Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Master Thesis

Web Application for Reliability Analysis within Civil Aviation Domain

Bc. Jakub Gruber

Supervisor: Ing. Bogdan Kostov

January 2021

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Gruber Jakub**	Personal ID number:	**456916**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Software Engineering**		

II. Master's thesis details

Master's thesis title in English:

Web application for reliability analysis within civil aviation domain

Master's thesis title in Czech:

Webová aplikace pro analýzu spolehlivosti v letecké dopravě

Guidelines:

FMEA (Failure Modes and Effects Analysis) and FTA (Fault Tree Analysis) are methods of investigation for determining how a product, process, or system might fail. The goal of the thesis is to develop a React web application on the chosen method of investigation that would be extendable for the remaining method. The application should provide an environment to input data for the analysis and visualize its output. Compared to traditional approaches, the application should be based on Semantic web technologies. The project will be done in close cooperation with FMEA (or FTA) domain expert that will validate developed tool on a real use case from the aviation domain (e.g. design of aircraft engine). Core conceptualization of the methods will be provided in the form of ontology built on top of Unified Foundational Ontology.
Instructions:
1) become familiar with Semantic Web technologies to represent (OWL, RDF, JSON-LD) and query (SPARQL) domain knowledge
2) review existing tools for both investigation methods and related graphical libraries
3) analyze requirements for the unified tool of both investigation methods
4) design the unified tool and implement a prototype of the tool on selected investigation method
5) test implemented prototype and validate it with the domain expert

Bibliography / sources:

- Bolčeková S., Reliability Analysis of Mechanical and Lubrication System of Aircraft Engine (2019)
- Kostov, B., UFO Conceptualization of FMEA and FTA (http://onto.fel.cvut.cz/ontologies/apqr)
- Walke, Jordan. 'React-A JavaScript library for building user interfaces.' (2013).
- Brickley, Dan, Ramanathan V. Guha, and Brian McBride. 'RDF Schema 1.1.' W3C recommendation 25 (2014): 2004-2014.

Name and workplace of master's thesis supervisor:

Ing. Bogdan Kostov, Knowledge-based Software Systems, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **07.09.2020** Deadline for master's thesis submission: **05.01.2021**

Assignment valid until: **19.02.2022**

Ing. Bogdan Kostov Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Abstract

Reliability analyses are key components in a risk assessment evaluation during the design phase in an aviation industry. Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) are commonly combined together to review the system and to evaluate possible failures. The combination of methodologies requires a unified data usable for all the analyses. Existing applications provide the tools separately which introduces inconsistencies, duplicates and typos when the data are migrated across the applications.

This thesis thus aims to create an extensible solution that would provide tools to perform one of FTA and FMEA techniques and yet rely on an ontological model usable for both. The thesis analyses existing solutions and ontologies and given these inputs proposes necessary requirements that are prioritized in cooperation with involved domain experts. The resulting solution implements an application focusing primarily on FTA which offers possibilities for system partonomy definition, FTA construction and an automatic conversion of the trees to FMEA tables given the unified ontological model. The application is finally reviewed by the domain experts on real aviation data.

Keywords: reliability analysis, FTA, FMEA, ontology, semantic web, JOPA

Abstrakt

Analýzy spolehlivosti jsou klíčovými složkami při hodnocení posouzení rizik během fáze návrhu v leteckém průmyslu. Analýza stromu poruch (FTA) a analýza poruchových režimů a efektů (FMEA) se běžně kombinují při analýze systému a vyhodnocování možných poruch. Kombinování metodik vyžaduje sjednocení struktury dat tak, aby byla použitelná pro všechny analytické metody zároveň. Existující aplikace poskytují nástroje samostatně, což vede k nekonzistenci dat, duplikátům a překlepům při migraci napříč aplikacemi.

Tato práce si klade za cíl vytvořit rozšiřitelné řešení, které by poskytlo nástroje k provedení jedné z technik FTA a FMEA a přitom se spoléhalo na ontologický model použitelný pro obě techniky zároveň. Diplomová práce analyzuje existující řešení a ontologie a na základě těchto vstupů navrhuje nezbytné požadavky, které jsou ve spolupráci se zúčastněnými doménovými odborníky prioritizovány. Výsledné řešení implementuje aplikaci zaměřenou primárně na FTA, která nabízí definování partonomie systému, konstrukci FTA a automatický převod stromů do FMEA vzhledem k jednotnému ontologickému modelu. Aplikace je na závěr otestována doménovými odborníky na základě skutečných leteckých dat.

Klíčová slova: spolehlivostní analýza, FTA, FMEA, ontologie, sémantický web, JOPA

Author statement for graduate thesis:

Acknowledgements

I would like to thank my supervisor Ing. Bogdan Kostov who has always been very supportive and helpful throughout the whole writing of the thesis. Next, I would like to thank all my colleagues from Faculty of Transportation Sciences who have provided me necessary knowledge to understand the topic and were available for consultations. Finally, I would like to thank my family and my friends for their support and patience.

Contents

Chapter 1

Introduction

Domain experts in the aviation industry rely on risk assessment tools and reliability analyses to conduct studies which help to identify possible issues in the system design that could lead to catastrophic accidents. There are several analyses that can be used, but the ones often employed are Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA). While these techniques work in a reverse direction, their goal is to similarly break the system in pieces which are then analysed either in a top-down or a bottom-up fashion. Identification of failures in an early stage of development saves time, money and resources, yet the analyses are ready to be updated by operational data as well.

To perform the analyses, analysts firstly need to have a definition of the system itself in a graphical representation depicting space dependencies and component partonomies to understand how the issues propagate and what subsystems can be influenced. Furthermore, earlier mentioned FTA and FMEA are commonly combined together to cross-validate themselves and thus need to have a shared data to operate on. Most of existing applications do not allow such interchangeability of the data between the techniques and thus must be copied which introduces inconsistencies in the model basis. In addition to that, system partonomies have to be defined elsewhere and then migrated in analyses tools which emphasises the inconsistency issue.

