FLA2FT: Automatic Generation of Fault Tree from ConcertoFLA Results

Zulqarnain Haider, Barbara Gallina, Enrique Zornoza Moreno
IDT, Mälardalen University
Västerås, Sweden
zulqarnain.haider@mdh.se

***Abstract*—Dependability-critical systems (e.g., space systems) need to be engineered according to dependability standards (e.g. ECSS standards), which require the application of various dependability analyses, including Fault Tree Analysis (FTA). Due to the complex nature of such systems, conducting FTA may turn out to be time-consuming and error prone. Thus, automation is highly desirable. In this paper, we build on top of our previous work and we propose FLA2FT, a tool-supported Fault Tree (FT) generation from ConcertoFLA results. More specifically, we integrate FTA in a well-established existing system modeling and analysis methodology to generate FT automatically using model transformations. To illustrate the usage of FLA2FT, we apply it to the space domain and automate the generation of ECSS-compliant FTs for an Attitude Control System (ACS). Finally, we draw our conclusions and sketch future work.**

Keywords-Model Driven Engineering; Fault Tree Analysis; Attitude Control System

I. INTRODUCTION

The development of dependable (safety-critical) embedded systems is regulated by standardization bodies, which issue domain specific standards, and certification bodies, which are responsible for checking compliance. Space software systems for instance are engineered according to the standards proposed by European Cooperation for Space Standardization (ECSS). More specifically, ECSS-Q-ST-30c [1] and ECSS-Q-ST-40c [2] list requirements for the assurance of dependability and safety respectively along with the methods for analysis and assurance of these properties. Fault Tree Analysis (FTA) is a deductive method for analyzing system to identify the causes of system failure. Performing FTA manually for such complex systems is time consuming and error prone [3]. Automatic generation of Fault Tree (FT) and integration with other automated activities in a standard Software Development Life Cycle (SDLC) could reduce the cost and effort and enable compliance to the applicable standards.

Model driven and component-based engineering represent established engineering methods to master complex systems. CHESS [4] is one such tool-supported methodology targeting high integrity embedded systems. CHESS provides a dependability profile [5] implemented in CHESS Modeling Language (CHESS ML), which itself is a profile of SysML [6]. ConcertoFLA [7] is a tool-supported failure logic analysis method (part of CHESS toolset), which calculates the failure behaviour of a system based on the failure behaviour of composing components (specified in CHESS ML). System design represented via CHESS ML models and their enrichment for safety analysis are coherent and supportive to standards, e.g., in space domain ECSS-E-ST-40C [8] and ECSS-Q-ST-40c [1] referring to system design and safety respectively.

In our previous work [9], we discussed about the feasibility of exploiting the causality paths generated by ConcertoFLA, to derive ECSS compliant FTA results manually. In this paper, we move a step further and we propose FLA2FT, a tool-supported fault tree generator, which generates fault trees from ConcertoFLA results. FLA2FT empowers an existing toolset by producing the certifiable evidences (compliance means) automatically. FLA2FT is based on a master thesis presented in [10]. It will soon be integrated into PolarSys OpenCert tools platform [11] and available to download.

The remainder of this paper is organized as follows. Section II provides the background information. Section III gives an overview of FLA2FT. Section IV, presents the mapping and implementation of transformation. Section V, describes the application of FLA2FT to Attitude Control System (ACS). Section VI discusses related work. Finally, in Section VII, we present conclusion and sketch future work.

II. BACKGROUND

In this section, we present the background information relevant to our work.

A. Model Driven Engineering (MDE) and Model Transformation

Model Driven Engineering (MDE) is a methodology, which raises the level of abstraction and produces well-structured and maintainable systems. In MDE, the development life cycle of a system is automated through the usage of models [12]. Object Management Group (OMG) proposed Model Driven Architecture (MDA) to standardize and implement MDE. In this regard, a general purpose modeling language is proposed i.e., Unified Modeling Language (UML), which conforms to Meta-Object Facility (MOF) [13] another OMG standard for specifying, manipulating and integrating metadata. Eclipse Modeling Framework (EMF) [14] implements MDA and provides support for defining MOF compliant meta models, and its code generation feature enables to build tool support. The meta models explained in Section II.B and II.C are defined using EMF and tool support is provided through the above-mentioned code generation feature.

