Model-Based Co-Engineering and NFP Analysis in Embedded Software Sub-Systems Developed Using Heterogeneous Modeling domains

Padma Iyenghar[1], Arne Noyer[1], Joachim Engelhardt[2], Elke Pulvermueller [1]
1-Software Engineering Research Group, University of Osnabrueck, Germany
2-Institute for Distributed Systems, Ostfalia University of Applied Sciences, Germany
{piyengha, anoyer, elke.pulvermueller}@uos.de, jo.engelhardt@ostfalia.de

***Abstract*—In Embedded Software Engineering (ESE) scenarios involving heterogeneous modeling domains, the sub-systems may be developed using more than one modeling domain. In such scenarios, the need arises for co-engineering and model-based synchronization of Non-Functional Properties (NFPs) which are spread/linked across various modeling domains, before performing an NFP analysis. This paper proposes an integrated, generic workflow for model-based co-engineering and NFP analysis of embedded software systems developed by interdisciplinary teams, using heterogeneous modeling domains. Based on this workflow, a mechanism for model-based co-engineering and reliability analysis of an embedded software system, developed using Unified Modeling Language (UML) and Matlab/Simulink is discussed. An experimental evaluation of the proposed mechanism is presented.**

Keywords*—Embedded software; heterogenous domains; co-engineering; reliability analysis; UML; Matlab/Simulink;

I. Problem Statement and Motivation

Recent advancements in Embedded Software Engineering (ESE) indicate that ESE has progressed from classical software development approaches (e.g. V-model) to adopting new paradigms such as Model Driven Development (MDD). The idea behind MDD is to start with models (based on requirements) and proceed to their implementation via a set of successive model transformations. Ultimately, the implementation level source code is obtained using automatic code generation techniques. This implies that it is now easier to validate, correct and implement on different platforms, thereby enabling easy integration and inter-operability across different systems. Towards this direction, model-based performance analysis approaches are aimed at gaining insight into the quality aspects of the system.

For instance, an approach [1] for Non-Functional Properties (NFPs) analysis comprises of the following steps (in brief): (a) add annotations to the design model to describe NFPs (b) define model transformations from annotated software models to formalisms useful for NFP analysis (c) analyze NFPs and (d) give feedback to the designers. However, when the embedded software system involves interdisciplinary teams and heterogeneous modeling domains, the sub-systems may be developed using more than one modeling domain. In this case, the need arises for co-engineering and exchange/matching of NFPs which are spread/linked across various modeling domains and then perform the steps (c), (d) mentioned above.

Let us consider an embedded software system development involving interdisciplinary teams, each responsible for developing various sub-systems. For example, in such an ESE scenario, a software engineer/architect may be involved in the entire software architecture design using Unified Modeling Language (UML) (e.g. 90%), whereas a control engineer may be involved only in the controller design using Simulink (e.g. 10%). As seen in Fig. 1, some of the software components from the UML domain may be realized/implemented in the

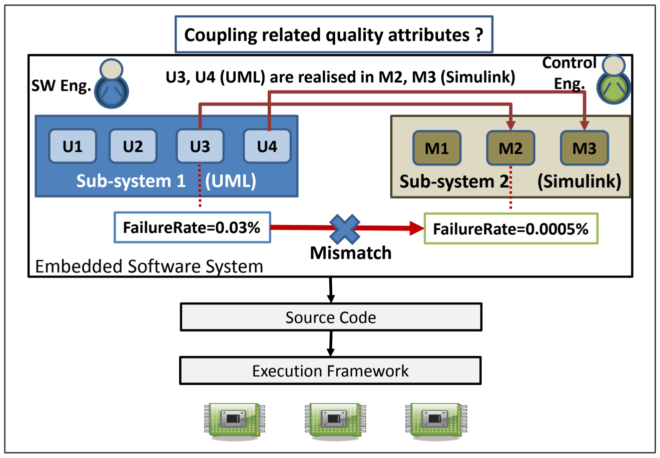


Fig. 1. Mismatch of related quality attribute among sub-systems

Simulink domain. For example, the software components U3 and U4 (UML) are implemented by M2 and M3 (Simulink) respectively (Fig. 1). The steps (a)-(d) mentioned above, may be applied for performing an NFP analysis for this scenario in Fig. 1. However, while applying such an approach for NFP analysis of the embedded software system as a whole, there arises a need to specify correct NFPs for components developed across different modeling domains. For instance, in the example in Fig. 1, the *FailureRate* reliability attribute of U3 (part of the overall design in UML), which is implemented in Simulink, is incorrectly specified in the UML domain. This may be due to the fact that the software engineer is ignorant of the specifics in the control engineering domain. It may also arise because of lack of appropriate communication of the quality attribute over several iterations of performance analysis among the interdisciplinary teams.

Thus, in ESE scenarios such as shown in Fig. 1, to perform NFP analysis (with correct values for NFPs) and integrate it with the design process, an appropriate workflow involving co-engineering among interdisciplinary teams must be in place. Enabling model-based co-engineering and synchronization of NFPs among components developed across heterogeneous modeling domains, would be a significant advantage for the design process as a whole. Whilst, to the best of our knowledge, a workflow for model-based co-engineering and exchange of quality attributes in ESE projects involving heterogenous modeling domains has not been proposed so far. This is an emerging research challenge in ESE, which will be of interest to academia and also benefit the practitioners significantly. This paper aims to close the aforementioned gaps, with the following novel contributions:

(1) A generic workflow for model-based co-engineering and NFP analysis in embedded software sub-systems developed by interdisciplinary teams using heterogeneous modeling domains.
(2) A mechanism for model-based exchange/synchronization of reliability attributes, during NFP analysis of embedded software systems, developed using UML and Matlab/Simulink (by applying the workflow).
(3) An experimental evaluation (of reliability analysis) using the proposed mechanism in a real-life, light weight screwdriver embedded software system example.