The thesis of Mrs. Bolčeková [1] has proven that using an ontological approach significantly improves the consistency of the data and reduces duplicate entries within FMEA analysis. The goals of this paper thus will be to propose and implement a solution based on semantic web technologies employing ontological model to create an application combining both FTA and FMEA analyses together on top of a unified model. The solution should fully implement one of the analyses, perform steps for employing the second one and finally be verified by domain experts involved in the application development.

1.1 Structure

The Chapter 2 provides an introduction to a topic of reliability analyses and explains their significance in an aviation domain. We go through both FTA and FMEA techniques, describe their methodologies and suggest a recommended approach how to conduct them.

The Chapter 3 introduces the motivation behind Semantic Web and points out the role of ontologies. Furthermore, it focuses on technologies relied on within this paper and is concluded by description of storage options for semantic data.

Next, the Chapter 4 puts emphasis on existing ontologies within aviation domain and points out their intended usage, strengths and possible weaknesses in case they would be reused as a model ontology for the application.

Chapter 5 sets requirements on graphical libraries that are necessary for visualisation of fault trees and presents several candidates that fulfill the purpose.

The research part is concluded in Chapter 6 by enumerating similar applications, analysing their approaches, functionality to take over and pointing out features to improve.

Given the acquired knowledge, Chapter 7 determines the scope of the thesis by prioritizing functional and non-functional requirements of resulting solution. The system architecture is then proposed accordingly to support requested functionality.

Next Chapter 8 demonstrates the details how the functionality was implemented in the application and what was a solution to encountered issues.

The Chapter 9 then describes how the testing was performed and explains verification process involving domain experts who have reviewed the resulting application.

Finally, Chapter 10 concludes the paper by evaluating results and mentions future steps to improve the application.

Chapter 2

Reliability Analyses

This chapter presents reliability, its importance and the role it plays in the aviation development process. It enumerates and describes selected reliability and risk assessment methodologies which the thesis later builds upon. Later, the methodologies are introduced and their intersection and joint building blocks are highlighted.

Reliability is generally understood as a behavior that is expected from the system. Any deviation from the specification should be avoided and is not wanted. Aviation industry defines reliability with respect to system or a component as a probability that it will perform its function according to specification, under specified operational and environmental conditions during a specified interval [2].

The risk management of aviation systems requires inputs from several disciplines. It provides risk assessment data to support the process and the success lies in the consistency and reasonable assumptions while performing the analyses [3]. To support that, the analyses are based on unified methodologies which are described in the next sections.

2.1 Reliability Methodologies

The most important aspect in an aircraft development process is to ensure the safety. Components and particular subsystems must be designed to prevent catastrophic accidents in case of a failure of a subsystem. The system is designed with redundancies that prevent local failure leading to a failure of the whole system [4]. Reliability engineering provides methodologies whose goals are to [1]:

- apply knowledge to mitigate or to reduce the likelihood or frequency of failures
- identify and correct the causes of failures that occur despite the mitigation
- determine ways of coping with failures that occur, if their causes have not been fixed
- apply methods for estimating the reliability and for analyzing reliability data

[1]https://study.com/academy/lesson/reliability-engineering-definition-purpose.html

The reliability engineering is implemented through reliability analysis process which focuses on acquiring, identifying and organizing system specific data required to make decisions about the system. The objective of the analysis is to construct an overall view over the observed system and to minimize or completely eliminate possible failures leading to life endangering situations [1].

The system overview can be constructed in either bottom-up or top-down manner. The inductive (bottom-up) approach is typical for Failure Modes and Effects Analysis (FMEA) and builds a view of the system by identifying local and later next effect that propagate through the system. Deductive (top-down) approach, whose main representative is Fault Tree Analysis (FTA), first identifies the root failure and then explores the possible causes [5].

The methodologies mentioned in previous paragraph are the main topic of this thesis and will be explained in depth in Section 2.2 and Section 2.3. Needless to say, the reliability topic is too broad and thus only techniques important for this paper are further discussed here.

2.2 FTA

As mentioned in Section 2.1, FTA is a top-down reliability analysis technique, "whereby an undesired state of the system is specified (usually a state that is critical from a safety or reliability standpoint), and the system is then analyzed in the context of its environment and operation to find all realistic ways in which the undesired event (top event) can occur" [6]. As depicted by Figure 2.1, the tree itself is represented as a graphical model which organizes failure paths possibly leading to a top event failure into block components. These components are tied together by logical gates that are adding extra value in form of probability propagation definition of underlying events.

Furthermore, the top event probability can be used as an input to probabilistic risk assessment (PRA) methodology that evaluates risk given the probability and severity.

2.2.1 Building Blocks

The building blocks of FTA can slightly differ based on intended usage and terminology, but the main idea stands in the events and gates being connected together forming the tree shape. Figure 2.2 enumerate visual symbols representing the blocks.

- The Basic event represents an event which is in the leaf of the tree and has explicitly defined probability of failure.

- External event, also called House event in NASA handbook [6], is an event which normally occurs, should be included, but its probability cannot be influenced.

- Conditioning event is used together with Priority And or Inhibit gates. It specifies conditions applied to the gate.

