Scientific Research Publishing

Journal of Software Engineering and Applications, 2018, 11, 204-222
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Automatic Derivation of Fault Tree Models from SysML Models for Safety Analysis

Bashar Alshboul, Dorina C. Petriu

Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada
Email: BasharAlshboul@sce.carleton.ca, Dorina.Petriu@sce.carleton.ca

How to cite this paper: Alshboul, B. and Petriu, D.C. (2018) Automatic Derivation of Fault Tree Models from SysML Models for Safety Analysis. *Journal of Software Engineering and Applications*, **11**, 204-222.
https://doi.org/10.4236/jsea.2018.115013

Received: April 13, 2018
Accepted: May 22, 2018
Published: May 25, 2018

Abstract

Safety Critical Systems (SCS) are those systems that may cause harm to the user(s) and/or the environment if operating outside of their prescribed specifications. Such systems are used in a wide variety of domains, such as aerospace, automotive, railway transportation and healthcare. In this paper, we propose an approach to integrate safety analysis of SCSs within the Model Driven Engineering (MDE) system development process. The approach is based on model transformation and uses standard well-known techniques and open source tools for the modeling and analysis of SCSs. More specifically, the system modeled with the OMG's standard systems modeling language, SysML, is automatically transformed in Fault Tree (FT) models, that can be analyzed with existing FT tools. The proposed model transformation takes place in two steps: a) generate FTs at the component level, in order to tackle complexity and enable reuse; and b) generate system level FTs by composing the components and their FTs. The approach is illustrated by applying it to a simplified industry-inspired case study.

Keywords

Safety Analysis, Model Transformation, Fault Trees, SysML, MDE

1. Introduction

Computer systems are widely used today in a multitude of domains. Individuals, communities, governments, industry, and organizations rely on computer technology to produce or innovate many aspects in a variety of areas, such as communication, education, healthcare, transportation, food, services, entertainment. The increase in the utilization of computer systems has significantly raised their complexity levels. Furthermore, there is an increase in the demand to build systems that meet various Non-Functional Properties (NFP), such as performance,

dependability, security, safety. Regulatory authorities and concerned bodies have put regulations and standards in different domains to control the development of such systems. This has triggered many research efforts in the academia and industry targeting the analysis of NFPs.

Dependability is the general NFP of interest in this paper. Dependability is defined as the ability to deliver services that can justifiably be trusted [1]. Dependability comprises five main attributes: Availability, Reliability, Maintainability, Integrity, and Safety. The last one, Safety, is the specific NFP of interest in this paper.

Safety is defined as the absence of catastrophic consequences on the user(s) and the environment [1]. Safety-Critical Systems (SCS) are the type of systems that can cause harm to the user(s) and/or the environment when operating outside of the prescribed specifications. These systems are used in various domains, such as: aerospace, automotive, railway and healthcare. Safety Analysis (SA) is performed on SCS to ensure that they are safe enough to be operational.

The survey in [2] performs a systematic literature review on SCS, concentrating on the evidence artifacts of systems for safety certification, including studies between 1990 and 2012. It shows that the number of publications targeting SCS and their safety evidence is increasing in all domains based on their respective standards. An interesting conclusion is that very few works are using model-based methodologies to obtain compliance evidence with safety standard.

Many SA techniques have been around for a considerably long time and have proven their effectiveness in performing SA; hence they are recommended and, in some cases, mandated by industry standards and certification authorities. However, applying such techniques imposes various challenges. Two well established SA techniques are Fault Tree Analysis (FTA) and Fault Model and Effect Analysis (FMEA). Traditionally, both techniques are performed manually in preliminary stages of the development cycle, but their application is error prone and time consuming, especially for systems with high complexity.

This paper contributes to an aspect of safety analysis that was found to be less supported by tools: the automatic derivation by model transformation of safety analysis models (namely fault trees) from SysML design models of the system under construction annotated with relevant safety information. The goal is to avoid error-prone manual work and to maintain the traceability between the software models used for development and the analysis models used to verify their NFPs.

1.1. Objective

This paper's objective is to develop a MDE-based approach with safety concerns in mind, utilizing standard and well-known techniques and open source tools. The aim is to integrate safety verification in the model-driven development of SCSs, allowing its artifacts to be maintained throughout the system life-cycle.

The paper proposes an approach for modeling SCS, with emphasis on

representing the safety behavior by using the OMG standard SysML language. One of the advantages of SysML is the ability to model component-based systems, which are used to tackle the system's complexity and to enhance the expressiveness and abstraction of the model. Related safety information will be added to the SysML model using another standard from OMG, the UML Profile for Modeling and Analysis of Real-Time Embedded Systems (MARTE) [3] along with its extension, the Dependability Analysis and Modeling (DAM) profile [4].

One of the objectives of this work is to support the construction, reuse, and composition of FTs at the component level. Therefore, we propose a two-step transformation of SysML models with safety annotations into safety analysis models. The first step incorporates the component-based development approach to synthesize component level fault tree safety models, which allow for reuse and simplification. In a consecutive step, system-level FT safety models are obtained by composing the component-level FTs built in the previous step. The generated FTs are used for system safety verification. A visualization of the FTs is also provided.

Another aim of the paper is to support the proposed method with tools easy to learn and use. For this reason, we are using existing modeling languages and notations to represent the system and the safety models, as well as open-source tools (e.g., Papyrus, Eclipse Epsilon, Eclipse EMFTA).

SysML has been chosen as the modeling language in this approach considering the following facts as described in [5]. First, it supports the specification, design, analysis, and verification of systems as a general-purpose modeling language. Second, it provides graphical models representing requirements, structure, behavior, and properties of systems and their components. Third, it is a standardized and robust language. Finally, it is an extension of a subset of the standard UML language, which adds the benefit of extending SysML with the standardized profiles defined for extending UML, such as MARTE [3].

1.2. Overview of the Approach

This section provides an overview of the proposed approach for performing SA on SysML based models. The main activities are as follows (see **Figure 1**).

- *System requirements* (including safety requirements), must be elicited and specified, whether the approach is applied to a new project or an existing one. A formal approach covering this activity is still under development. Hence, it will not be further detailed in this paper.
- *Architecture modeling* utilizing SysML's modeling power based on component-based modeling. The system is decomposed into a set of cooperating components, composite and simple. Composite components contain instances of other composite and/or simple components, while simple components are the most fine-grained level of decomposition and do not include other components.
- *Internal structure modeling*: specifies the composition and the interaction