The remaining of this paper is organized as follows. Next to this introduction, section II elaborates on the related work. Section III introduces the generic workflow proposed in this paper. A mechanism for co-engineering and reliability analysis is discussed in section IV. An experimental evaluation is presented in section V. A summary is provided in section VI.

II. Related Work

There exists several proposals/schemes to classify the NFPs into distinct categories. For example, the NFPs are divided into three categories such as product, organisational and external requirements in [2]. The classification scheme adopted in [3] makes use of eight categories to classify the NFPs, namely: appearance, usability, performance, operational, legal, maintenance and support, security, cultural and political. The workflow proposed in this paper, in general, is targeted at the critical NFPs in the embedded software engineering domain such as the performance/timing and dependability (e.g. reliability, safety). However, please note that the proposed workflow can be applied, in general, for various categories of NFPs, for instance discussed in [2], [3].

Often, economical factors are preempted over the NFPs in the early design stages, when the overall system architecture is still subject to refinement [4]. This could be because of the fact that the NFP analysis methods are time consuming, not well integrated with the design process and/or involve less automation. The emphasis on models in MDD, facilitates the analysis of NFPs of the embedded software under development, based on its model [1]. This could give rise to less time consuming and more automated NFP analysis approaches. However, model-based assessment of NFPs is still an open research challenge, although considerable progress has been made and different approaches have been developed [5].

For instance, a comprehensive review on the research in the field of model-based performance prediction at software development time is available in [6]. The study states that, much detailed information from the implementation/execution scenarios is needed in order to carry out performance analysis. This is typically also applicable when the software architecture involves more than one modeling domain (e.g. UML, Simulink), which is one among the main goals addressed in the workflow proposed in this paper. More recently, a performance engineering approach for industrial embedded data processing systems is proposed in [7]. It demonstrates how light weight models can be used in the early stages of system development to estimate the influence of design changes on system performance. Similarly, experiences in conducting model-driven NFP analysis based on the steps (a)-(d) described in section I, is presented in [1]. However, the work presented in [1], [4], [6], [7] concentrate on NFP analysis and integration of its results in the early design stages, involving only one modeling domain. On the other hand, the workflow proposed in this paper concentrates on co-engineering and NFP analysis of embedded software sub-systems developed using heterogeneous modeling domains.

Some of the NFPs in ESE are performance (e.g. throughput, latency), schedulability, power usage, dependability and security. Dependability in turn encompasses several NFPs such as availability, reliability, safety, integrity and maintainability. NFP analysis approaches in ESE based on MDD, primarily employ UML [8] (and/or its related languages) as the underlying modeling domain. Another frequently used domain, in ESE, for control engineering is the Matlab/Simulink tool domain [9]. In the following, related work pertaining to NFP analysis approaches (e.g. reliability analysis) in ESE scenarios involving UML and Matlab/Simulink domain are discussed.

A class of successful approaches for NFP analysis, relies on building analysis models from system descriptions [1], i.e., based on steps (a), (b) described in section I. For instance, the UML design model is annotated with NFP requirements using UML profiles. Some examples are using MARTE profile [10] for performance analysis [1], using MARTE-DAM profile for dependability analysis [5] and using a custom-defined profile for reliability analysis [4]. In [11], an example of constructing such analysis models, closely based on the semantics of the fault tree analysis tool FaultCAT [12], using model transformations for reliability analysis is presented (i.e., step (b)). In the Simulink domain, sophisticated profiles do not exist for annotation of the NFPs. For example, a variety of construction techniques for synthesis of fault trees, intended for reliability analysis of Simulink models is mentioned in [13]. Some fault tree synthesis techniques for Simulink models include, construction of fault trees based on failure functions and schematic diagrams of system components [13], [14].

From the above, it is clear that a workflow for model-based co-engineering and synchronization of the NFPs, which are spread/linked across different modeling domains is missing.

III. A workflow for NFP evaluation

In this section, an integrated model-based workflow (Fig. 2) for co-engineering and NFP analysis of embedded software sub-systems developed using heterogeneous modeling domains is outlined. The proposed workflow, closely adheres to a general approach for model-based NFP analysis. It follows the