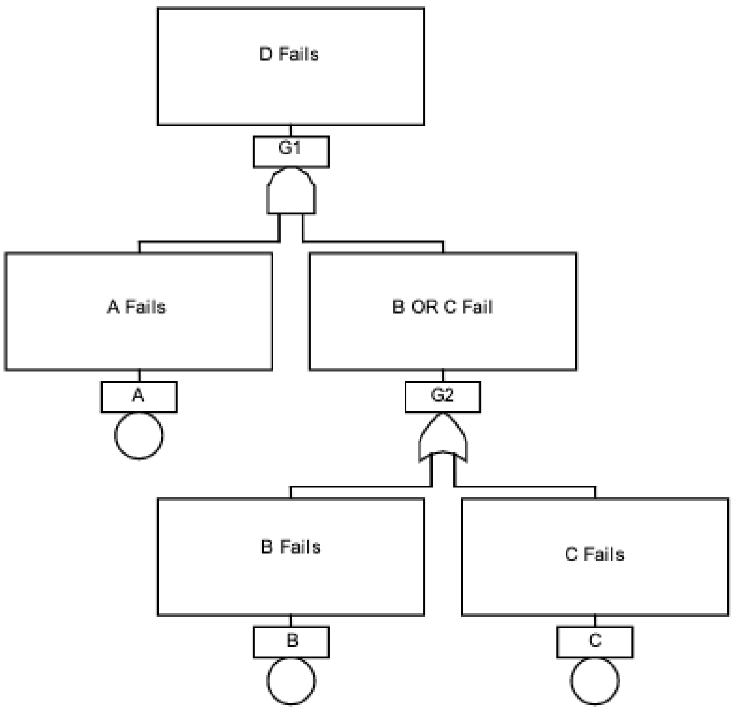


Figure 2.1: Simplified FTA [6]

- Undeveloped event lies in the leaf of the tree same as Basic event, but it is not final, instead of that, the exploration of causing failures was omitted for chosen granularity level.

The goal of reliability analyst is to construct a visual representation of the analysed system by combining the events with the gates whose functions are described in following list:

- OR Gate is a logical gate propagating the probability of input events where at least one has to succeed.
- AND Gate is a logical gate propagating the probability that all input events are successful.
- XOR Gate demands only and exactly one event out of all inputs to succeed.
- Priority AND Gate is a special form of a regular AND Gate. Aside from the input events, the gate needs specification of order in which the inputs are evaluated. The output probability is equal to probability that all inputs fail in given order.
- Inhibit Gate is a special type of AND Gate where Conditional events come handy. They control the evaluation of AND Gate where the output of the gate occurs under the condition specified by conditional event [6].

NASA FTA Handbook [6] further defines two more gate types - TRANSFER-IN and TRANSFER-OUT. These gates are available for use in cases when the fault tree grows

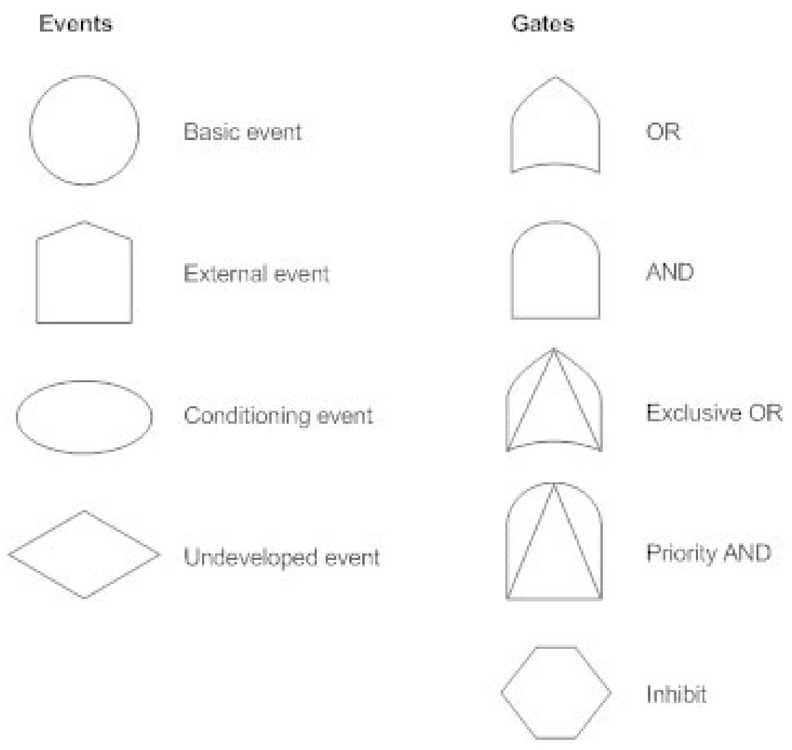


Figure 2.2: Fault Tree Building Blocks [7]

together with the analysed system and the branches become difficult to use. As the name suggest, the gates provide transfers in and out between the trees. In other words, subtrees are imported/exported between fault trees. Needless to say, the resulting probability of the top failure event is not influenced.

2.2.2 Ground Rules

Yet the graphical model of fault tree looks simple, many ground rules have already been established throughout the years of development and practical use to help performing successful fault tree analysis [6].

1. Despite the name Fault Tree, a Success Tree can be created as well by using the same methodology and symbolic. That is a perfectly valid case, but both approaches must not be combined in one tree.

2. Event definition should be precise and environment should be specified.

3. Outputs of the component in normal working state must not be considered as failure modes.

4. The tree construction must alternate events and gates. No event may be connected directly to another event. Same applies for gates. If necessary, insert intermediate events.

An important addition to construction of the system view states that on the contrary to FMEA discussed in upcoming Section 2.3, FTA boundaries of the analysed system are imaginary and the tree may be related to multiple systems. The faults are not tied neither to components nor functions. The statement further means that FTA manifests the failure probability of top event, but it lacks information about which subsystems were influenced [6].

2.2.3 Construction Process

The correct construction of the tree is a key to success. In the Subsection 2.2.2 earlier, ground rules for tree construction were established. But following the rules, is only one part of the process. As denoted in Figure 2.3, the complete process consists of 6 (or more precisely 8) steps, but it can be simplified for purposes of this thesis into 3.

Firstly, analysis scope and ground rules are established. Secondly, the tree itself is constructed into a graphical model. Last but not least, the results are evaluated.

Based on findings of Mrs. Bolčeková in her thesis [1], a process of information collection is followed by manually inserting the data in the systems which results in data duplicity and inconsistencies, thus this paper will also focus on preceding step of initial data collection.