Model transformation is a set of rules, which defines the translation of one model to another model often referred to as the *source* and *target* model respectively. Epsilon Transformation Language (ETL) [15] is a hybrid model-to-model transformation language. Thus, provides both declarative and imperative features to handle the wider range of transformation scenario. The former is provided through transformation rules, and targets higher abstractions, where the latter is supported via transformation operations and addresses the transformation on lower level.

B. *Fault Tree Analysis*

IEC 61025 [16], a specific FTA standard referred to in ECSS, defines FTA as "*a deductive (top-down) method of analysis aimed at pinpointing the causes* (faults) *or combinations of causes that can lead to the defined top event*". Faults, their causes and the relation between them, specified via Boolean logic, are represented graphically (see [17] for the details regarding the graphical notation). In this work, the focus is on the generation of qualitative FT from 1) existing models (representing the system design at various level of abstractions, i.e., high level, low level etc.), and 2) dependability analysis results. In [17] a meta-model is proposed to represent FTs. Fig. 1 (adopted from [19]) shows this meta-model, which defines the abstract syntax of the language for representing FTs. The meta-model has following three elements: 1) *FTAModel*, represents FT and contains all of its events and their relation in the form of gates, 2) *Event*, is associated to FTAModel as a root and many containing events, 3) *Gate*, represents the relationship between an event and causing events. Both event and gate could be of different types, where all the applicable types are defined using enumeration entity and referred to as *EventType* and *GateType* respectively. To this end, for representing qualitative FT, our approach uses two events i.e., *Basic* & *Intermediate* and two gates i.e., OR & AND. An open source EMF based tool [13] implements this meta-model as well as provides support for editing and graphically visualizing the FTs.

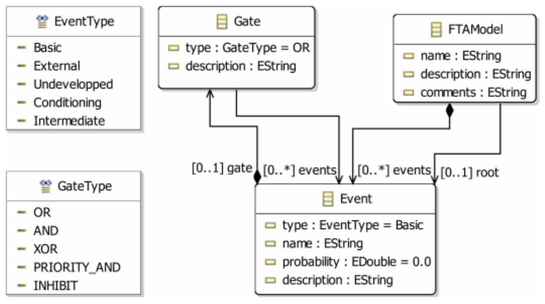


Figure 1. FTA Metamodel (adopted from [14])

C. *ConcertoFLA*

ConcertoFLA [7] is a failure logic analysis approach, which builds on top of Failure Propagation Transform Calculus (FPTC) [20] and calculates the failure behaviour of a system using the failure behaviour of composing components of the system. In ConcertoFLA, FPTC rules are used as a set of logical expressions to relate the failures on the output ports of a component with the failures occurring on its input port to compose the failure behaviour of the component. ConcertoFLA, supports three abstract failure types/modes i.e., value, timing and provision with specializations for each as Subtle/Coarse, early/late and Omission/Commission respectively. FPTC rule is a combination of input expression mapped to the output expression, defined as "Inputport.failuretype" and "Outputport.failuretype" respectively. Fig. 2 shows Failure Logic Analysis Meta Model (FLAMM), proposed in [22], which defines an abstract syntax for the language to represent the ConcertoFLA results. The elements of FLAMM and their relations are explained as below.