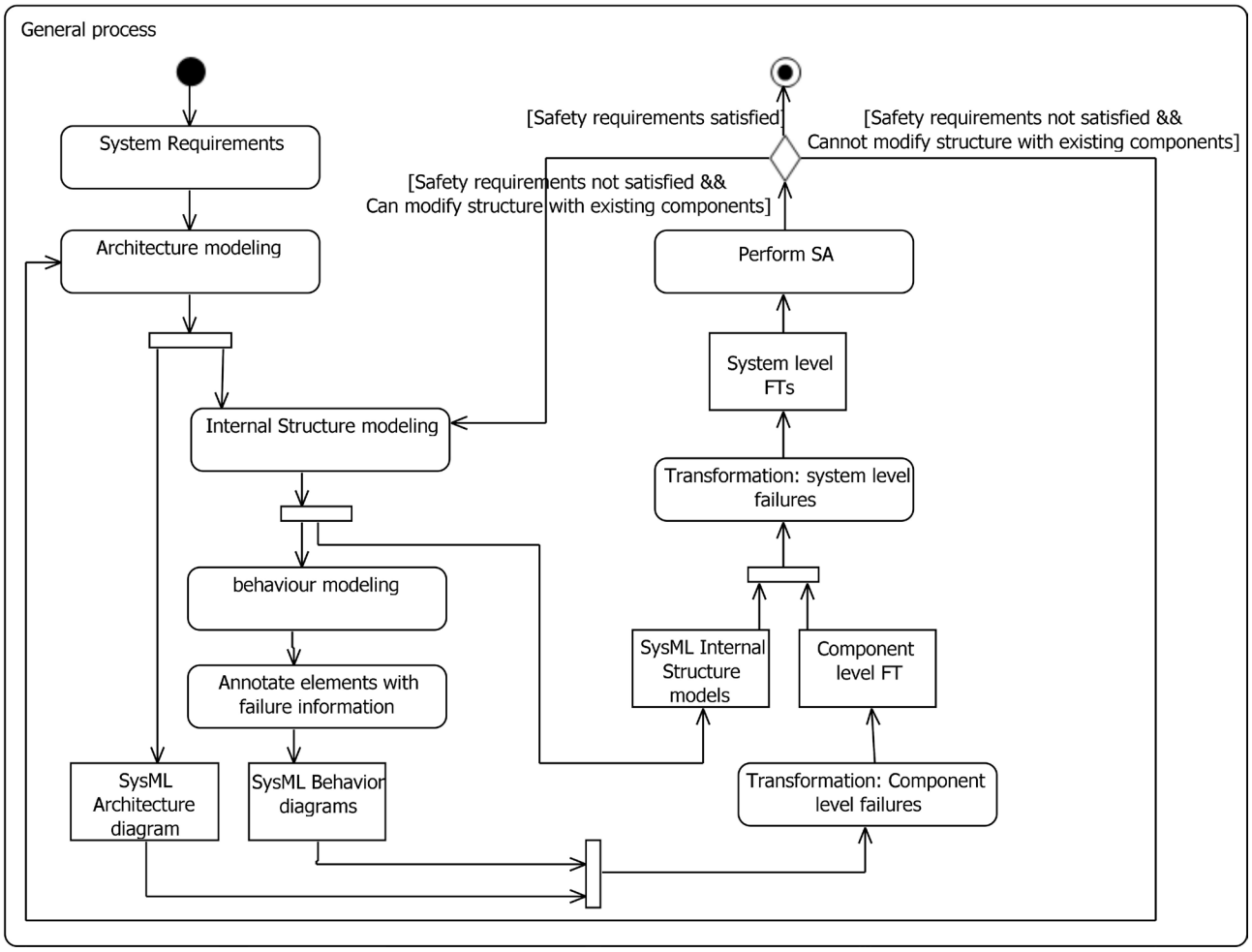

Figure 1. Overview of the proposed approach.

between component instances. A set of interfaces and possible item flows are realized modeling the component internal structure.

- *Behavior modeling*: the representation of the system regular behavior and its reaction to certain events.
- *Annotate elements with failure information*: adding safety behavior annotation to the previously specified component behavior.
- *Transformation*: *component level failures*, the generation of analysis FT models at the component level.
- *Transformation*: *system level failures*, the generation of analysis FT model representing system level safety behavior. This is achieved by the composition of component-level FT models, according to the interconnection of their corresponding component instances.
- *Perform SA*: perform analysis of the generated analysis models with existing FT tools.
- *Safety requirements satisfied or not*; if yes, accept the safety solution, otherwise apply design modifications to the proposed system to improve system safety. Examples of modifications use redundancy of existing components or

introduce new components into the architecture. This aspect is under formalization and will not be further detailed in this paper.

Model transformation is performed using an Eclipse-based model management platform called Epsilon (Extensible Platform of Integrated Languages for Model Management) [6]. The Epsilon platform supports different task specific languages. A shared expression language called Epsilon Object Language (EOL) is used in all task specific languages within Epsilon. EOL is a model-based language combining JavaScript style with Object Constraint Language (OCL) strength in object querying. Another task specific languages, Epsilon Transformation Language (ETL) facilitates model-to-model transformation. It accepts multiple source models and can generate multiple target models. ETL is a rule-based language categorized as hybrid for supporting both declarative and imperative rules. Our proposed two-step transformation is implemented in ETL.

2. Related Works

This section provides a literature review of related works. The survey in [7] reviews the work in modeling and analysis of software system dependability within the context of MDE using UML. Different approaches were surveyed targeting one or more of the dependability attributes. One of the conclusions is that the surveyed works provide support mainly in the early phases of the software life cycle (*i.e.*, from requirement to design), while there is a lack of support for later phases.

A recent paper [8] discusses current practices in industry working with SCS, analyzing the SCS challenges and the benefits of MDE to tackle such challenges at Siemens. The challenges discussed are: building and maintaining SA throughout the development life cycle, accommodating for changes of the system while maintaining the traceability with SA artifacts; SA artifacts reuse; and SA automation. The use of MDE helps the SA process automation, as it has the ability to represent the system at different levels based on the development phase and its modularity, allowing for the reuse of SA artifacts and providing traceability between system model elements and SA artifacts.

Another recent survey [9] analyses the status of the automatic application of FTA for system safety verification. An interesting point is that most current works provide their own model rather than using a standardized modeling language, such as AADL and SysML. Also, the representation of the failure behavior of the system is performed using different non-standardized approaches. Furthermore, the algorithms used for the generation of the FT models were also diverse due to the diversity of system models proposed.

The work presented in [10] proposes an approach to be used in an early stage of the life cycle along with a profile for SysML to add related safety information to model elements. The procedure starts in the Requirements Elicitation phase where safety requirements are identified. A manual analysis is performed on the functional architecture proposing safety functions, then the component archi-

tecture is analyzed manually adding the information from each analysis to the system model by stereotyping elements in the Activity diagrams (AD) and Block Definition diagrams (BDD) using the proposed profile.

There are various approaches that target the automation of the generation of SA models. They vary in the use of modeling language, the representation of the safety information, synthesizing, analysis and feedback of the analysis model and tools used.

The work presented in [11] performs automatic model transformation for systems modeled in UML into FTs to perform dependability analysis. The system is modeled using Use Case Diagrams (UC), Sequence Diagrams (SD), and Composite Structure Diagrams (CSD). The UML model is then annotated with dependability information using the profiles MARTE and DAM. The annotated UML model is transformed into a FT model by a transformation written in ATLAS Transformation Language (ATL) [12].

A methodology is introduced in [13] to perform SA on SysML models by using Python, a generic programming language. The SA techniques used are FMEA and FTA. The system functional architecture is modeled using a set of ADs representing system functions and their hierarchical breakdown. A Python program parses the XMI model file and generates a template for the Functional FMEA table, which contains the list of functions with a predefined list of generic failure modes. The table must be completed and validated by a safety expert. In the next step, components are assigned to functions, generating the logical architecture modeled in BDD. The Python program extends the Functional FMEA table, by adding the assigned components, then passes it back to the safety expert for completion and validation. The FMEA table information is integrated again in the XMI system model file with some proposed stereotypes. A FT with a generic top failure event is generated from IBDs based on the block connections from a single output to the inputs. It is proposed to use the Component FMEA results to build a FT for a specific failure mode without detailing the procedure. Similar to the FMEA table, the generated FTs need to be completed and validated by a safety expert.