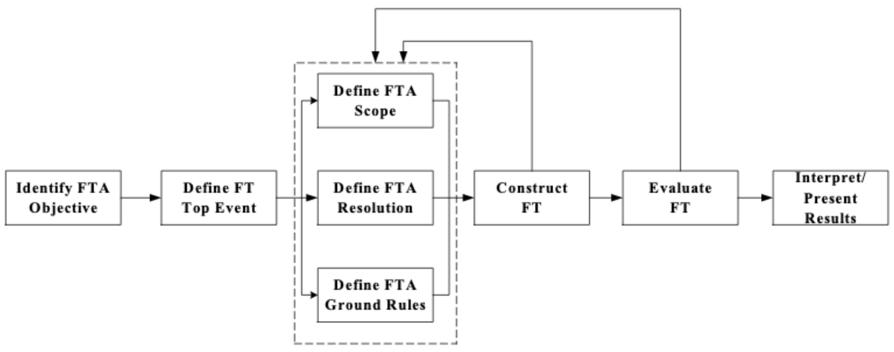


Figure 2.3: FT Construction Process [6]

2.3 FMEA

FMEA is another representative of reliability analyses technique which focuses on an early identification of failure modes and an assignment of mitigation actions. As opposed to a deductive FTA, FMEA is inductive technique modelling the system in a bottom-up manner by gradually exploring further failure modes until it reaches desired granularity [8].

As opposed to the graphical visualisation of FTA model, FMEA structures its content into a tabular form. The form of the table is not strictly declared, but we will analyze an example of FMEA table in Figure 2.4 and later add necessities used in this thesis.

The content should begin with the table header with general information about author and analysed system.

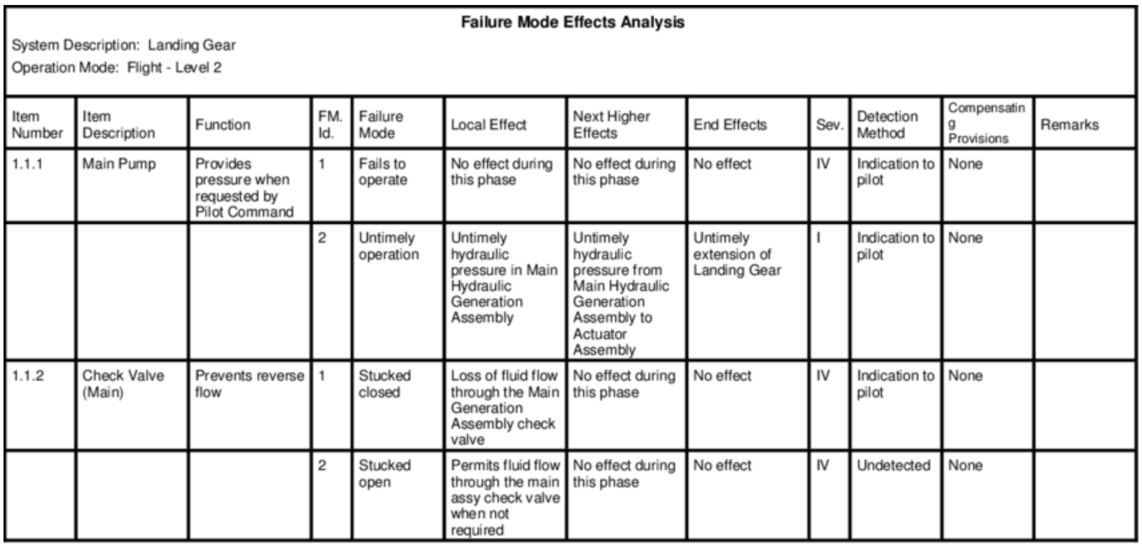
Firstly, the table should begin with columns describing analysed items, their functions and optionally assigned identifiers.

Secondly, failure modes, their local, next higher and final effects have to be defined. The analysts then estimates severity (S), occurrence (O) and detection (D) of such path. SOD scale differs based on the environment and company practices and can be either verbal or numeric. The important to note here is that the verbal definition of SOD should use only enumerated, e.g. low, medium, high. Second option with numeric values on a scale from 1 to 10 will be used throughout this paper as it will further be used to calculate a Risk Priority Number (RPN).

$$RPN = S * O * D \tag{2.1}$$

Furthermore, each row in the table may have assigned mitigation processed and actions that have already been taken.

Last but not least, RPN may be reevaluated by analysts once again with respect to taken actions and recommended mitigation processes.



Failure Mode Effects Analysis

System Description: Landing Gear

Operation Mode: Flight - Level 2

Item Number	Item Description	Function	FM. Id.	Failure Mode	Local Effect	Next Higher Effects	End Effects	Sev.	Detection Method	Compensating Provisions	Remarks
1.1.1	Main Pump	Provides pressure when requested by Pilot Command	1	Fails to operate	No effect during this phase	No effect during this phase	No effect	IV	Indication to pilot	None	
			2	Untimely operation	Untimely hydraulic pressure in Main Hydraulic Generation Assembly	Untimely hydraulic pressure from Main Hydraulic Generation Assembly to Actuator Assembly	Untimely extension of Landing Gear	I	Indication to pilot	None	
1.1.2	Check Valve (Main)	Prevents reverse flow	1	Stucked closed	Loss of fluid flow through the Main Generation Assembly check valve	No effect during this phase	No effect	IV	Indication to pilot	None	
			2	Stucked open	Permits fluid flow through the main assy check valve when not required	No effect during this phase	No effect	IV	Undetected	None	

Figure 2.4: FMEA Table [9]

To sum up, each row analyses one failure mode, mentions its influence, propagation path, risk information and recommendations to suppress potential damage. The technique itself provides a uniform method to design a safer system with an early identification of a Single Point of Failure (SPOF) and generates an additional information for ranking by RPN [3]. FMEA exists in several variants based on the phase of development process - System FMEA, Design FMEA, Process FMEA, Defect FMEA or Functional FMEA. It was also extended by a criticality analysis FMECA.