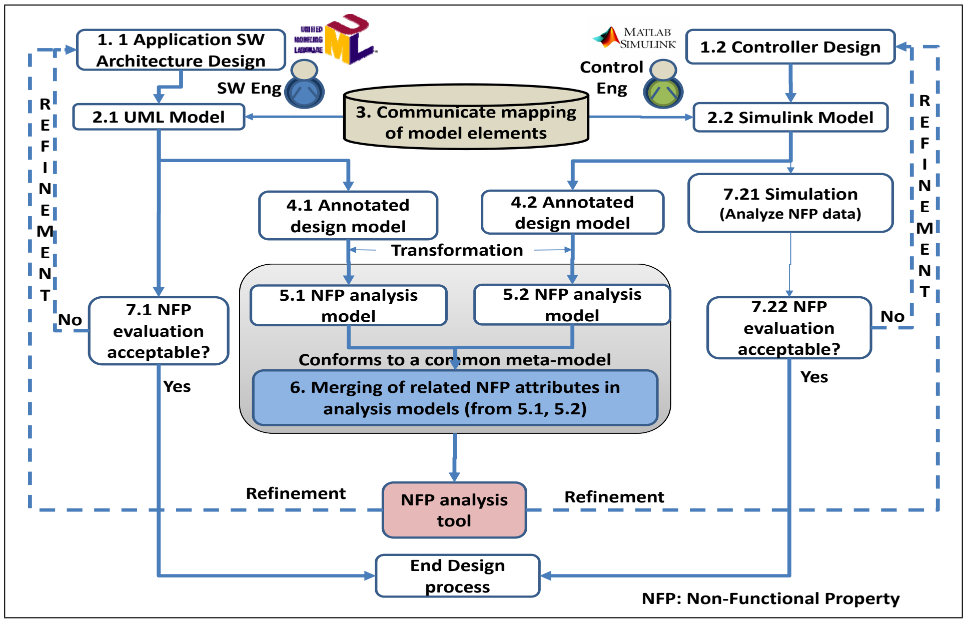

1. 1 Application SW Architecture Design
SW Eng
1.2 Controller Design
MATLAB SIMULINK
Control Eng
2.1 UML Model
3. Communicate mapping of model elements
2.2 Simulink Model
4.1 Annotated design model
4.2 Annotated design model
7.21 Simulation (Analyze NFP data)
Transformation
5.1 NFP analysis model
5.2 NFP analysis model
7.1 NFP evaluation acceptable?
7.22 NFP evaluation acceptable?
No
Yes
Conforms to a common meta-model
6. Merging of related NFP attributes in analysis models (from 5.1, 5.2)
REFINEMENT
Refinement
NFP analysis tool
End Design process
NFP: Non-Functional Property

Fig. 2. A Generic workflow for NFP Analysis

steps (a)-(d) mentioned in section I for NFP analysis. However, the proposed workflow is aimed at performing quality analysis and its integration with the design process, involving interdisciplinary teams and heterogeneous modeling domains. The main challenge in such a scenario, is to enable model-based co-engineering and synchronization of NFP attributes, which are spread/linked across different modeling domains. Following are some of the aspects considered in the workflow (in Fig. 2), to address the aforementioned challenge:

(A) It takes into consideration an ESE scenario, wherein an embedded software system is developed using heterogeneous modeling domains such as UML and Simulink. These two domains are chosen as examples for co-engineering in this workflow, as they are among the two most commonly used modeling domains in ESE projects.
(B) The workflow considers that (a) the overall software system architecture design is created in the UML domain and (b) a majority of the software components are developed in the UML domain (e.g. 70%–90%) and only the components involving control engineering tasks are implemented in Simulink (e.g. 10%–30%). This is one among the emerging industry practices, in the field of ESE involving both UML and Simulink [4] [15].

The basic idea of the workflow is to retain the separation of concerns but only exchange the relevant, related quality attributes which are necessary to perform an NFP analysis. As seen in Fig. 2, the workflow comprises of two main (vertical) design processes. Steps 1.1 to 7.1 deals with the design of the overall application software architecture (in UML). This overall design in UML may have one or more components requiring implementation in the control engineering domain. Steps 1.2 to 7.22, in Fig. 2, deal with the design of the controller components in Simulink. This controller design may not only implement the components of the overall software architecture (required from step 1.1); but, it could also deal with the integration of this controller (required in the UML domain) with several other hardware aspects for the entire system under consideration. Thereby, the NFP attribute of the controller component (required in the UML domain) may depend on further components modeled in Simulink (which are not part of the software architecture design in UML).

Thus, during the design process, assumptions regarding the quality attributes in the steps 1.1, 2.1 without the knowledge/involvement/inputs about the quality attributes from the Simulink domain may yield incorrect/misleading NFP analysis results. Therefore, during the design process itself, while performing an NFP analysis for the overall software architecture, it is imperative to include correct inputs regarding NFP attributes from the control engineering domain. This is especially true for the components included in overall design in UML (e.g. U2, U3 in Fig. 1), but realised in the control engineering domain. When this methodology is dealt with as an iterative NFP analysis and estimation process, involving refinements of the NFPs, one may avoid misleading performance results during the software architecture evaluation as a whole. Addressing the above challenge, a model-driven mechanism to synchronize/exchange the relevant and related NFPs (among UML and Simulink domains) which are needed to perform a performance analysis is illustrated in Fig. 2.

In this workflow (Fig. 2), an initial design model is created based on the requirements as seen in steps 2.1 and 2.2, for UML and Simulink domains respectively.

In step 3, both the teams involve in an one-time communication for exchanging information about the related model elements. An example is the exchange of the names and/or initial NFPs of the components (e.g. M1="MotionController") in the Simulink domain, which implements the respective control software component for the UML domain (e.g. U2="TorqueProtection").

In steps 4.1 and 4.2 the design model is annotated with NFP attributes. Using model-to-model (M2M) transformation, the annotated design model (from steps 4.1 and 4.2) is converted to a NFP analysis model (in steps 5.1 and 5.2 respectively). This NFP analysis model, conforms with a NFP analysis meta-model, which in turn adheres closely with the formalisms/semantics of the NFP analysis tool.

An NFP analysis would be performed based on the resulting NFP analysis model in step 5.1 or 5.2 (e.g. steps (a)-(d) in section I, [1]), if only one modeling domain was involved. Since more than one modeling domain is involved here, before performing an NFP analysis, the workflow in Fig. 2, introduces a step for model-based co-engineering and synchronization of NFPs of software components, which are linked across the two domains.

In step 6 in Fig. 2, synchronization of the related performance attributes in the NFP analysis models (created in steps 5.1 and 5.2) is carried out. A mechanism for step 6 is described in detail in section IV, thus not dealt with in detail here. Note that, once the related NFP attributes are merged in step 6, an NFP analysis can be carried out.