Work using Architecture Analysis & Design Language (AADL) language presented in [14] proposes the use of AADL to model SCS architecture; the failure behavior is specified textually using Error Model Annex V2 (EMV2). A FT is generated from the annotated AADL models based on the data flow in the system architecture model. The Open Source AADL Tool Environment (OSATE) is used to generate AADL models and annotate them. The FTs are visualized and analyzed using an inhouse developed open source tool based on the Eclipse Modeling Framework (EMF) called EMF-based Fault Tree Analysis (EMFTA). EMFTA is integrated into the OSATE and provided as a standalone plugin for Eclipse. We are also using EMFTA in this paper to perform FTA.

An extension of the FT was proposed in [15], which is called Component Fault Tree (CFT). CFT is an extension of the FT by adding the notations of

components, input ports and output ports. This highly couples the component concept from the software model with the one from the fault tree model. CFT was implemented by a proprietary research tool which is not available to us. This extension has been originally proposed by a working group in [15] and then various enhancements and extensions were also introduced in [16]-[22]. This approach exploits the XMI file content for exploring the model using generic modeling language.

3. Source and Target Models

This section describes the source and target models used in our approach presented in this paper, illustrated with a case study.

3.1. Source Model

This subsection discusses the source model in the proposed approach, which is represented in SysML. The following SysML diagrams have been identified as being sufficient to model a system in order to enable its safety evaluation. A comprehensive SysML specification can be found in OMG's SysML specification [23].

- *Block Definition Diagram* (BDD) is used to define system blocks, their features, and possible relations among each other, including associations, generalizations and dependencies, modeling system hierarchy or a classification tree.
- *Internal Block Diagram* (IBD) is used to specify the internal structure of a block regarding its properties and their connectors.
- *State Machine Diagram* (SMD) is used to model an instance behavior as a finite state transition system. States can be simple states or composite states, which in turn contain nested states.

The case study used here was inspired by a tutorial [24] presented at ISSRE'2016. The Electric Kettle (EK) is an excerpt concentrating on the excessive boiling of the water that can cause harm to the user and/or the environment.

Applying our approach to the EK system begins with Architecture modeling, whose result is a SysML BDD diagram shown in **Figure 2**. The first block identified is the EK block, which is the system to be analyzed for Safety. EK is a subsystem of the Electric Kettle which interacts with other systems/subsystems using two proxy ports: one to accept raw temperature data signal, EKIn, and another to provide a decision signal based on a set of specifications, EKOut. Also, it contains an instance of a Heat Sensor, HS, and an instance of a Processing Unit, PU. A Heat Sensor block is the component responsible for accepting raw temperature value and delegating it to connected components. It can be interconnected with other blocks/components by accepting raw temperature data at a proxy port, HSIn, and passing it to another proxy port, HSOut.

The Processing Unit block accepts a temperature and evaluates whether it has reached the boiling point and needs to be shut down to prevent overboiling.

Processing Unit communicates with other blocks/components as it accepts temperature information at a proxy port, PUIn, which is processed and based on the specification produces a decision signal to another proxy port, PUOut.

Next, we model the Internal Structure of EK using a SysML IBD, which shows the interconnections among the block instances composing EK. The outcome IBD of the activity is realized in Figure 3. There are three interconnections in EK IBD, each one using an Item Flow realized with a connector. The first goes from the EK EKIn port to the HS HSIn port; the second from HS HSOut port to the PU PUIn port; and the third from PU PUOut port to EK EKOut port.

Next, we move to modeling behaviors. Each block/component of the EK has its own behavior specified by a SysML state-machine that represents the regular and failure behavior. Behavioral modeling of the EK system yields two SMD: one

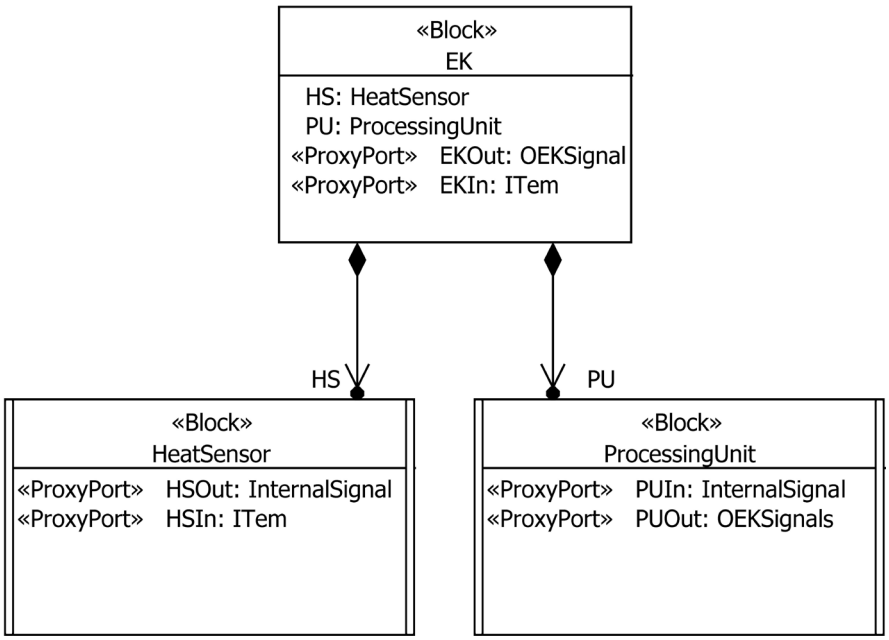

Figure 2. SysML EK BDD.

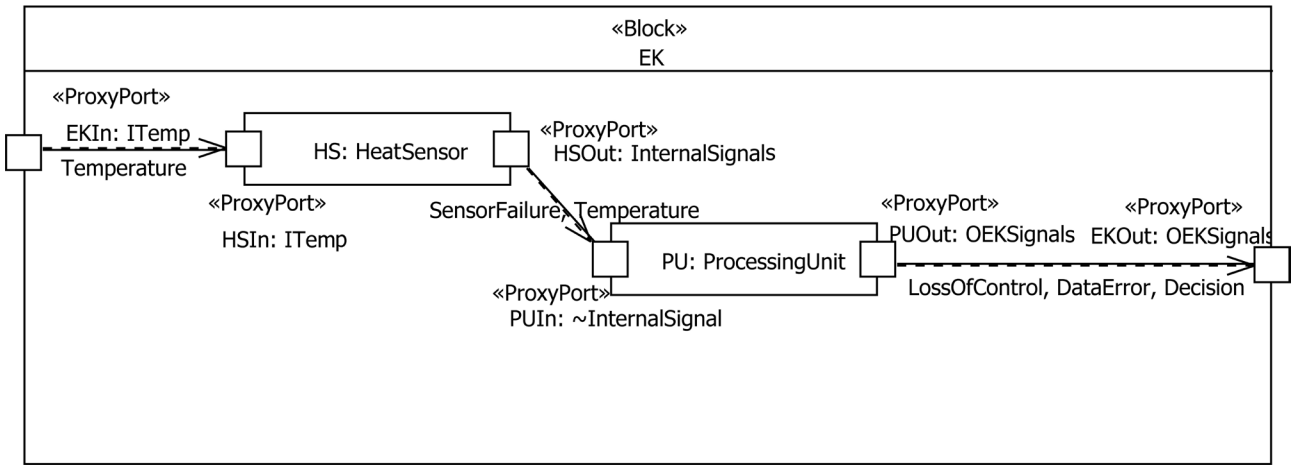

Figure 3. EK internal block diagram.

for the Heat Sensor and another for the Processing Unit block type for their regular behavior. Both SMD are similar in construct, having a state where the component is accepting an input signal on a specified port, then is performing specific logic for each component. Lastly, it will provide an output to a specified port, where some other components/systems are connected to.