2.3.1 Construction Process

Based on recommendations from NASA JPL, FMEA consists of 6 essential steps [3]:

1. Construct an overview of the functional dependencies in the system. Use Reliability Block Diagram or other modelling technique.
2. Identify failure modes for each involved component.
3. Perform FMEA for all components in step 1.
4. Properly identify and index systems and subsystems.
5. Manage and update critical items list from steps 1 to 3.
6. Documentation step - analysis appendices and assumptions.

The first two steps can further be broken into a more specific processes [10]. The analyst initially collects system information and constructs an overview to discover functional dependencies. The modelling of the overview is up to personal choice, but common approaches are e.g. Reliability Block Diagram (RBD) or an aggregation diagram written in UML. The FMEA table header is annotated with a common analysis information such as author, analysed system or start date. Given the system overview, each component is evaluated to identify possible failures, its effects according to failure propagation path and sub-system influence. This constructs initial part of the FMEA row. The row is then assigned an RPN number according to severity, occurrence and detection of the failure mode. If necessary and possible, mitigation with description how the RPN can be lowered is proposed.

2.4 Methodologies Overlap

In previous sections we have described a basic overview of reliability analyses, FTA and FMEA. While the visual side of the methods looks different, they have a lot of in common and are both combined in practice to achieve better results. The techniques focus on an early identification of failures in the system, lowering their probabilities and optionally propose a mitigation and recommended actions.

While the FTA puts emphasis on the top event and with the visual guide helps an easier discovery of reasons of failure, FMEA on the other side tries to distinguish failure modes effects on the components and their functions. Furthermore, it catalogs all failure modes and generates risk priority numbers used in prioritization. Given that, the methods are combined, because FTA scope may be cross-system and lacks information about the influence and FMEA is displayed without the visual guide, requires more focus to understand complex systems and does not support PRA.

That being said, as we could see in Subsection 2.2.3 and Subsection 2.3.1, everything stands and falls with the initial data collection and experience of the analyst. Beginning with an incorrect model, the analyses will either lack or duplicate events which will lead to impossibility of proper document management and updating by operational data later on[1]. To support that, methodologies should preferably begin with the same initial diagram that would support both of them and unify used system information. In the best case, model unification offers a possibility to combine and interchange underlying structure to work on both techniques simultaneously.

To conclude the methodologies overlap, while it might look as an option, despite FMEA flat-like structure and focus on single events with a reference to next effects, FMEAs cannot be combined to construct an FTA. The FMEA primary failures are not selected consistently as in FTA and the successors lack any logical relation and must not necessarily be direct one, thus the output tree could be incomplete, misleading or displaying wrong overview of the structure [6]. Yet, as mentioned earlier, the methods work well together and are often combined to cross-validate each other.

Chapter 3

Ontologies and Semantic Web

The chapter goes through the concepts of ontologies, explains the motivation and importance of semantic web and briefly introduces the technologies that stand behind it. Furthermore, it covers main transportation and representation formats, their usage in graph databases and methods how to query such structured data. In the end, the concentration is put on JOPA library that efficiently accesses graph databases and is used throughout this paper.

3.1 Ontology

The term of ontology originates from the Greek compound of words *onto* and *logia* which together mean "study of being". Information Science borrows the term and uses ontology as a formal specification to represent knowledge [11]. The specification could be divided in two parts - *conceptualization* and *formalization*.

3.1.1 Conceptualization

The overall goal of ontology is to create a conceptual model that stands on a shared vocabulary, specifies entity relations and presents axioms (more on that later) that are all together logically consistent. Such model is independent from implementation and specific technologies and offers a better understanding of communication between computer agents and users. The process of constructing such model is called *conceptualization*. Unlike database logical and physical models, the ontological model is said to be on a semantic level due to its similarity to first-order languages [11].

The ontological approach is thus beneficial in reusing domain knowledge, making it explicit and separating it from the operation knowledge [12].

The main building components of the ontologies are a little similar to an object oriented approach - *classes, attributes, relations*. It is then extended by a concept of *function terms, restrictions, rules and axioms*.

3.1.2 Formalization

The rules and axioms are terms that allow *formalization*. Yet the conceptualization brings in definition of relations, the formalization takes it further and works with the concept of rules and axioms. These terms are typically used to describe formal semantics of the language in the context of predicate and description logic. They can be understood as assumptions that are always expected to be true.

Both rules and axioms represent ways to formally specify the semantics. While rules are constructed in *if-then* form, the axioms are based on predicate logic, thus they can be defined mathematically.

Nevertheless, by introducing these terms, the ontologies can then perform a process of *inference* [13]. The inference starts with an ontological model that specifies a vocabulary enumerating concepts, their properties and specification of relations. Based on that, a *semantic reasoner* then traverses the datastore and infers logical dependencies between concepts by which new facts are generated and uncovered. These facts are inferred according to rules and axioms.

The semantic reasoner can either derive new facts in a forward or a backward fashion, thus called *forward chaining* or *backward chaining*. The difference stands in a process how the inferred data are derived. Forward chaining starts with a set of facts and infers new facts applying the rules until a goal is reached, backward chaining on the other side starts with the goal and applies rules to discover known facts.

An important difference to highlight here regarding the rules and axioms is that, during an inference process, while an evaluation of axioms is independent of their order, the inference according to rules may produce different results if they are evaluated in a different order.

3.1.3 Design

The design must be produced in an iterative way and yet there is no correct or incorrect way how the proposed ontology should look like. Despite of that, it should still provide a representation close to real world objects and relationships [12].

The following list suggests an approach how to iterate [12] [10]:

1. Scope determination

   Decide how deeply will the domain be covered. Which properties will be used and which are insignificant. Decide who should maintain the ontology and what answers should it respond to. Think ahead and try to come up with a solution covering wider area of interest.

2. Consider reusing existing ontologies

   As mentioned in the step above, the solution does not necessarily need to cover only personal interests but might be design for others as well. With that being said, make a research about existing vocabularies and think about an extension to those.

3. Highlight important components

   As a continuation to the first step, analyse deeper the ontology focus, requirement and try to be more specific about the classes and properties. Enumerate allowed values.