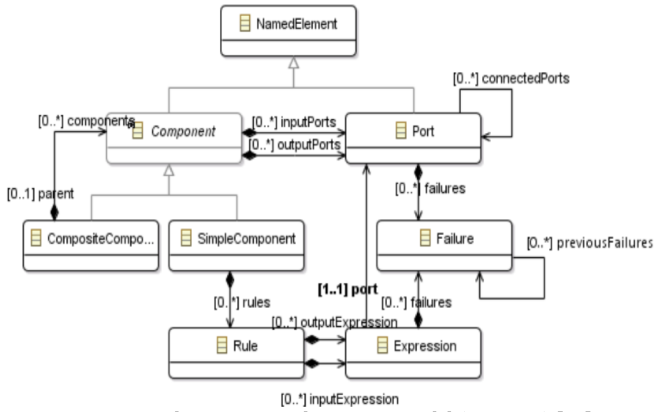
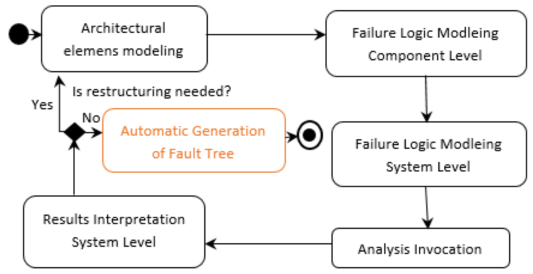


Figure 2. Failure Logic Analysis Meta Model (FLAMM) [18]

A *Component* can be defined as a *CompositeComponent* or a *SimpleComponent*. The former represents a system, while the latter refers to its composing components. The parent attribute of component describes the relation of composite and composing components. Component has *InputPorts* and *OutputPorts* of element *Port* type. The failure behaviour discussed above is specified on these input/output ports. The port element has a reference to itself to support the connected ports, as component based systems assembly is achieved by connecting the ports of composing components in a required configuration. This relation is used to determine the path of failure propagation and transformation. SimpleComponent has rules describing its failure behaviour, composed of *InputExpression* and *OutPutExpression*. The Expression associates the Failure and Port. Similar to port, failure has an association to itself referring to the previous failures (i.e., causes of this specific failure). ConcertoFLA is also supported with an EMF based tool support, which implements the FLAMM and enables a support to decorate component based architectural models with above-mentioned FPTC rules, executes failure logic analysis and back propagates the analysis results to the model. The tool support extends its predecessor CHESS-FLA [20], and is integrated in CHESS toolset.

III. FLA2FT OVERVIEW

In Fig. 3, we recall the overall approach that builds on top of our previous work [9], and initially presented in [7] and further developed in [22]. In this paper, the manual step in previous work [9] for generation of FT is automated and a tool support has been proposed (highlighted in the Fig. 3). Similar to [9], the approach is customized in the context of ECSS to perform the failure logic analysis at sub/system level, improve the design and automatically generate ECSS complaint FT using ConcertoFLA results and FT generation plugin (implemented in this work) respectively.


Architectural elemens modeling
Failure Logic Modleing Component Level
Is restructuring needed?
Yes
No
Automatic Generation of Fault Tree
Failure Logic Modleing System Level
Results Interpretation System Level
Analysis Invocation
Figure 3. Overview of the approach (adopted from [7])

The overall approach presented above, involves two model-to-model transformations as illustrated in Fig. 4. The system model annotated with dependability information conforms to the CHESSML. ConcertoFLA plugin transforms this model in to FLA results, which conforms to FLAMM. Lastly, the FT generator plugin transforms FLA results into FT model expressed using EMFTA meta-model. This final transformation is presented in the next section.

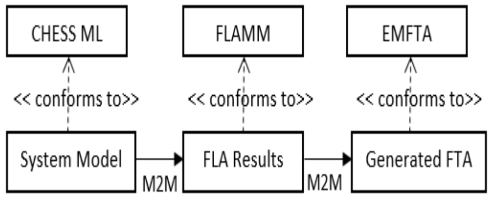


Figure 4. Model Transformation of overall approach

IV. FLA2FT Transformation

This section presents the transformation mapping of FLA2FT along with the implementation and integration of this mapping.