Now consider the failure behavior for each component. The Heat Sensor have been identified to have one failure concern, when the sensor fails and leads into a Sensor failure state, which is added to its SMD. As for the Processing Unit, the component failure is considered and added, that can be caused by the failure of the component itself or by a failure signal received on the input port. Processing Unit has another safety concern, the corruption of the data that can result in an Erroneous data failure state. The SMDs with added failure behavior are shown in Figure 4 and Figure 5.

Next, we need to annotate safety-relevant model elements with failure information. This is accomplished using the powerful profile mechanism of SysML that is inherited from UML, which allows for tailoring models for specific needs, such as extending elements to model domain specific concepts. A profile is composed of stereotypes, tagged values and constraints. A dependability focused extension of OMG's standard MARTE profile, called DAM profile, contains the DaStep stereotype, which adds dependability information to the model element it extends. For instance, DaStep can hold the following information: 1) kind of threat to dependability (*i.e.*, Fault, Error, Failure and Hazard), associated with

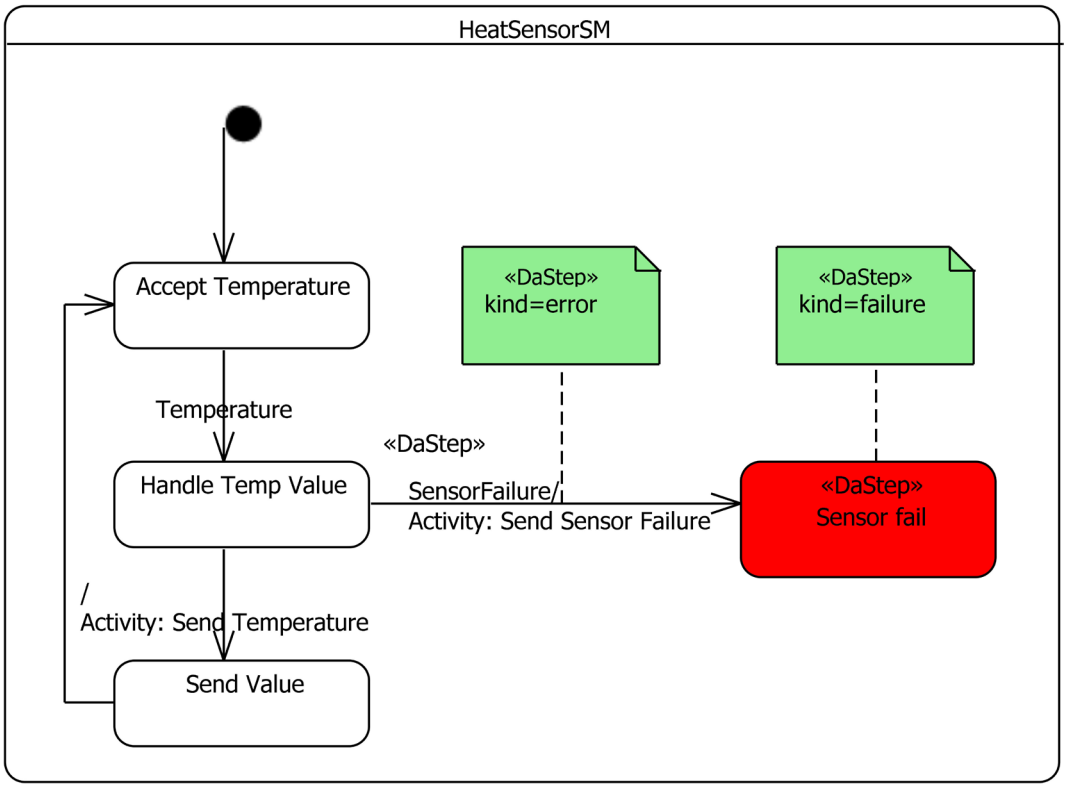

Figure 4. Heat sensor state machine diagram.

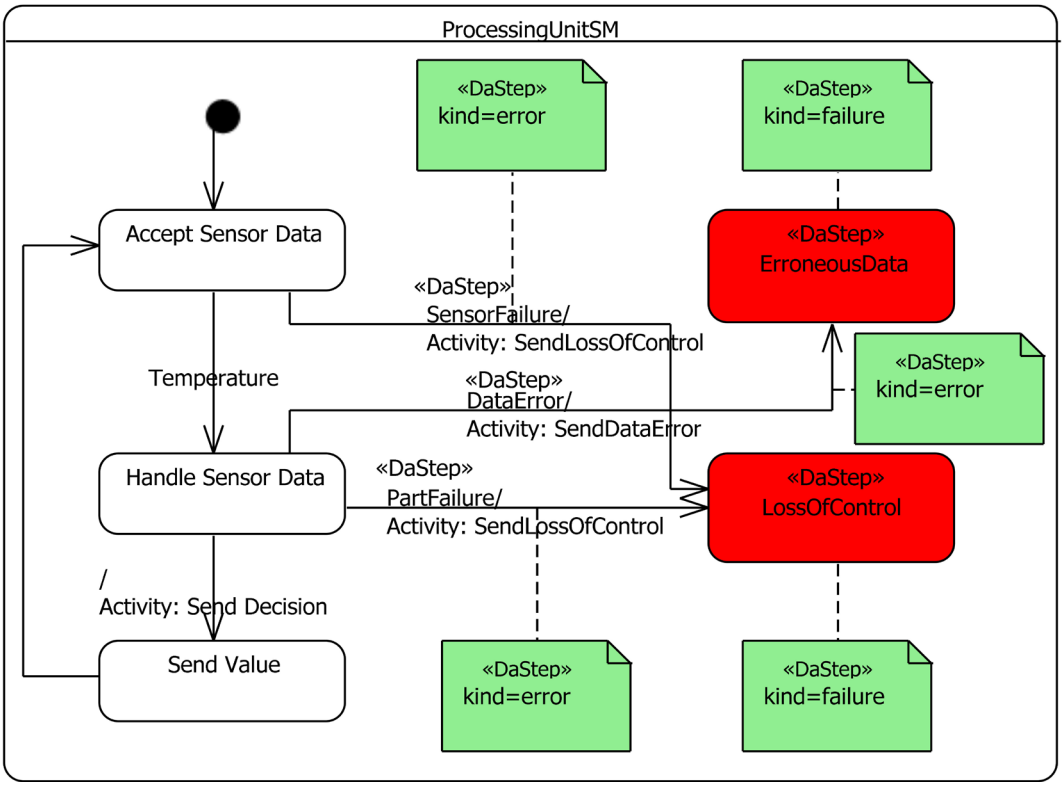

Figure 5. Processing unit state machine diagram.

the extended element; and 2) probability of its occurrence.

We applied the DAM profile to the EK system model by stereotyping the safety relevant states and their incoming transitions using DaStep stereotype. Failure states identified previously are set as "failure kind" and their incoming transitions are set as "error kind". The two SMDs resulting from this activity are depicted in **Figure 4** and in **Figure 5**.