An initial iteration of the NFP evaluation can be considered to be complete, once the NFP analysis results are fed-back for refinement of the design (indicated by dotted arrows in Fig. 2). In the Simulink domain, based on these NFP analysis estimates, further simulations can be carried out to arrive at an acceptable value for NFP attributes for all the components designed in the control engineering domain (step 7.21). If the results are acceptable (steps 7.1 and 7.22), one can conclude the design process. If not, several iterations of these steps in 4.1, 5.1, 6, 7.1 (UML domain) and steps 4.2, 5.2, 6, 7.21, 7.22 (Simulink domain) can be carried out, until an acceptable evaluation of the design is obtained. Note that step 3 needs to be carried out in the subsequent iterations only if there are changes to the overall design, such as addition of new components and/or changing of names of components.

IV. A Mechanism for model-based coupling of reliability attributes

Based on the workflow proposed in the previous section, a mechanism for model-based synchronization of reliability attributes, which are spread across different modeling domains is discussed in this section. In the following, a brief introduction about reliability and analysis of reliability attributes using fault trees is presented. This is aimed at providing a background on reliability as a NFP and its analysis. Next, a mechanism for synchronizing the related reliability attributes, before performing a reliability analysis, is discussed.

A. Reliability: An NFP attribute and its Analysis

Dependability is that property of a computer system such that reliance can justifiably be placed on the service it delivers [2], [3]. Dependability has several attributes, including *availability*, *reliability*, *safety*, *security (confidentiality and integrity)* and *maintainability*. The *reliability* of a system is a measure of the ability of a system to keep operating over time. Three key reliability measures are the *Mean Time to Failure (MTTF)*, *Mean Time Between Failures (MTBF)* and *failure rate* [5]. An example to validate the proposed approach and a mechanism for coupling NFP attribute(s) spread across different modeling domains is presented in this paper based on the reliability attribute, *failure rate*. The reliability attribute *failure rate* represents the rate of occurrence of failures.

A popular analysis technique for reliability, among others, is the Fault Tree Analysis (FTA). Some basics regarding fault trees and FTA is provided below. This would serve as a background for understanding the proposed mechanism for model-based coupling of the *failure rate* reliability attribute spread across heterogeneous modeling domains, before performing a reliability analysis.

1) Fault Tree: Fault trees are a graphical means to represent causalities for possible system hazards resulting from system component failures. These trees are often used to see what different parts of a system can contribute to make a component fail. The top event of the fault tree is the undesired event that can happen, whilst the different paths below it are a combination of events and gates. The events (e.g. intermediate event, basic event)) have a probability (e.g. failure rate) of inducing the undesired event and gates serve to permit or inhibit the passage of fault logic up the tree.

2) FaultCAT: A fault tree analysis tool: The fault trees are often used to find all credible ways of how an undesired event can occur, using the FTA methodology. Among the several FTA tools available, FaultCAT 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, as there is no standard meta-model for fault trees defined in advance, in this paper a fault tree meta-model is built which closely adheres with the semantics used by the FaultCAT tool.

The meta-model in Fig. 3, created using Eclipse Modeling Framework (EMF) [16], describes fault trees, which are fully supported by the FaultCAT tool used in this work. From Fig.

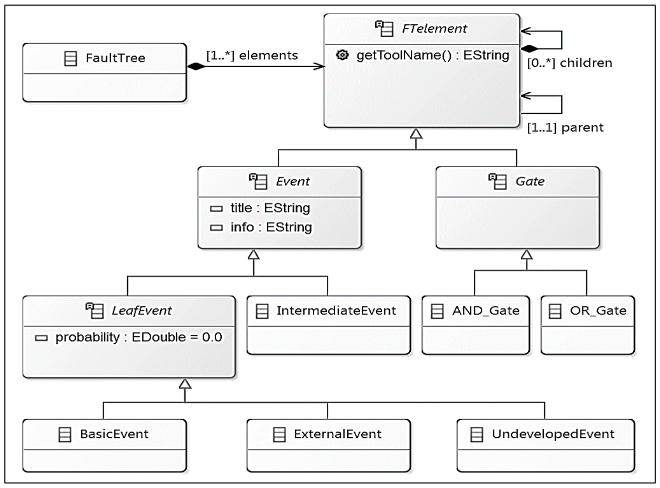
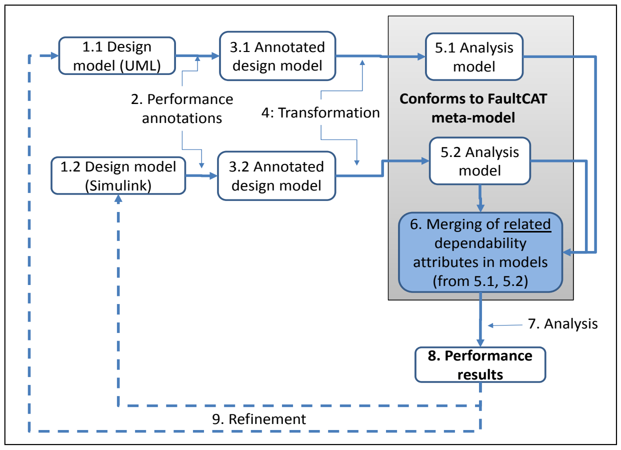


Fig. 3. Fault tree analysis meta-model

3, 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 events types (*BasicEvent*, *UndevelopedEvent* and *ExternalEvent*) have a *Probability* attribute, which is the occurrence probability of the event (e.g. *failure rate*).