A. *Mapping of the entities from source to target*

Table 1. provides transformation mapping rules of the elements and attributes of the source model (FLAMM) to the elements and attributes of the target model (EMFTA). A combination of Port and Failure is mapped to a specific event. The name of an event is a combination of the name of port, the component it belongs to and the id of failure – this is to identify the events uniquely in a FT. The *previousFailures* attribute of the Failure determines the type of the event in the target model. If a failure has no previous failures, this suggests that the event type should be Basic as it refers to the root cause of system failure. Rule and its attributes *input/outputExpression* transforms to gate and determines the type of gate between the input/output events. For a specific *outputExpression* in a rule, if the number of *inputExpression* exceeds than one, AND type of gate shall be created between the input and output event. The rationale is that all of the failures on the ports specified in the *inputExpression* shall occur to cause the failure of the *outputExpression*. Each of the output port of composite component along with the failures is transformed into a new FTAModel – this is to generate a separate FT for each system level failure. Fig. 5 shows the corresponding ETL implementation, where each output port and failure is transformed to a new FTAModel. This transformation mapping is realized in following two steps: 1) composite component output ports to FTAModel and generation of top event, and 2) population of FT.

1) *Composite component output ports to FTAModels and generation of top event*

Each output port of the composite component and the failure modes on that specific port corresponds to a separate FT; each of these FTs refers to the causes of a particular system level failure. In Figure 6. the algorithm shows the steps for the generation of FTAModel and identifying the top event. In step six and seven, for each output port that belongs to the composite component and for all failures on a specific port a new FT is generated with a top event referring to that specific port name and failure id.

TABLE I. Transformation Mapping

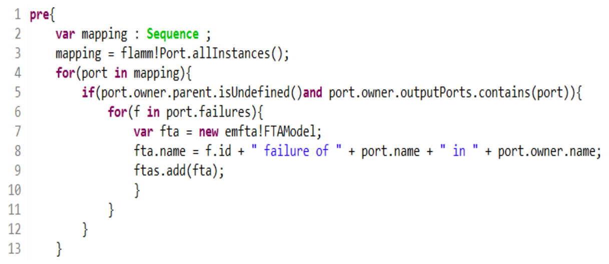
Source (FLAMM)		**Target (EMFTA)**	
Element	***Attribute***	***Element***	***Attribute***
Port	name	Event	name
	owner		name
	connectedPorts		gate
Failure	previousFailures	Event	type
	id		name
Rule	input/output Expression	Gate	type
Composite Component OutputPort	failures	FTAModel	name
	name		
	owner		

2) *Population of the Fault Tree*

By population of FT, we refer to the identification of remaining intermediate & basic events and the corresponding Boolean logic (gates) to connect these events. In FLAMM structure, the ports have failures and connected ports. To populate the FT we use a top down approach and start with the output port of composite component and its failures, which together refers to the top event of FT (as shown in Section IV.A.1). Using this output port, we identify the connected port and its failure to transform it into FT event. This algorithmic solution recursively identifies the causing events with respect to each connected port until all the ports are explored or port has no failure. This top-down recursion has similarities with a typical depth first search. During this process, three different types of ports can be identified via the connected ports. Fig. 6 illustrates the algorithm for FT population, which lists these three cases referring to the identified ports in the form of *if* conditions in step two, thirteen and sixteen. These three cases are discussed in detail next.

```
Algorithm 2 FT Population Algorithm
1:  procedure POPULATEFT($flaPort, flaFailure, FT$)
2:      if $flaPort.isOutputport() \in$ SimpleComponent then
3:          $FT.E \leftarrow createIntermediteEvent()$
4:          $R \leftarrow getAllrules()$
5:          for all $r \in R$ do
6:              if $r.outExpress() \in$ flaPort then
7:                  if $r.inExpressSize() > 1$ then
8:                      $FT.E.gate.type \leftarrow AND$
9:                  else
10:                     $FT.E.gate.type \leftarrow OR$
11:                 for all $iexp \in InExpression$ do
12:                     $populateFT(iexp.port, iexp.failure, FT)$
13:     if $flaPort.isInputport() \in$ SimpleComponent then
14:         $FT.E \leftarrow createIntermediteEvent()$
15:         $FT.E.gate.type \leftarrow OR$
16:         $populateFT(flaPort.connectedPort, flaFailure, FT)$
17:     if $flaPort.isInputport() \in$ CompositeComponent then
18:         $FT.E \leftarrow createBasicEvent()$
19:     return $FT$
```