3.2. Target Metamodels

The approach uses FTA to perform SA on a SCS. FTA is a deductive top-down failure-based analysis technique [25]. FTA is a deductive technique, top-down in nature, which starts with a high-level event towards its lower level causing events, represented in a tree-like structure called Fault Tree (FT) model. A FT depicts how the set of identified events are combined logically leading to the specific undesired top event. It contains various symbols for the several types of events and logical relations called gates. An example of a FT is provided in later section in **Figure 6**.

In our approach, we first generate component-level fault trees (CFT), which have the advantage of supporting the reuse of components and of their FTs. The CFT are composed in a second transformation phase to generate system-level FTs (SFT).

3.2.1. System Level FT

The SFT model is developed based on the metamodel of the EMFTA tool [26],

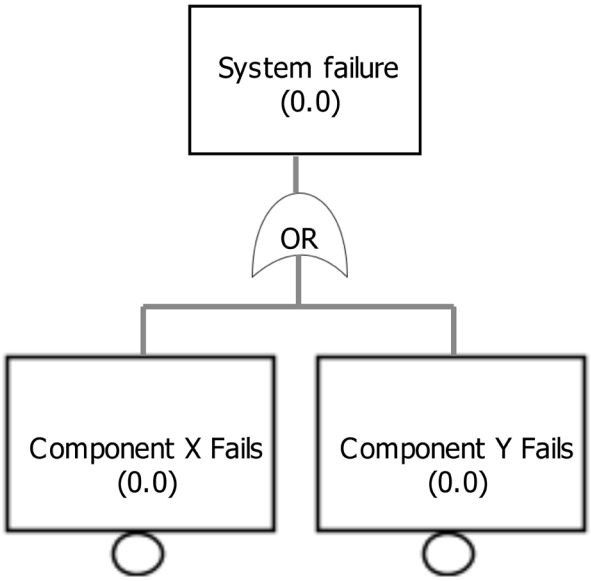

Figure 6. A sample FT.

that was introduced in [14]. We use EMFTA as the analysis tool for generated FTs. The reasons we selected EMFTA are the following: a) it is an open source software; b) it is designed on top of the Eclipse Modeling Framework (EMF) – the same framework used by Epsilon; and c) it is developed based on industry standards. EMFTA is available as an Eclipse plugin. The SFT metamodel is shown in **Figure 7** and consist of:

- FTA Model: the top element in the model, which is the container for the other model elements, events and gates.
- Event: one or more events contained within a FTAModel. One event can be set as the root of a fault tree when handling a single tree, each event can contain one gate. It can have a type attribute and a probability of occurrence as a float. The types are:
  - Basic: the lowest level of identified events leading to the top event that does not require further analysis.
  - External (House): events that are normal to occur.
  - Undeveloped: events that haven't been analyzed either due to the unavailability of its information or lack of effects.
  - Conditioning: specify conditions or restrictions affecting the logical gates.
  - Intermediate: events that have been further analyzed and their immediate causing events have been identified.

  The first four types of events are sometimes referred to as primary events.
- A Gate is used to link the causing events as input lower level events to the consequent event as an output upper level event using Boolean logic, which are set as a type of the gate.
  - And: all the input events must occur for the output event to occur.
  - Or: an occurrence of any input event can cause the output event to occur.
  - XOR: Exclusive Or, exactly one input has to occur for the output to occur.

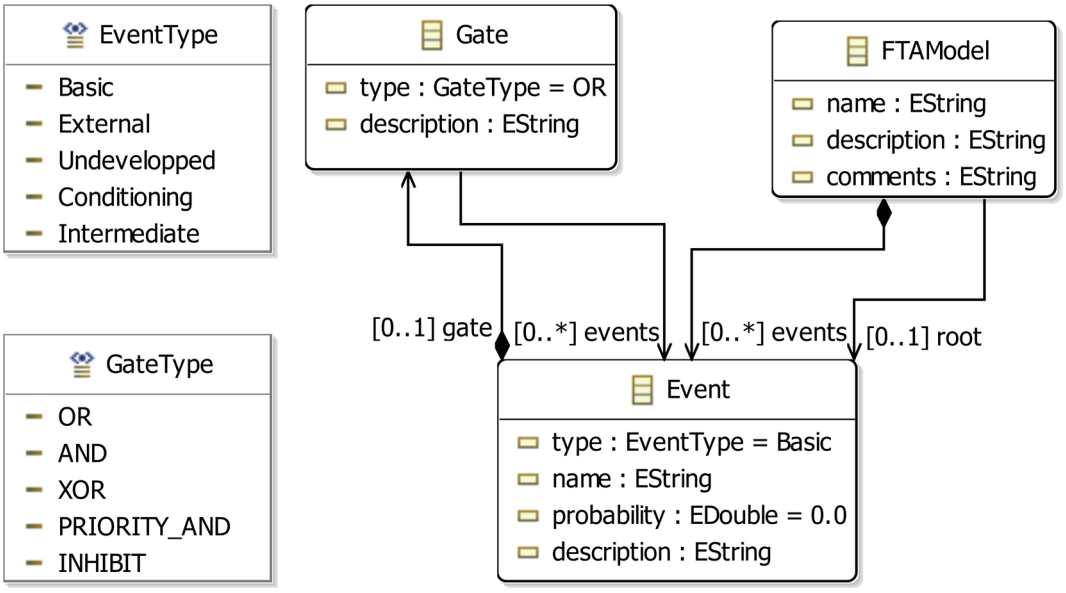

Figure 7. SFT metamodel (emfta).

- Priority And: The input events have to occur in a specific order for the output to occur. The order is specified as a conditional event.
- Inhibit: The input has to occur with the satisfying of a condition for the output event to occur. The condition is specified as a conditional event.

3.2.2. Component Level FT

Our transformation approach uses an intermediate model, the component-level fault tree (CFT) depicted in **Figure 8**.

This metamodel reuses the definition of the target analysis model from **Figure 7**. System is the main container for all the components and their CFT. Each block/component defined in the system is added as a component which might contain other components and ports along with its FT failure behavior. A port is contained within a component and it has a direction attribute.

4. Transformations

SysML model adoption proved its benefits for modeling various views for different domains and stakeholders, but it needs to be transformed into an analysis model to formally perform safety analysis. This chapter will discuss the transformation of SysML models as a source model into a classical FT model as target model to perform SA on the system.

The proposed transformation has been split into two consecutive steps. Our implementation allows to perform them either as two consecutive steps or as one single step. The transformations performed in this approach are depicted in **Figure 9**.