B. A Mechanism for coupling reliability attributes



1.1 Design model (UML)
3.1 Annotated design model
5.1 Analysis model
2. Performance annotations
4: Transformation
Conforms to FaultCAT meta-model
1.2 Design model (Simulink)
3.2 Annotated design model
5.2 Analysis model
6. Merging of related dependability attributes in models (from 5.1, 5.2)
7. Analysis
8. Performance results
9. Refinement

Fig. 4. A mechanism for coupling reliability attributes from UML and Simulink domain, before performing a reliability analysis

As seen in Fig. 4, based on the system design model (steps 1.1 and 1.2 in UML and Simulink domains respectively), the first steps towards NFP evaluation involve:

(a) Step 2: Adding annotations to the design model to describe the NFPs, thereby arriving at an annotated design model. This is indicated by steps 3.1 and 3.2 in UML and Simulink domains respectively, in Fig. 4.
(b) Step 4: Defining model transformations from annotated software models to formalisms useful for NFP analysis.

Note that the possibilities to annotate design model (in UML and Simulink) with reliability attributes (e.g. *failure rate*) is described with the aid of examples in section V. Similarly, examples of model transformations creating the reliability analysis model (target) based on the annotated design model (source) are also described in section V. Thus, these aspects are not dealt with in detail in this section.

Let us consider that the reliability analysis model (based on the annotated design model) from both UML and Simulink domains is available in steps 5.1 and 5.2 (in Fig. 4) respectively. These models conform with the FaultCAT meta-model shown in Fig. 3. In line with our proposed workflow, the next step, namely step 6 in Fig. 4 must be carried out before performing a reliability analysis of the system (i.e., step 7 in Fig. 4). This is required to ensure that there is no mismatch of reliability attributes of components which are implemented across different modeling domains (i.e., UML and Simulink).

An example scenario illustrated in Fig. 5 is used to describe the coupling mechanism in step 6 (in Fig. 4). The Fig. 5 comprises of two parts, Fig. 5–(A) and Fig. 5–(B), which indicate the state of attributes of the events in the FaultCAT analysis model, before and after performing step 6 in Fig. 4 respectively. In Fig. 5, the boxes on the left side represent the events in the FaultCAT analysis model from the UML domain (i.e., available at step 5.1 in Fig. 4). Similarly, the boxes on the right side represent the events in the FaultCAT analysis model from Simulink domain (i.e., available at step 5.2 in Fig. 4). The basic events shown in Fig. 5, comprise of three attributes such as the *title*, a *probability attribute* and an *info* attribute. Note that the analysis models themselves conform with the FaultCAT meta-model (shown in Fig. 3).

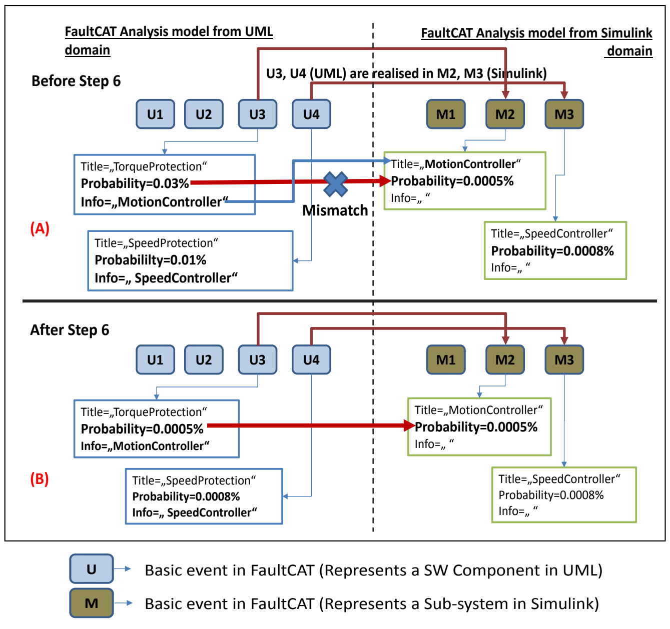


Fig. 5. Before and after performing step 6 (in Fig. 4) for coupling reliability metric

In the scenario in Fig. 5, the software components U3 and U4 (UML) are implemented by M2 and M3 (Simulink) respectively. Hence, the NFP attributes of these related software components must be in sync with each other, in order to perform a valid overall NFP analysis. In order to enable model-based coupling/synchronization of the NFP attributes, for example the *failure rate* in this case, a simple but effective mechanism described below can be used.

As mentioned in the description of the FaultCAT meta-model, the *info* field of the events in the fault tree, may be used to store additional useful information. In our scenario for coupling, the events in the UML domain can make use of this *info* field to store the names of their corresponding software components in the Simulink domain. This information can be specified along with performance annotations during step 2 in Fig. 4 and mapped to the *info* field of events in the fault tree (during step 4 in Fig. 4). Note that the software (UML) and control engineers (Simulink) can exchange such information (i.e., names of mapped/related elements) during step 3 in the overall workflow.

For example, the event U3 has the following attributes in the fault tree: *Title=TorqueProtection*, *Probability=0.03%* and *info="MotionController"*. Similarly the event M2 has the following attributes: *Title=MotionController*, *Probability=0.0005%*. It is clear that the *info* field of the event U3 (UML) corresponds with the name of the event M2 (Simulink). However, their *probability* attribute, which denotes the *failure rate* reliability metric used for FTA, does not match. This mismatch is indicated by a cross mark in Fig. 5–(A).

Now that the two analysis models (one each from UML, Simulink domain) are available, along with the aforementioned mapping information (using the *info* field), a simple match/merge transformation (e.g. using Atlast Transformation Language (ATL)) can be carried out to synchronize the *probability* attribute of both these related elements (from UML and Simulink domain). The values of the attributes of the events in the analysis model after performing a synchronizing operation (step 6 in Fig. 4) is shown in Fig. 5–(B). From Fig. 5–(B), it is seen that the *probability* attribute of the element U3 is now synchronized with the *probability* attribute of the element M2.