Figure 6. FT Population Recursive Algorithm



```
pre{
    var mapping : Sequence ;
    mapping = flamm!Port.allInstances();
    for(port in mapping){
        if(port.owner.parent.isUndefined()and port.owner.outputPorts.contains(port)){
            for(f in port.failures){
                var fta = new emfta!FTAModel;
                fta.name = f.id + " failure of " + port.name + " in " + port.owner.name;
                ftas.add(fta);
            }
        }
    }
}
```

Figure 5. ETL code

a) Output port of a simple component

Output port of a simple component is connected to the output port of the composite component, which refers to the top event and identified in Section IV.A.1. For each failure on the output port of simple component, the contributing failures and the logical gate is determined through the rules of the component. The output port and failure is matched against the output expression of each rule. For each matched *outputExpression* the numbers of input expression are determined to identify the gate – if the number of input expression is greater than one, the events shall be connected through the AND gate otherwise OR gate. The former refers to the scenario that more than one failures need to occur to cause the resulting failure on the output port. Once, the gate is determined for the connecting ports; all of the input ports and their failures from each input expression is analyzed to identify the remaining corresponding events of FT, via recursively calling same method.

a) Input port of a simple component

A new intermediate event is created, having an OR gate, in the FT corresponding to this port and its failure. Input port of a simple component can be connected either to the output port of another simple component or an input of the composite port. The former is already discussed in previous section.

b) Input port of the composite component

In this case, a new basic event is created in FT referring to the port and the failure. The FT population stops when the search reaches the boundary of the system i.e., the input port of the composite component or when the connected port does not have any failure.

B. *Transformation Implementation and Integration in CHESS Toolset*

The above-mentioned model to model transformation is implemented using ETL. An ETL rule is developed to implement the Algorithm 1 and a recursive ETL operation implements the Algorithm 2 shown in Fig. 6 and Fig. 7 respectively. An Eclipse Plugin is developed which integrates this transformation, along with the functionalities to generate, visualize and edit FT in CHESS tool set. A user (System/Safety Engineer) selects the *Fault Tree Generation* menu bar entry and navigate to select the input FLAMM file, which contains the ConcertoFLA results. The plugin invokes the ETL transformation and generates the number of fault trees equal to the number of output ports of composite component and the type of failures. This results in less complex FTs due to being smaller, and provide opportunity to study a specific system level failure in isolation. The generated FTs are stored separately in a folder and a user can visualize and edit a FT by opening it into the fault tree viewer (editor) shown in Fig. 9.

```
Algorithm 1 FTAModel Generation and Top Event Identification
1: procedure GENERATEFTAMODELANDTOPEVENT($flammModel$)
2:     $P \leftarrow getAllOutputPorts(flammModel) \in$ Compositecomponent
3:     for all $p \in P$ do
4:         $F \leftarrow getAllFailures(p)$
5:         for all $f \in F$ do
6:             $FT \leftarrow createNewFTAModel(p.name, f.id)$
7:             $FT.events \leftarrow createTopEvent(p.name, f.id)$
8:     return $FT$
```

Figure 7. FTAModel Generation Algorithm

V. FLA2FT APPLICATION TO ATTITUDE CONTROL SYSTEM (ACS) ENGINEERING

ACS implements the functions required to maintain the orientation of a satellite in space relative to a reference of frame. ACS can have different operational modes, which typically involve different devices [23]. For example, in Sun Acquisition and Survival Mode (SASM) the ACS measures the gyroscopic rates and the Sun direction using Gyroscope and Sun Sensor for computing and sending control torque commands to propulsion thrusters.