4. Define hierarchy

   Finally, with the determined scope and important components, construct class hierarchy, assign properties, create instances.

To support the ontology development, developers can rely on other methodologies as well. The NeOn Methodology[1] emphasises reuse of ontological resources and offers guidelines on specification requirements. Rapid methodology on the other side focuses on, as the name suggest, rapid ontology development and reminds an agile methodology from software engineering. It proposes 6 steps that, by enriching the previous one, finally compose the ontology [14]. The last representative of methodologies is Systematic Approach for Building Ontologies (SABiO) which relies on the Unified Foundational Ontology (UFO) described in next section [15].

The ontology design will be concluded by recommendation of ontology design tools. Commonly used tools to design ontologies are e.g. *NeOn*[2], *Protégé*[3] from Stanford University or *Menthor*[4] editor for UFO based UML class diagrams.

3.1.4 UFO

The process of designing of the ontology can be further supported by using unified concepts that aim to define a real world in ontological terminology and relations. The models allow to describe concepts in a domain of interest using well established modeling patterns. One of these top-level ontologies is the Unified Foundational Ontology (UFO) presented by Giancarlo Guizzardi in 2005 [16]. The UFO lays the foundations of fundamental concepts in conceptual modeling.

It categorizes concepts into categories, defines formal relations in between and specifies axioms for verification. The Guizzardi's paper [17] divides the model in three main categories:

- UFO-A: ontology foundations. Focuses on *endurants*.
- UFO-B: extension of UFO-A. Focuses on *perdurants (events)*.
- UFO-C: extension of UFO-B. Focuses on *"the spheres of intentional and social things, including linguistic things."* [17]

[1] mayor2.dia.fi.upm.es/oeg-upm/index.php/en/methodologies/59-neon-methodology
[2] http://neon-toolkit.org/wiki/Main_Page.html
[3] https://protege.stanford.edu/
[4] https://github.com/MenthorTools/menthor-editor

To explain the used terms, *endurant* characterizes an object in a real world. While its behavior or properties might change in time, the endurant itself stays the same.

On the other side, *perdurant* represents events and processes. It is composed of temporal parts and happens in time. It might accumulate temporal parts and thus change itself and be extended by the parts. Examples represent e.g. conversation or a symphony execution [17].

Furthermore, the *endurants* are categorized in *objects* and *moments*. The moments are dependent on other object and this existential dependency is call *inherence*, thus the property is called *inheres* [17]. The relationship of endurant, object and moment[5] is depicted in Figure 3.1.

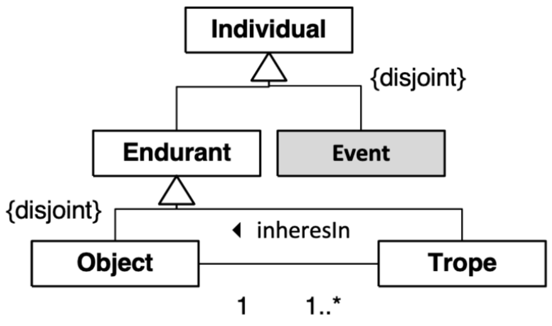


Figure 3.1: Relationship of Endurant, Object, Trope [17]

The moments further define terminology for their properties as *qualities* and *dispositions*. The qualities describe quantifiable properties of objects e.g. color and strength of a magnetic field, whereas dispositions characterize a potential of object to be involved in events, e.g. "the disposition of a magnet to attract metallic material" [17].

Figure 3.2 shows an excerpt of UFO patterns that are related to description of events and their relationships to other individuals in this thesis. Note that UFO also allows to describe metamodels (multi level modeling) using the so called powertypes, i.e. types whose instances are domain types (e.g. events in a fault tree represent types of events which may occur during system operation).

3.1.5 Types

According to the usage and a data the ontologies hold, they can be classified into a hierarchical structure. The top category is a *top-level ontology* that defines concepts like space, time, matter, etc. These information are essential and thus contained in all categories below. Subsequent types are called *domain* and *task ontology*. They relate to a vocabulary for a specific domain or a task execution. Last but not least is the *application ontology* which is a specialization of all types mentioned earlier and represents both concepts depending on a domain and a task ontology [18]. Following Figure 3.3 depicts the hierarchy.