Thus, before performing a NFP analysis, in this case a reliability analysis using FTA based on *failure rate* for the software components, the values of the NFP attributes which are spread across/implemented using various modeling domains may be synchronized. This can be carried out using the simple but effective mechanism described in this section. In the following section, an experimental evaluation of the proposed mechanism, in a real-life ESE example is presented.

V. Experimental evaluation

A workflow for model-based synchronization of related NFP (e.g. *failure rate*, reliability attribute) was described in the previous sections. In this section, practical examples involving annotation of the design model (i.e., step 2 in Fig. 4) with the reliability attribute (e.g. *failure rate*) for both the UML and the Simulink domains is described. Similarly, the transformations used to convert the annotated design model to analysis models (which conforms with FaultCAT meta-model in Fig. 3) is described with examples. Matching of the *info* attribute in the events in the analysis models (from UML and Simulink) for synchronizing the *probability* attribute (which denotes *failure rate*), are described with examples. These examples are based on an implementation of a practical (real-life), light-weight screwdriver embedded software system scenario.

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). While the overall embedded software architecture (and the control-logic) is designed in the UML domain (using Papyrus [17]), the required control loops are implemented in Matlab/Simulink [9].

A. Sub-System modeled using UML

The sub-system in UML (Fig. 6) comprises of several classes such as *overheat/undervoltage protection*, *torque protection*, *controller*, *bit protection* and *button set*. For instance, the *controller* class is responsible for decoding the incoming commands for basic screwdriver operations. The *Overheat/Undervoltage/Torque protection* classes implement the overheat/undervoltage/torque protection mechanisms respectively. The controller components (in Simulink), handle the queries for parameters received from the UML domain. For example, upon requirement, an invocation of a controller-step function from the UML domain (e.g. from *torque protection* class) results in communication with the controller to determine if *maximum torque* value (of the motor) is exceeded.

1) Annotation of reliability attributes in UML: To derive a reliability analysis model, the UML design specification needs to be annotated with quantitative reliability attributes (i.e., steps 2, 3 in Fig. 4). An existing profile Dependability Analysis Model (DAM) [5], is used in this example. The usage of the profile consists in stereotyping model elements and assigning tagged values to the stereotype attributes of the stereotyped elements. Note that, one can also use a custom-defined profile tailored for specific needs (as in [4]) to specify the NFRs.

In Fig. 6, the classes are stereotyped with ≪*DaComponent*≫ from the DAM profile. This stereotype has an attribute *failure* of type *DaFailure*. The *DaFailure* complex type allows users to specify reliability attributes such as the *MTTF*, *MTBF* and *occurenceProb*. In our example, the *failure rate* of the software component is specified in *occurenceProb* as a probability value (in %). Similarly, the name of the corresponding software component in the Simulink domain is specified in the *description* field of the *DaFailure* attribute of the *DaComponent* stereotype, for a class in Fig. 6.

2) Creation of analysis models (step 4 in Fig. 4): The annotated design model in UML (Fig. 6) needs to be transformed to a fault tree analysis model (step 4, 5 in Fig. 4). This transformation is implemented using ATL. The input of this transformation is the annotated UML design model (Fig. 6). The output of this transformation is the respective fault tree analysis model. The resulting fault tree analysis model conforms with its respective fault tree meta-model in Fig. 3. Based on the meta-models for both UML [18] and fault trees (Fig. 3), a mapping from the source (i.e., the class diagram with DAM annotations) to the target (fault tree meta-model) created during ATL transformation is as shown in Table I. The intermediate fault tree analysis model created based

TABLE I. Mapping from UML (source) to fault tree meta-model (target)

Annotated design model in UML	**Fault tree meta-model**
Model	Intermediate event (top event)
≪*DaComponent*≫Class	Intermediate event
DaFailure	Basic event
occurenceProb in *DaFailure*	*Probability* of event
description in *DaFailure*	*Info* field of event

on the annotated UML design model (Fig. 6) is shown in Fig. 7. As seen in Fig. 7, the *TorqueProtection* intermediate event, has a basic event named *MotionController*. Note that the *TorqueProtection* class was specified with a *DaFailure* attribute, with description as *MotionController*. This indicates that a failure event corresponding to the *TorqueProtection* class is available in *MotionController* software component, which in our case is implemented in the Simulink domain.

B. Sub-system modeled using Simulink

A sub-set of the controller implementations in Simulink, comprising of the *MotionController* and the *SpeedController* is shown in Fig. 8. In the Simulink domain, specification of NFRs is not automatically supported (e.g. using profiles). In the prototype custom-defined masks are created for sub-systems in the Simulink domain to annotate the Simulink design model with the NFPs. For example, a custom mask created for the *MotionController* sub-system in Simulink is shown in Fig.