ACS is engineered according to the ECSS standards, which require that FTA shall be conducted at all levels of system/sub-system development for safety and dependability analysis and assurance [1] [2]. In design phase, the analysis contributes in the evaluation of the dependability (reliability and safety) of the system as well as an improvement of high-level design through the identification of weaknesses and may consequently result in the refinement of safety and reiliability requirements.

To apply FLA2FT approach for ACS engineering, we used the flow illustrated in Fig. 3. We start with modeling the architectural specifications of ACS in SASM mode using CHESSML. In SASM mode, ACS has following six components: 1) *SignalConditioner*, process and transforms sensor data to satellite reference frame, 2) *StateEstimator*, estimates the satellite state using the current state measurements and historical data, 3) *PDController*, provides the proportional and derivative torques based on these estimates, 4) *SteerController*, also computes torque but using different gains and control law, 5) *FeedforwController*, compensates for the cross coupling torques, and 6) TorqueSelector, selects the applicable torque for maintaining the target attitude corresponding to current state..

We considerd a hypothetical scenario, that the sensor units provide inaccurate measurements. Therefore, we injected the ACS with *valueCoarse* fault on its input ports. In the preliminary design, the components propagate this *valueCoarse* type of failure – this behaviour is modeled using FPTC rules as discussed in Section II.C. We executed the ConcertoFLA, which calculated the failure behaviour of ACS based on the failure behaviour of its composing components and stored it into a FLAMM based file. Fig. 8 shows the architecture of ACS, injected faults and back-propagated system failure behaviour i.e., the result of ConcertoFLA. Finally, the FLA2FT transformation is executed to generate the FT from the ConcertoFLA results produced in previous step. Fig. 9 shows the partial view (due to the limitation of space) of corresponding FT generated using the implemented plugin presented in Section IV.B. The top event in the Fig. 9, i.e., *valueCoarse failure of ctrlTorque in ACSComposite*, refers to the system failure and corresponds to the valueCoarse failure on the ctrlTorque output port of ACSComposite component as illustrated in Fig.8.

A. *Compliance to ECSS Standards*

The generated FT complies with the ECSS standards as it fulfills the syntactic and procedural guidelines provided by IEC-61025 [17], which is incorporated in ECSS system.

For example, [17] requires that the top event should be defined unambiguously – this can be seen in Fig. 9, where the top event is shown as the combination of the type of failure, port name and the system name. Another requirement is that the FTs need to be drawn vertically or horizontally. In our approach, the FT visualizer shown in Fig. 9 represents FTs vertically. In order to avoid the omittance of a failure mode, the standard [17] requires a strict adherence to the immediate cause. ConcertoFLA generates the failure propagation paths, by considering all the failure modes of every port of each component – this guarantees that none of the failures are omitted. According to [17], the repeated events or common cause events should be shown repeatedly in the FT. In our approach, we generate the common cause events repeatedly and could be visualize using FT-Visualizer illustrated in Fig. 9.