4.1. Transform Component Level Fault Tree model

In this transformation step, the failure behavior at the component level will be

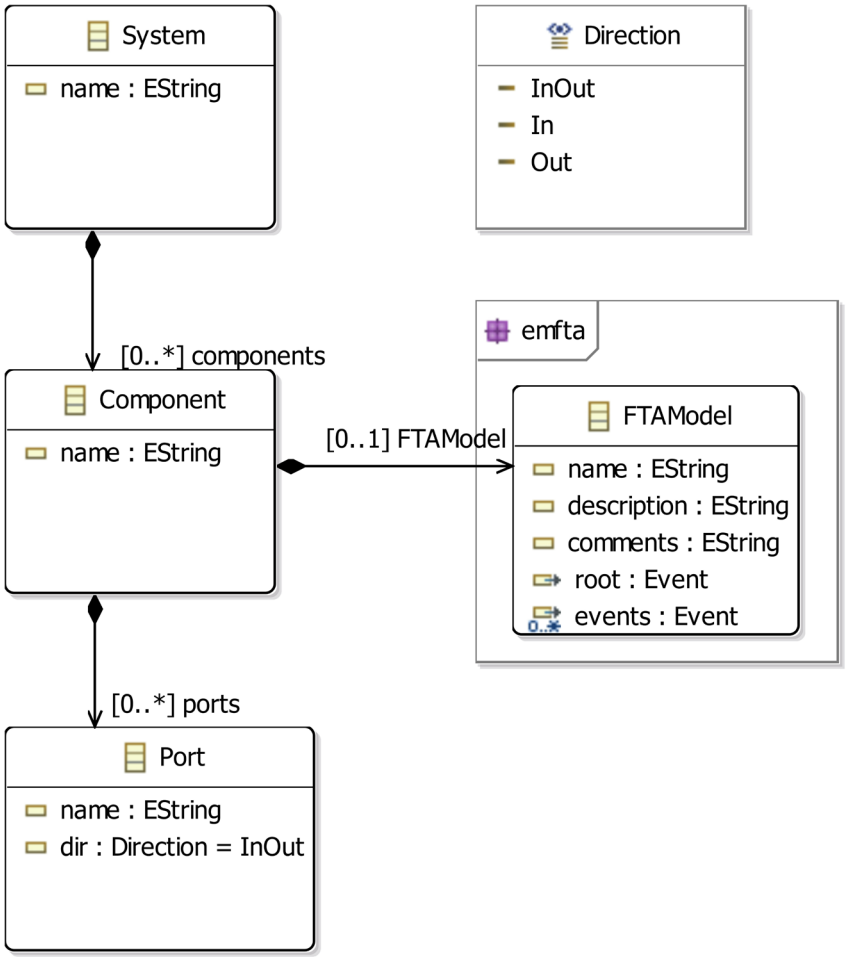

Figure 8. CFT metamodel.

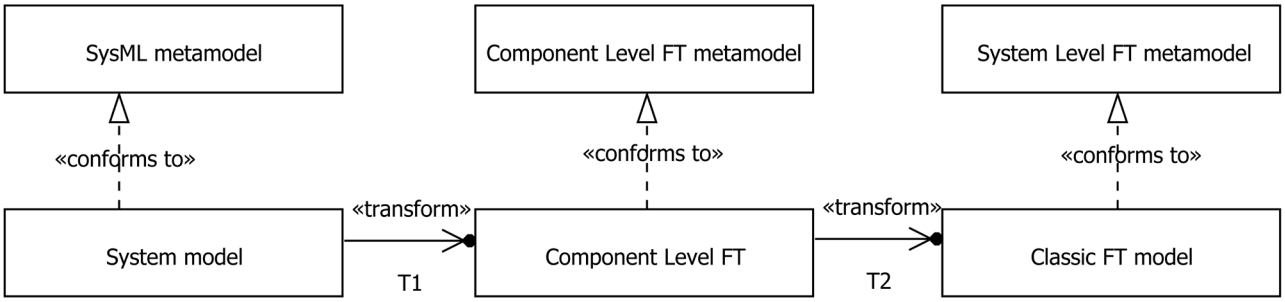

Figure 9. Model Transformation.

transformed into a component-level FT model. A component can fail in different ways, which are modeled as multiple failure states in the state machine, as shown in the case study. Each way to fail may correspond to a CFT with a different top failure event.

The transformation starts with transforming each block defined in the source model into a component in the intermediate model. A block is examined to check for other elements that are to be transformed as well, such as ports and behavioral model. The structural model that shows how the components are

connected is not considered in this step.

The port transformation considers the port type. The source model ports are modeled using proxy ports, typed with Block Interface elements, which in turn contain Flow Properties having a direction attribute. Thus, the port type is used to identify its direction.

As discussed previously, the behavioral model of a block is specified using state machines. A block's SM is transformed into a FT model of the block's component. This is followed by examining the SM's elements to synthesis the complete CFT.

Each failure state stereotyped with DaStep, is transformed into an event with an OR logical gate. In [25] it is stated that an event cannot be directly connected to another event, there should be a logical gate related to consecutive events. This rule was implemented in the FT metamodel, for both the component and system levels models. A failure event is considered as the top events for a CFT. Next, we are looking for the contributing preceding events.

A SM transition stereotyped with DaStep is considered an erroneous transition. A contributing event to a SysML failure-state (that is mapped to a CFT top event) is the event that triggered a transition leading to the respective failure state. A send signal action is considered a contributing event to the destination failure state, hence, translated into an event and linked to its resulting top event.

For instance, **Figure 10** provides the mapping of the behavior of the PU component model to a CFT model. It shows that the two failure states in the SysML model have been mapped into two top events for two CFT synthesized for the component. The triggers of the transitions that have a failure state as destination have been mapped into a basic event. Another mapping is that, when there is more than one incoming failure transition to a failure, once they are mapped into events, they will be linked with the top event using an OR-gate.

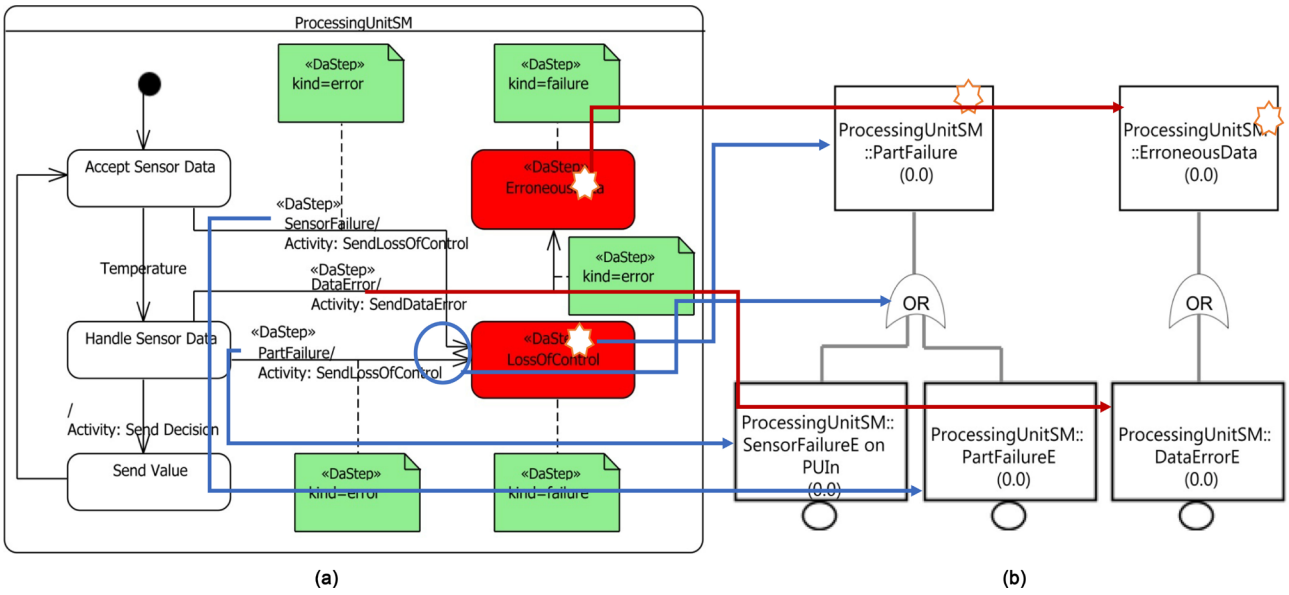

Figure 10. Mapping between SysML and component level FT.

