHeat
Sub System OverHeatProtection
Sub System ScrewDriverControlLogic
Sub System ScrewdriverOFF
Sub System SpeedController
Sub System SpeedProtection
Sub System TemperatureSensor
Sub System TorqueController
Sub System TorqueProtection
Sub System UnderVoltage
Sub System VoltageSensorFault
Out Port Block voltage_to_hardware
Sub System UnderVoltageProtection
Identifier Reference UnderVoltageProtection
Property Failure_Occurrence_Probability(%)

Property	Value
Name	Failure_Occurrence_Probability(%)
Source	MASK
Type	StringProperty
Value	0.002

Fig. 7. An excerpt of the EMF representation of the screwdriver design model in Simulink (shown in Fig. 5) using MASSIF [18]

Matlab Simulink Integration Framework for Eclipse (*MASSIF*) [18] is used for converting the annotated design model in Simulink to the EMF format. An advantage of this step is that, when the Simulink model is available in EMF format, the elements in the design model may be elementarily subject to further operations/transformations (e.g. EMF representation of Simulink design model $\xrightarrow{M2M\_transformations}$ instance of fault tree meta model). MASSIF's features support easy handling of Matlab/Simulink models by providing import/export capabilities to/from EMF. In the prototype, the design model of the screwdriver ESE example in Simulink is converted to its respective EMF representation as shown in Fig. 7.

1) M2M transformations: In the next step, M2M transformations are used to convert the EMF model which represents the annotated design model in Simulink (e.g. Fig. 7) to an instance of the fault tree representation (i.e., EMF representation of Simulink model $\xrightarrow{M2M\_using\_ATL}$ fault tree analysis model). The resulting FTA model conforms with the fault tree meta model in Fig. 4. In the prototype, the M2M is implemented in the model transformation language ATL. For the screwdriver ESE example, the input is the EMF representation of the screwdriver design model in Simulink, an excerpt of which is available in Fig. 7. The output of this transformation, is an instance of the FaultCAT meta model, as seen in Fig. 8. This comprises of a fault tree (Fig. 9) constructed based on the screwdriver design model annotated with reliability attributes (in Fig. 5).


Properties of selected Model Element

Property	Value
General	
Parent	
Title	UnderVoltage
Info	
Probability	0.1

Import/Export | Imported Model

- Fault Tree
  - Intermediate Event ScrewdriverControlLogic
    - OR Gate
      - Intermediate Event TorqueProtection
      - Intermediate Event UnderVoltageProtection
        - OR Gate
          - Basic Event UnderVoltage
          - Basic Event VoltageSensorFault
      - Intermediate Event SpeedProtection
      - Intermediate Event OverHeatProtection
      - Basic Event ScrewdriverOFF
      - Basic Event ModeController

Import | Export | Import and Export

Fig. 8. Instance of the FaultCAT meta model representing a fault tree for the screwdriver design model

D. FTA of the screwdriver system in FaultCAT

The FTA model available from the previous step (in Fig. 8), can now be subject to reliability analysis in the FaultCAT tool. In Fig. 9, the path that is most likely to cause a system

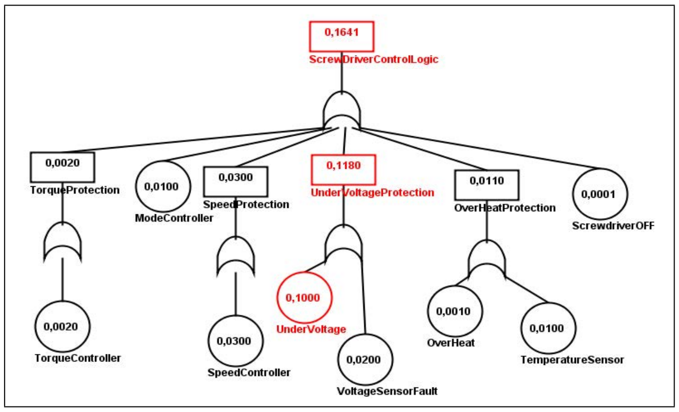


Fig. 9. FTA result in the FaultCAT tool, for the screwdriver system

failure is highlighted (in red color) by the FaultCAT tool. In this case, it is the *UnderVoltageProtection* path. This path is determined by the FTA tool, based on the annotated reliability NFP, namely *failure occurrence probability*, in the Simulink design model. These analysis results would provide an insight about the reliability aspects of the design model, based on which the design model may be subject to further refinement.