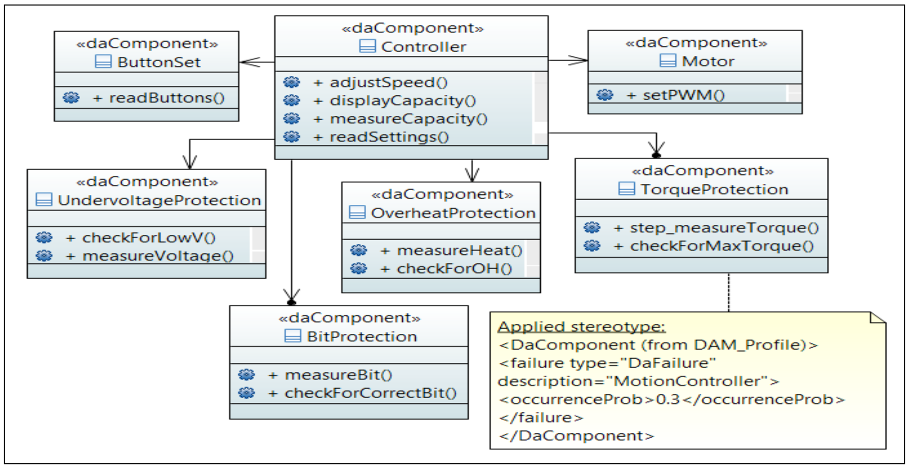
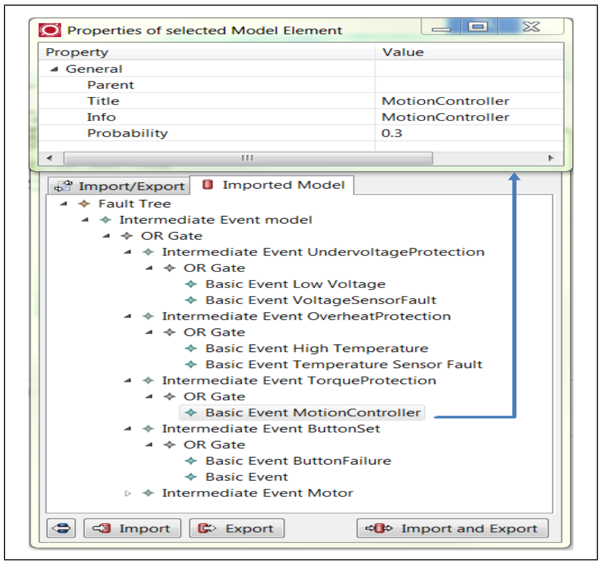


Fig. 6. Screwdriver software system: Annotated class diagram view in UML


Properties of selected Model Element

Property	Value
General	
Parent	
Title	MotionController
Info	MotionController
Probability	0.3

Import/Export | Imported Model

```
Fault Tree
  Intermediate Event model
    OR Gate
      Intermediate Event UndervoltageProtection
        OR Gate
          Basic Event Low Voltage
          Basic Event VoltageSensorFault
      Intermediate Event OverheatProtection
        OR Gate
          Basic Event High Temperature
          Basic Event Temperature Sensor Fault
      Intermediate Event TorqueProtection
        OR Gate
          Basic Event MotionController
      Intermediate Event ButtonSet
        OR Gate
          Basic Event ButtonFailure
          Basic Event
      Intermediate Event Motor
```

Fig. 7. Intermediate reliability analysis model for design model in Fig. 6

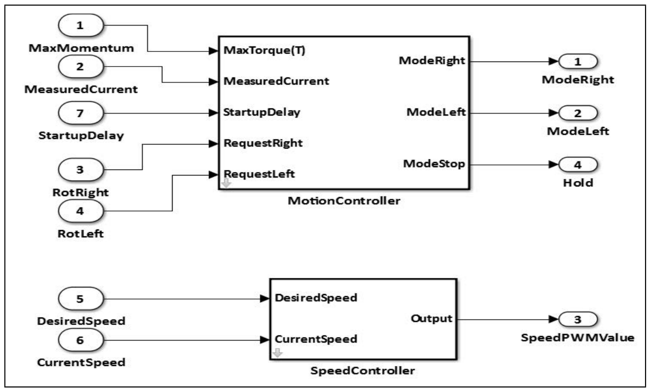
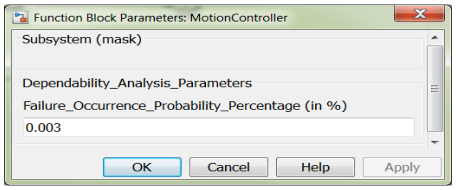


Fig. 8. Screwdriver software system: controller implementation in Simulink

9. The *Failure_Occurance_Probability* for the corresponding sub-system in Simulink can be specified (0.003%) as shown in Fig. 9. With this, the steps 2 and 3 in Fig. 4 is complete for the Simulink design model. The next step is to create the reliability analysis model (step 5.2 in Fig. 4) for the annotated


Function Block Parameters: MotionController

Subsystem (mask)

Dependability_Analysis_Parameters

Failure_Occurrence_Probability_Percentage (in %)

0.003

Fig. 9. Custom-defined mask to specify *failure rate* reliability attribute

design model (in Fig. 8) using model transformations. The resulting reliability analysis model should conform with the fault tree meta model in Fig. 3. In the prototype, a Matlab Simulink Integration Framework for Eclipse (MASSIF) [19] is used for converting the annotated Simulink design model, to EMF format, so that it may be elementarily subject to further operations/transformations. In the next step, M2M transformations, implemented in ATL, are used to convert this EMF model (representing the annotated design model from Simulink) to an instance of the fault tree representation. The resulting FTA model conforms with the meta-model in Fig. 3.

C. Synchronization of probability (failure rate) attribute

In our fault tree examples from both UML and Simulink domain, the *failure rate* of the *TorqueProtection* class in UML is 0.3 % (Fig. 7), whereas its corresponding implementation in Simulink, the *MotionController* sub-system has a *failure rate* of 0.003% (specified in Fig. 9). Thus, a need arises to synchronize the reliability attribute of the software component in UML, i.e., the *failure rate* of the *TorqueProtection* event from UML with the correct *failure rate* of the *MotionController* sub-system from Simulink (step 6 in Fig. 4). In the prototype, this synchronization of *failure rate* attribute among the fault tree events from both the domains, is also carried out using simple ATL transformations. For instance, an ATL rule (*syncProbability*) takes as input the basic events in the fault tree from the UML domain. For every basic event (as input) based on the value of its *info* field, it checks for the corresponding matching name (of an event) in Simulink domain. Once a match is found, the value of *probability* attribute for the corresponding event in Simulink is copied to its respective event in UML (e.g. as in Fig. 5). The resulting FTA model,