A sample rule is shown in Figure 11 transforming a failure state from the SysML source model to an event in the CFT model, marked with stars in Figure 10. The rule can be read as transforming an element of type SysML::State stereotyped as DaStep into an element of type CFT Event. First, an equivalent of the containing SysML State Machine in CFT element, this will be used to correctly link the mapped elements. The name of the state will be used as the name of the event. Lastly, this is considered as a top event, hence, it must have causing events, which will be linked to the gate of this event (which defaults to OR-gate).

4.2. Transform System Level FT

In this step of the transformation, system level FTs are synthesized representing the failure behavior of the system of interest. This is accomplished by examining the interconnections between the components whose behavior was analyzed in the previous section to identify how their failure affects the system as a whole. This is achieved by analyzing the structure of the system that shows how the components are interconnected. In other words, the CFTs generated in the previous step will be composed based on the interconnection of the respective instances of their blocks.

The transformation first generates a FT model for the system of interest that will model the failure behavior of the system. In this transformation the internal structure is targeted, that is, how the blocks/components interact with each other and the effects of that on the failure of the system. This is accomplished by examining the block types, block instances, Item Flows and their realizing connectors modeling the interactions of the instances in an IBD.

The item flows/connectors are examined to follow the failure propagation from one component failure state transition to the following receiving state transition with consideration of the components instances. Delegation connectors model the receiving of an item or a propagated failure signal which get translated into basic events of the system of interest FT. In case the delegation connector is between output direction ports, then this will hold the outcome of the system or propagating a failure signal which will be translated into top events.

The assembly connectors model the flow of items between the internal parts and the possible failures propagated among them. These are used to connect the

```
rule state2topevent
    transformstate :SysML!Stateto event : CFT!Event {
    guard :state.hasStereotype("DaStep")

        var ft = state.containingStateMachine().equivalent();
        event.name = nameElementWithParent(state.name, ft.name);
        event.gate= newCFT!Gate;
        event.gate.type = CFT!GateType#OR;
        ft.events.add(event);
}
```

Figure 11. Epsilon ETL source code of state2topevent rule.

component-based FTs where failures are propagated, synthesizing a system level FT. So, a top event of a CFT can become an intermediate event in the system level FT if its failure has been propagated.

Continuing with the case study, Figure 12 show the mapping from CFTs to SFTs. The figure shows three nested figures: Figure 12(a) represents the CFT for the Heat Sensor Component; Figure 12(b) the component level FT for the Processing Unit; and Figure 12(c) the system level FTs.

The FT model produced by the transformation is provided in Figure 12(c). It shows that the system has two FTs, which indicate that the system can fail into two possible ways.

4.3. Example of Analysis

After the FT model have been synthesized depicting the failure behavior of the system, FTA can then provide valuable information about the system safety. FTA mainly provides critical qualitative value, but it can be extended to provide quantitative value as well. The key qualitative information provided by FTA is a set of basic events, called a "cut set", whose simultaneous occurrence will cause the top unwanted event to occur as well. A FT can have one or more cut sets. The basic events are events that cannot be further decomposed due to their nature or the limited information available concerning it.

In [25] a number of benefits in system design are discussed for the outcomes of FTA, mainly FT and the cut sets, which provide a logical comprehension of the causes and intermediate events inducing a specific top event, prioritization of the top event contributors, a proactive top event prevention tool and evaluation of

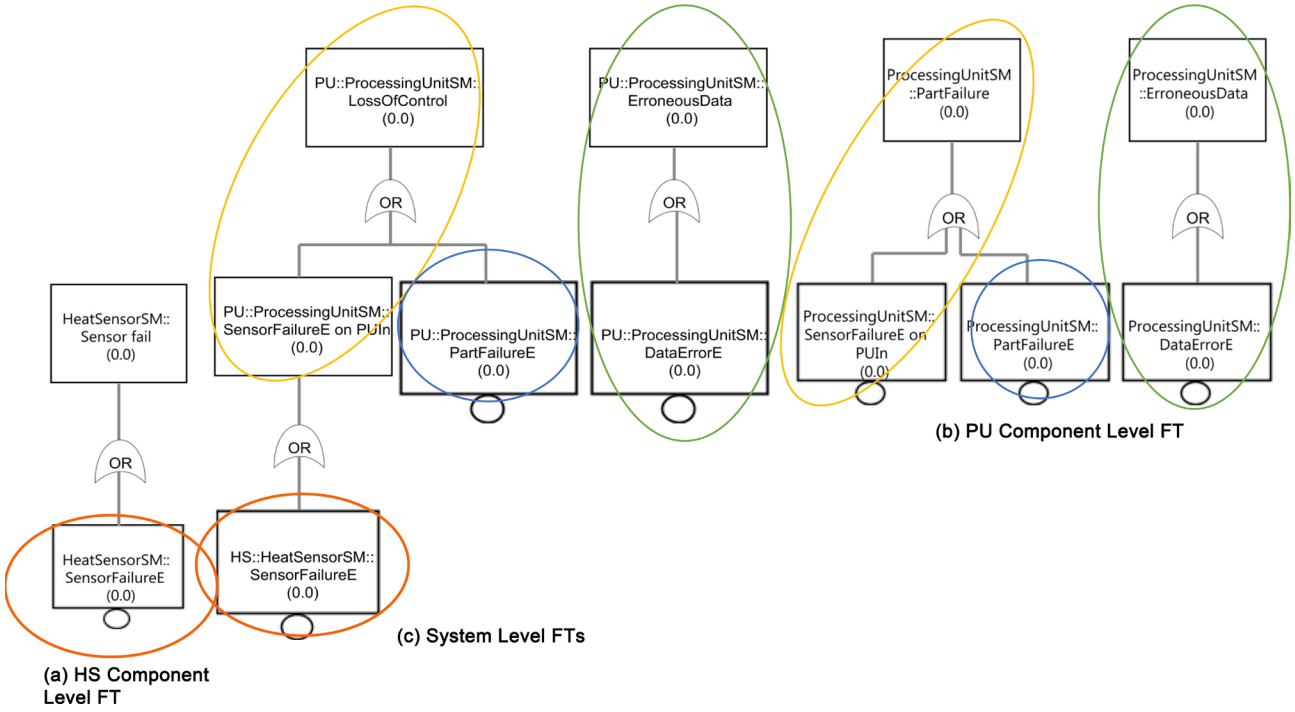

Figure 12. Mapping between component level FTs to system level FTs

design alternatives on the top event.

The EMFTA tool is used to automatically generate the cut sets for the SFTs shown in Figure 12(c). Table 1 contains the identified cut sets for both of the SFTs. Each column contains the cut sets for a specific top event/failure, where the EK system has two SFTs, one with Loss of Control (LoC) as a top event and the other with Erroneous Data (ED) as a top event.

For the top undesired event LoC, it contains two cut sets, each with a single event, which means that it is sufficient for any of these events to occur, in order for the LoC to occur, as well. As for the ED event, it contains a single cut set with a single event, that means, the occurrence of this event will cause the ED to occur.

Based on these cut sets, some design decisions can be made to protect against the occurrence of the identified failures. A decision can be to add a redundant Heat Sensor instance preventing it from being a single point of failure or to make the processing unit more resilient to failure. After a decision is made, the modified model can then be analyzed based on the design modification with reference to the general activities illustrated earlier in Figure 1 to evaluate its effect on the system.