V. Conclusion

The proposed workflow which utilizes the salient features of the model-driven paradigm such as *abstraction* and *automation* bolstered by support for common data/model interchange format (i.e., the fault tree meta model) is a boon for projects requiring fully automated, completely model-driven workflow for reliability analysis. Further, the proposed workflow, in general, results in a tight integration of the MDD tools for software development and the reliability analysis tools, which is highly beneficial for industrial automation systems. A future direction is to apply the novel model-driven workflow for a significantly large, industrially relevant example, to prove its scalability.

Acknowledgment

This work is part of a project supported by a grant (id: KF2312004KM4) from BMWi-ZIM co-operation, Germany. This project work is carried out in close cooperation with Willert Software Tools GmbH and SymtaVision GmbH.

References

[1] Matlab and Simulink, http://www.mathworks.com/, 2016.
[2] D. C. Petriu, *Software Model-based Performance Analysis*. John Wiley & Sons, Inc., 2013, pp. 139–166.
[3] IEC 61025: International Standard for Fault Tree Analysis (FTA), https://webstore.iec.ch/publication/4311, 2016.
[4] Z. Zhao and D. C. Petriu, "UML Model to Fault Tree Model Transformation for Dependability Analysis," in *Proceedings of the International Conference on Computer and Information Science and technology*, Ottawa, Canada, 2015.
[5] The UML profile for Modeling And Analysis of Real-Time and Embedded Systems (MARTE), http://www.omgmarte.org/, 2016.
[6] F. Tajarrod and G. Latif-Shabgahi, "A novel methodology for synthesis of fault trees from matlab-simulink model," *World Academy of Science, Engineering and Technology*, vol. 41, pp. 630–636, 2008.
[7] Y. Papadopoulos and M. Maruhn, "Model-based synthesis of fault trees from matlab-simulink models," in *International Conference on Dependable Systems and Networks, 2001.*, July 2001, pp. 77–82.
[8] FaultCAT: An Open Source Fault Tree Creation and Analysis Tool, http://www.iu.hio.no/FaultCat/, 2016.
[9] A. Noyer, P. Iyenghar, E. Pulvermueller *et al.*, "A model-based workflow from specification until validation of timing requirements in embedded software systems," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2015, pp. 1–4.
[10] S. Bernardi, J. Merseguer, and D. C. Petriu, *Model-Driven Dependability Assessment of Software Systems*. Springer Inc, 2013.
[11] C. Lauer, R. German, and J. Pollmer, "Fault tree synthesis from uml models for reliability analysis at early design stages," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 1, pp. 1–8, Jan. 2011.
[12] A. Cervin, K.-E. Arzen, D. Henriksson, M. Lluesma *et al.*, "Control loop timing analysis using truetime and jitterbug," in *IEEE International Conference Computer Aided Control System Design*, 2006.
[13] J.-C. Laprie, *Dependability: Basic concepts and terminology*. Springer, 1992.
[14] M. Roth, M. Wolf, and U. Lindemann, "Integrated matrix-based fault tree generation and evaluation," *Procedia Computer Science*, vol. 44, pp. 599–608, 2015.
[15] A. Majdara and T. Wakabayashi, "Component-based modeling of systems for automated fault tree generation," *Reliability Engineering & System Safety*, vol. 94, no. 6, pp. 1076 – 1086, 2009.
[16] Eclipse Modeling Framework (EMF), https://eclipse.org/emf/, 2016.
[17] Atlas Transformation Language, https://eclipse.org/atl/, 2016.
[18] MATLAB Simulink Integration Framework for Eclipse, http://www.incquerylabs.com/, 2016.