[5]moment and trope are used as synonyms

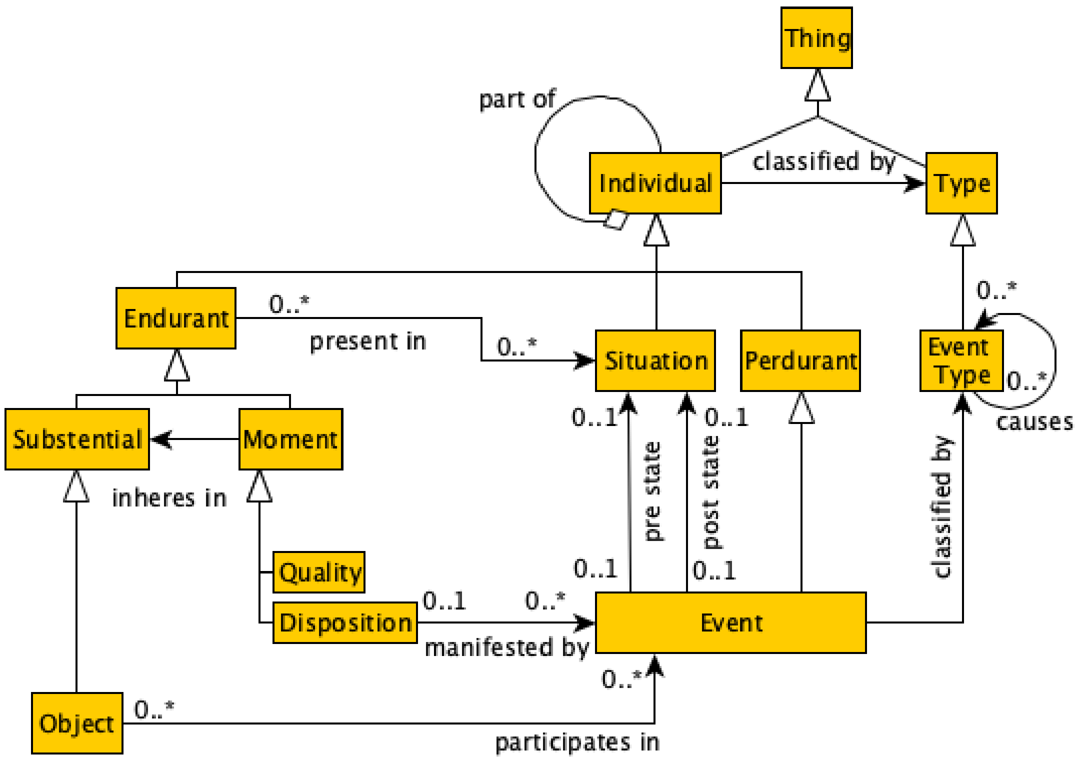


Figure 3.2: UFO ontology excerpt

The concept of ontologies and its knowledge base reusability are building blocks in the Semantic Web described in Section 3.2.

3.2 Semantic Web

With the increasing usage of the Internet, an average user produces 1.7MB of data per second. That means every day, the mankind generates 2.5 quintillion bytes [6]. To maintain and find useful information in such large amount of data, it has to be structured in a systematic way. It is critical so that applications generate data according to an agreed vocabulary and thus offer a possibility to be further processed by automated agents.

The Semantic Web is an initiative by W3C aiming to make the Internet data machine-readable so that computers are interconnected together and understand the data produced elsewhere. In the heart of the infrastructure stands *Linked Data* as a collection of technologies that share same vocabulary, define relations and can be queried [19]. In a simplified way, the goal is to enrich the data by their semantics and link it together. The Semantic Web is sometimes referred to as "Web 3.0".

The architecture of the semantic web is depicted on Figure 3.4 and the technologies are further analysed in following subsections.

[6]https://techjury.net/blog/how-much-data-is-created-every-day

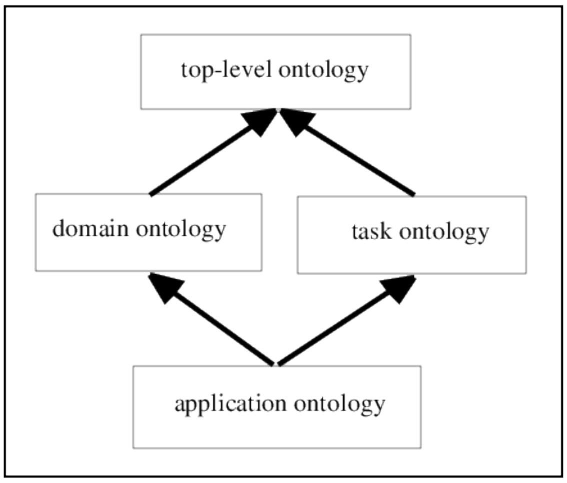


Figure 3.3: Ontology Types [18]

3.2.1 RDF

As discussed in Section 3.2, the information published on the web need to linked together to open possibilities of automated machine processing. To do so, the resources have to be identifiable and labelled.

The Resource Description Framework (RDF) is a framework that illustrates how to interlink published data. It builds the logic on so-called *RDF Triples* that remind real world relations and brings this approach to World Wide Web to make statements about resources. The triples "expresses relationships between two resources" [21]. The structure of a simple triple can be seen below.

$$< Bob >< likes >< Alice > \tag{3.1}$$

The statement holds an information about relationship between two resources - *Bob and Alice*. The relation is explained by the *likes* predicate.

By using such a fairly simple principles, the resources can be interlinked to form a graph representing their connections in a complex system. The graph consists of resources depicted as nodes and predicates visualized by edges. The resources are distinguished by a URI which is a more general identifier than URL commonly used on web.

3.2.2 RDFS

While RDF provides a basic elements describing a formal representation of the triples in its namespace *rdf:*[7], such as *Statement*, *subject*, *predicate*, *object*, *type* and more, RDF Schema (RDFS) defines the schema for RDF graphs by publishing a definition of taxonomy for hierarchical structures. The hierarchy introduces *classes*, *subClasses* and can be assigned *properties* or *subProperties*. In addition to that, properties are enriched with *domains* and *ranges* to illustrate which resources they can refer to [20]. Yet the RDFS extends