5. Conclusions

This paper proposes an approach for integrating well established safety analysis techniques within a Model Driven Engineering (MDE) system development process. The approach integrates standard and well-known tools and techniques for the modeling and analysis of safety-critical systems (SCS), which can be applied to new and already existing projects with a small learning curve. This adds multiple benefits, such as increasing the safety and the level of confidence in SCS, reducing the costs in various aspects and enhancing the communication between all stakeholders, which means safer systems with minimal additional costs.

The proposed approach follows the Model Driven Engineering (MDE) paradigm, by modeling the system under study with OMG's standard Systems Modeling Language (SysML). The SysML system model is extended with safety annotations using another standard from OMG, the Modeling and Analysis of Real-Time Embedded Systems (MARTE) UML profile, along with an extension profile focused on dependability, called Dependability Analysis and Modeling

Table 1. Generated cut sets for the system level FTs.

Top events/failures	
PU::ProcessingUnitSM::LossOfControl	PU::ProcessingUnitSM::ErroneousData
Cutset #0 PU::ProcessingUnitSM::PartFailureE Cutset #1 HS::HeatSensorSM::SensorFailureE	Cutset #0 PU::ProcessingUnitSM::DataErrorE

(DAM) profile. The automation with the minimal efforts for performing SA on SCS, allows for continuous use of SA throughout the development life cycle, maintaining acceptable safety levels for the system. Moreover, the availability of CFT at components levels minimizes the effort towards the SA by reusing component fault trees in different configuration and systems.

References

[1] Avižienis, A., Laprie, J.-C., Randell, B. and Landwehr, C. (2004) Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, **1**, 11-33. https://doi.org/10.1109/TDSC.2004.2

[2] Nair, S., De La Vara, J.L., Sabetzadeh, M. and Briand, L. (2014) An Extended Systematic Literature Review on Provision of Evidence for Safety Certification. *Information and Software Technology*, **56**, 689-717. https://doi.org/10.1016/j.infsof.2014.03.001

[3] Object Management Group (2011) UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). Standard. http://www.omg.org/spec/MARTE/1.1

[4] Bernardi, S., Merseguer, J. and Petriu, D.C. (2013) Model-Driven Dependability Assessment of Software Systems. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39512-3

[5] Friedenthal, S., Moore, A. and Steiner, R. (2014) A Practical Guide to SysML: The Systems Modeling Language. 3rd Ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. https://doi.org/10.1016/c2013-0-14457-1

[6] Kolovos, D., Rose, L., Paige, R. and Garcıa-Domınguez, A. (2010) The Epsilon Book. Eclipse. https://www.eclipse.org/epsilon/doc/book/

[7] Bernardi, S., Merseguer, J. and Petriu, D.C. (2012) Dependability Modeling and Analysis of Software Systems Specified with UML. *ACM Computing Surveys*, **45**, 1-48. https://doi.org/10.1145/2379776.2379778

[8] Zeller, M., Ratiu, D. and Höfig, K. (2016) Towards the Adoption of Model-Based Engineering for the Development of Safety-Critical Systems in Industrial Practice. *Computer Safety, Reliability, and Security. International Conference on Computer Safety, Reliability, and Security*, **9923**, 322-333. https://doi.org/10.1007/978-3-319-45480-1_26

[9] Berres, A. and Schumann, H. (2016) Automatic Generation of Fault Trees: A Survey on Methods and Approaches. *Risk, Reliability and Safety: Innovating Theory and Practice*, CRC Press, 2485-2492.

[10] Müller, M., Roth, M. and Lindemann, U. (2016) The Hazard Analysis Profile: Linking Safety Analysis and SysML. *Annual IEEE Systems Conference* (*SysCon*), Orlando, FL, 18-21 April 2016, 1-7. https://doi.org/10.1109/SYSCON.2016.7490532

[11] Zhao, Z. (2014) UML Model to Fault Tree Model Transformation for Dependability Analysis. MASc Thesis, Carleton University, Ottawa.

[12] Obeo ATL: ATLAS Transformation Language. https://eclipse.org/atl/

[13] Mhenni, F., Nguyen, N. and Choley, J.Y. (2018) SafeSysE: A Safety Analysis Integration in Systems Engineering Approach. *IEEE Systems Journal*, **12**, 161-172. https://doi.org/10.1109/JSYST.2016.2547460

[14] Feiler, P. and Delange, J. (2017) Automated Fault Tree Analysis from AADL Models. *ACM SIGAda Ada Letters*, **36**, 39-46. https://doi.org/10.1145/3092893.3092900

[15] Kaiser, B., Liggesmeyer, P. and Mäckel, O. (2003) A New Component Concept for Fault Trees. *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software*, **33**, 37-46.

[16] Höfig, K. and Zeller, M. (2016) SpARTA—State-Aware Fault Tree Analysis. Euromicro DSD/SEAA 2016.

[17] Mohrle, F., Zeller, M., Hofig, K., Rothfelder, M. and Liggesmeyer, P. (2015) Automated Compositional Safety Analysis Using Component Fault Trees. Presented at the: 2015 *IEEE International Symposium on Software Reliability Engineering Workshops*, Gaithersburg, 2-5 November 2015, 152-159. https://doi.org/10.1109/ISSREW.2015.7392061

[18] Höfig, K., Zeller, M. and Heilmann, R. (2015) ALFRED: A Methodology to Enable Component Fault Trees for Layered Architectures. 41*st Euromicro Conference on Software Engineering and Advanced Applications*, Madeira, 26-28 August 2015, 167-176. https://doi.org/10.1109/SEAA.2015.26

[19] Adler, R., *et al*. (2010) Integration of Component Fault Trees into the UML. Models in Software Engineering. *International Conference on Model Driven Engineering Languages and Systems*, Oslo, 3-8 October 2010, 312-327.

[20] Domis, D. and Trapp, M. (2008) Integrating Safety Analyses and Component-Based Design. In: Harrison, M.D. and Sujan, M.-A., Eds., *Computer Safety*, *Reliability*, *and Security*, Springer, Berlin Heidelberg, 58-71. https://doi.org/10.1007/978-3-540-87698-4_8

[21] Kaiser, B., Gramlich, C. and Förster, M. (2007) State/Event Fault Trees—A Safety Analysis Model for Software-Controlled Systems. *Reliability Engineering & System Safety*, **92**, 1521-1537. https://doi.org/10.1016/j.ress.2006.10.010

[22] Grunske, L. and Kaiser, B. (2005) An Automated Dependability Analysis Method for COTS-Based Systems. In: Franch, X. and Port, D., Eds., *COTS-Based Software Systems*, Springer, Berlin, Heidelberg, Vol. 3412, 178-190. https://doi.org/10.1007/978-3-540-30587-3_28

[23] Object Management Group (2015) OMG Systems Modeling Language (OMG SysML™) V1.4. Standard. http://www.omg.org/spec/SysML/1.4

[24] Höfig, K. (2016) Tutorial 9: Dependability Analysis in the Context of Component-Based System Architectures. 27*th International Symposium on Software Reliability Engineering*, Ottawa, 23-27 October 2016.

[25] Vesely, W., Dugan, J., Fragola, J., Minarick and Railsback, J. (2002) Fault Tree Handbook with Aerospace Applications.

[26] CMU-SEI (2017) EMFTA: EMF-Based Fault-Tree Analysis Tool. https://github.com/cmu-sei/emfta