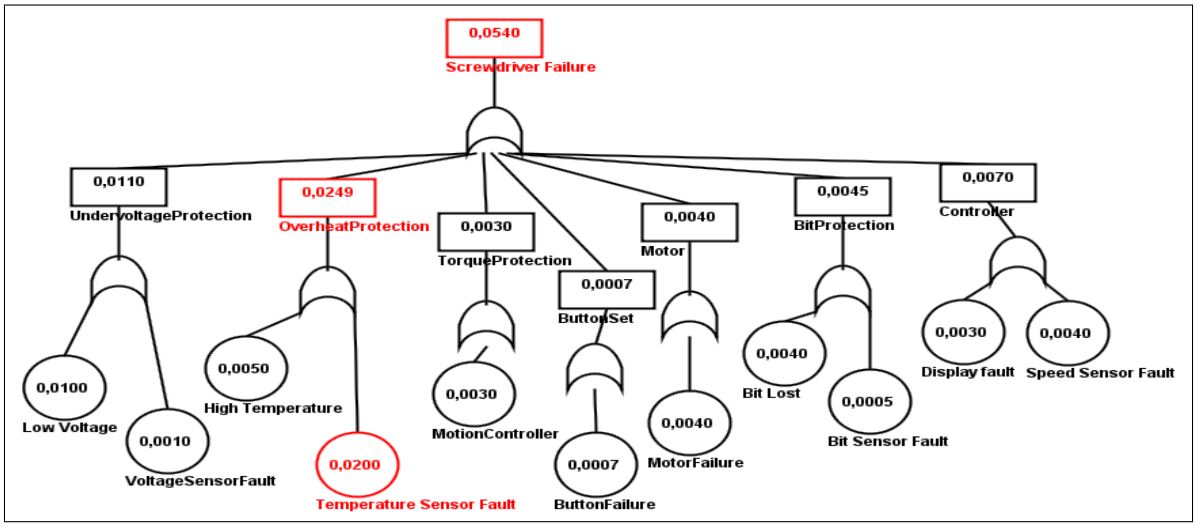


Fig. 10. FTA result in the FaultCAT tool, for the screwdriver system

after synchronization of the NFPs, is now subject to FTA.

D. FTA of the screwdriver system

In Fig. 10, the path that is most likely to cause a system failure is highlighted (in red color) by the FaultCAT tool, after performing a FTA. In this case, it is the *OverheatProtection* path. On the other hand, when there is incorrect *failure rate* specified for the *MotionController* (i.e., 0.3% instead of 0.003 %), it was seen that the FTA results indicated the *TorqueProtection* path as the most likely path in the fault tree to cause a system failure. This would have been the case, had the step to synchronize the attributes before NFP analysis (i.e., step 6 in Fig. 4) not been carried out. Thus, the need to to synchronize the NFP attributes of software components (before performing an NFP analysis) when they are implemented across heterogeneous modeling domains is reiterated, by this example.

VI. Summary

An emerging research challenge in NFP analysis in MDD is the model-based synchronization of NFP attributes, when the ESE scenario involves more than one modeling domain (e.g. UML, Simulink). Addressing this aspect, a generic work-flow for model-based co-engineering and NFP analysis of embedded software systems developed across heterogeneous modeling domains is proposed. A mechanism to synchronize a reliability attribute, across UML and Simulink domain is illustrated with examples. The FTA results reiterate the need to synchronize NFP attributes, before NFP analysis, when they are part of software components implemented across heterogeneous modeling domains. An item for future work is the evaluation of the proposed approach in a more sophisticated case study involving several NFPs.

Acknowledgment

This work is supported by a grant from the Federal Ministry of Economics and Technology (BMWi), Germany. This project work is carried out in cooperation with Willert Software Tools GmbH and SymtaVision GmbH. The authors would like to thank Stephan Wessels from University of Osnabrueck, for his valuable contribution towards the prototype implementation.

References

[1] D. C. Petriu, *Software Model-based Performance Analysis*. John Wiley & Sons, Inc., 2013, pp. 139–166.

[2] I. Sommerville, *Software Engineering: 10th Edition*. Pearson, 2016.

[3] J. M. Fernandes and R. J. Machado, “Requirements in engineering projects.” Springer International Publishing, 2016.

[4] 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.

[5] S. Bernardi, J. Merseguer, and D. C. Petriu, *Model-Driven Dependability Assessment of Software Systems*. Springer Inc, 2013.

[6] S. Balsamo, A. di Marco, P. Inverardi, and M. Simeoni, “Model-based performance prediction in software development: a survey,” *IEEE Transactions on Software Engineering*, vol. 30/5, May 2004.

[7] M. Hendriks, J. Verriet, T. Basten *et al.*, “Performance engineering for industrial embedded data-processing systems,” *Proceedings of the 16th International Conference on Product-focused Software Process Improvement*, 2015.

[8] Unified Modeling Language (UML), http://www.uml.org/, 2016.

[9] Matlab and Simulink, http://www.mathworks.com/, 2016.

[10] The UML profile for Modeling And Analysis of Real-Time and Embedded Systems (MARTE), http://www.omgmarte.org/, 2016.

[11] 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.

[12] FaultCAT: An Open Source Fault Tree Creation and Analysis Tool, http://www.iu.hio.no/FaultCat/, 2016.

[13] 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.

[14] 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.

[15] Willert Software Tools GmbH, http://www.willert.de/, 2016.

[16] Eclipse Modeling Framework (EMF), https://eclipse.org/emf/, 2016.

[17] Papyrus UML Tool, http://www.papyrusuml.org/, 2016.

[18] Object Management Group, http://www.omg.org, 2016.

[19] MATLAB Simulink Integration Framework for Eclipse, http://www.incquerylabs.com/, 2016.