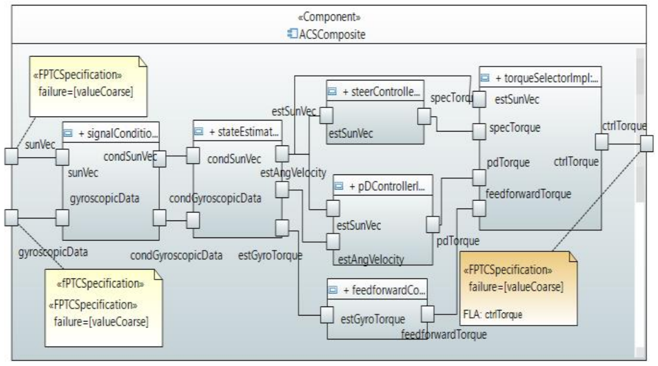


Figure 8. ACS with ConcertoFLA results

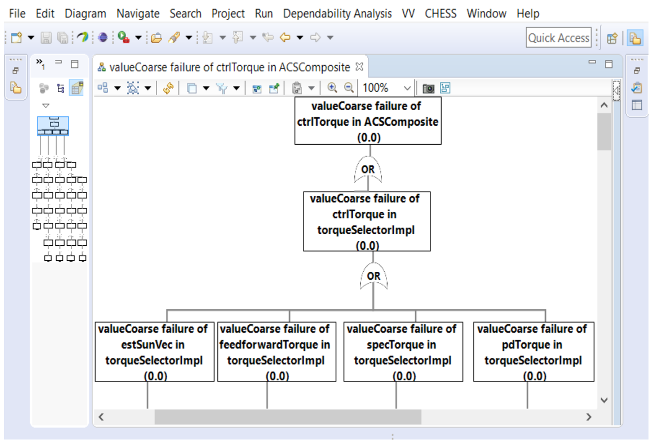

File Edit Diagram Navigate Search Project Run Dependability Analysis VV CHESS Window Help
Quick Access
valueCoarse failure of ctrlTorque in ACSComposite
100%
valueCoarse failure of ctrlTorque in ACSComposite (0.0)
OR
valueCoarse failure of ctrlTorque in torqueSelectorImpl (0.0)
OR
valueCoarse failure of estSunVec in torqueSelectorImpl (0.0)
valueCoarse failure of feedforwardTorque in torqueSelectorImpl (0.0)
valueCoarse failure of specTorque in torqueSelectorImpl (0.0)
valueCoarse failure of pdTorque in torqueSelectorImpl (0.0)

Figure 9. Fault Tree and FT-Visualizer

VI. RELATED WORK

In the literature, several works exist regarding the automatic generation of FTs. These works differ based on system model (given in a different modelling language or given with different perspectives, e.g. structural or behavioural) used as a basis to derive the FT analysis. In [24] a combination of structure and behaviour system model specified via Internal Block Diagram (IBD) and sequence diagram respectively, of the SysML is utilized to derive FT. The system model is first translated to reliability configuration model and then transformed into a FT model. IBD based system model using SysML is also utilized in [18] to automatically generate FT. The approach employs Modelling and Analysis of Real-time Embedded Systems (MARTE) and Dependability Analysis and Modelling (DAM) UML profiles in contrast to our approach which utilizes CHESSML (MARTE and SysML profile) and its dependability profile. In [25], a tool supported approach is proposed to automatically generate FT from the Simulink based system models, using Hip-Hops methodology. In

[26], Formal Safety Analysis Platform (FSAP) along with a safety assessment engine NuSMV-SA (based on NuSMV2 model checker) is proposed – FSAP enables automatic FT construction from NuSMV behavioural models. However, the automatically generated FT consists of cut-sets only and lack the failure propagation paths in comparison to our approach, which generates the FT reflecting the overall system architecture.

I. Conclusion and Future Work

In this paper, we proposed FLA2FT, a novel automatic Fault Tree-generator, which generates FTs from ConcertoFLA results. The automation has the potential to reduce the effort and indirectly contributes in improving the design as well as the dependability requirements, through timely feedback. FLA2FT is integrated in a well-established existing MDE based methodology and tool-set (CHESS). The integration contributes in empowering the toolset, by offering the substantiation of means of compliance for certification.

In the future, we aim at providing an empirical study for establishing the time and cost-effective nature of our approach through conducting case studies that are more comprehensive. In this regard, we aim to involve the students and AMASS [28] partners to conduct experiments, interviews etc.

Acknowledgment

This work has been partially financially supported by the AMASS project [28].

References

[1] European Cooperation for Space Standardization ECSS-Q-ST-30C Space product assurance - Dependability 06/03/2009.

[2] European Cooperation for Space Standardization ECSS-Q-ST-40C Space product assurance – Safety 06/03/2009.

[3] F. Mhenni, N. Nguyen, and J. Choley, “Automatic Fault Tree Generation from SysML System Models”, *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Besacon, 2014.

[4] PolarSys CHESS, https://www.polarsys.org/chess/

[5] L. Montecchi and B. Gallina, “SafeConcert: a metamodel for a concerted safety modeling of socio-technical systems”, *5th Int., Symp., on Model-Based Safety and Assessment*, Trento Italy, 2017.

[6] SysML v1.4 Specification Release September 2015, http://www.omgsysml.org/specifications.htm

[7] B. Gallina, E. Safer and A. Refsdal, “Towards Safety Risk Assessment of Socio-Technical Systems via Failure Logic Analysis”, *IEEE International Symposium on Software Reliability Engineering Workshops*, Naples, 2014.

[8] European Cooperation for Space Standardization ECSS-E-ST-40C Space Engineering – Software 06/03/2009.

[9] B. Gallina, Z. Haider and A. Carlsson, “Towards generating ECSS-compliant fault tree analysis results via ConcertoFLA”, IOP Conference Series: Materials Science and Engineering, 2018.

[10] E. Z. Moreno, Model-based approach for automatic generation of IEC-61025 standard compliant fault trees, Master Thesis, 2018.

[11] PolarSys OpenCert, https://www.polarsys.org/opencert/

[12] D. Cetinkaya, A. Verbraeck and M. D. Seck, “MDD4MS: A model driven development framework for modeling and simulation”, Proceedings of the Summer Computer Simulation Conference, 2011.

[13] Meta Object Facility, https://www.omg.org/spec/MOF

[14] Eclipse Modeling Framework, https://www.eclipse.org/modeling/emf/

[15] Epsilon Transformation Language, https://www.eclipse.org/epsilon/doc/etl/

[16] Internationl Electrotechnical Commission IEC 61025 fault tree analysis (FTA) IEC 61205:2008.

[17] P. Feiler and J. Delange, “Automated Fault Tree Analysis from AADL Models”, ACM SIGAda Ada Letters, 2017.

[18] B. Alshboul and D. Petriu, ‘Automatic Derivation of Fault Tree Models from SysML Models for Safety Analysis’, Journal of Software Engineering and Applications, 2018.

[19] EMFTA Tool, https://gitlab.fbk.eu/CPS_Design/EST

[20] B. Gallina, M. A. Javed, F. Ul Muram and S. Punnekkat, “Model-driven dependability analysis method for component-based architectures” *38th Euromicro Conference on Software Engineering and Advanced Applications*, Cesme, Izmir, 2012.

[21] M. Wallace, Modular architectural representation and analysis of fault propagation and transformation, Electronic Notes in Theoretical Computer Science vol, 141, issue 3, 2005.

[22] CONCERTO Deliverable D3.3 November 2015 Design and implementation of analysis methods for nonfunctional properties – Final version.

[23] R. Noteborn, L. S. Hanbury, R. Larsson, S. Veldman, J. Myatt, M. Pigg, M. Yu, A. Prezzavento, and J. Touaty, “FDIR and robustness for the solar orbiter AOCS”, *9th International ESA Conference on Guidance, Navigation & Control Systems*, Portugal, 2014.

[24] J. Xiang, K. Yanoo, Y. Maeno, K. Tadano, “Automatic Synthesis of Static Fault Trees from System Models”, *IEEE 5th International Conference on Secure Software Integration and Reliability Improvement*, Jeju Island, 2011.

[25] Y. Papadopoulos, M. Maruhn, “Model-based Synthesis of Fault Trees from Matlab-Simulink Models”. *International Conference on Dependable Systems and Networks*, Goteborg, Sweden, 2001.

[26] M. Bozzano and A. Villafiorita, “The FSAP/NuSMV-SA safety analysis platform,” Int. J. Softw. Tools Technol. Transfer, 2007.

[27] AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems). http://www.amass- ecsel.eu