[7]http://www.w3.org/1999/02/22-rdf-syntax-ns#

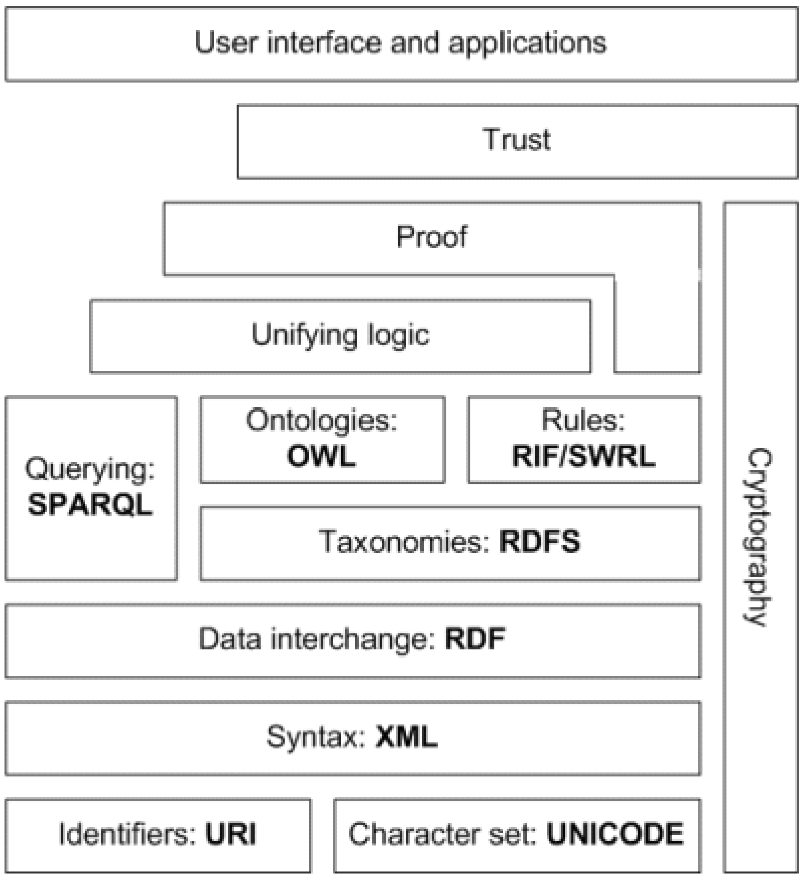


Figure 3.4: Semantic Web Layers [20]

RDF with hierarchy, it does not allow restricting the values according to enumerated options or to form advanced relations.

3.2.3 OWL

The issue of an additional rules, validation and content restriction is targeted by the Web Ontology Language. As was explained in Figure 3.4, the OWL uses underlying layers of RDF(S). The layers are enriched by elements restricting logical relations between resources, allowing combining of classes to e.g. *unions*, *intersections*, *complements*, or to enumerate allowed values with *someValuesFrom*. All possibilities are published here [22].

Furthermore, OWL has also a new OWL2[8] version which is fully compatible with the OWL that was just described, it adds new features e.g. *"local reflexivity" (classes related to themselves)* and is backed by SROIQ(D) decidable description logic formalism.

3.2.4 Serialization Formats

The previous sections introduced a way how RDF, RDFS and OWL can enrich any published data with a machine readable metadata. However, the initial data formats and means of transportation were omitted. It is understandable that bounds on formats still

[8]https://www.w3.org/TR/owl2-new-features

exist. This subsection will thus specify common serialization formats.

1. RDF/XML

   RDF/XML is an original serialization format where RDF, RDFS, OWL statements were added to element tag together with its schema. Despite being the first, its popularity is falling due to a new options being introduced. The main advantage stands in already existing XML parsers that only need to be extended to understand RDF, RDFS and OWL statements.

2. N-Triples

   As the name might suggest, N-Triples are representing the triples in format most similar to *subject predicate object* approach. Each line consists of a subject, predicate and object and is terminated by a dot. While being quite verbose, it can be transferred by a stream of lines which increases efficiency. It is robust to data losses or rearrangement of the file content because each line is independent.

3. Turtle

   A disadvantage of N-Triples is definitely in its verbosity and a possibility that the file content will not be properly structured. The statements about a single resources can be spread across the whole file which decreases readability and maintainability. Turtle format addresses this issue and structures resources and statements into logical blocks. The initial line contains an identifier, followed by a statement per line separated by a semi-colon and the last line terminates the block by a dot similarly to N-Triples.

4. JSON-LD

   JSON-LD is a representative of a modern serialization format compatible with commonly used JSON in Javascript. An advantage is that the format is understood by many programming languages similarly to RDF/XML, yet the readability is not as simple as in case of Turtle. It exists in several document forms - Expanded, Compacted, Flattened and Framed. Given the fact that it is basically a JSON with special attributes, it is typically used as a communication protocol in HTTP communication.

The following figures compare how *a person named James Wales with an email address jwales@bomis.com and a nickname Jimbo* can be written in the enumerated formats using a Friend of a Friend (FOAF) ontology.

Listing 3.1: RDF/XML

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:Person rdf:about="http://xmlns.com/foaf/0.1/">
    <foaf:name>James Wales</foaf:name>
    <foaf:mbox rdf:resource="mailto:jwales@bomis.com"/>
    <foaf:nick>Jimbo</foaf:nick>
  </foaf:Person>
</rdf:RDF>
```

Listing 3.2: N-Triples. FOAF and RDF namespaces shortened

```
<...foaf/0.1/> <...rdf#type> <...foaf/0.1/Person> .
<...foaf/0.1/> <...foaf/0.1/name> "James Wales" .
<...foaf/0.1/> <...foaf/0.1/mbox> <mailto:jwales@bomis.com> .
<...foaf/0.1/> <...foaf/0.1/nick> "Jimbo" .
```

Listing 3.3: Turtle

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

foaf:#JW
  a foaf:Person ;
  foaf:name "James Wales" ;
  foaf:mbox <mailto:jwales@bomis.com> ;
  foaf:nick "Jimbo" .
```

Listing 3.4: JSON-LD (Compacted form)

```
{
  "@id": "http://xmlns.com/foaf/0.1/",
  "@type": "http://xmlns.com/foaf/0.1/Person",
  "http://xmlns.com/foaf/0.1/mbox": {
    "@id": "mailto:jwales@bomis.com"
  },
  "http://xmlns.com/foaf/0.1/name": "James Wales",
  "http://xmlns.com/foaf/0.1/nick": "Jimbo"
}
```

3.3 Triplestores

The RDF in Subsection 3.2.1 has introduced a new approach how to view and understand information. Such information could be stored in a classical relational databases, e.g. by creating a table with subject, predicate, object columns, but it is obvious that such approach would not be optimal due to indexing, handling of null values in predicate columns or simply for resources with several properties. It would quickly violate normal forms of relational databases.

Instead, semantic data are persisted in *RDF Triplestores* that are a specialized types of graph databases where the resources represent nodes and relations edges. The data form a network of objects, which is flexible, indexable and can be traversed by graph algorithms in an efficient way.

Another advantage lies in a support for automatic reasoning (inference) which is typically executed according to rule based systems. The downside of the reasoning in triplestore is that the semantics of OWL stored in triplestores are compromised as the rule based systems cannot capture the expressivity of SROIQ(D) formalism backing up OWL2.

Representatives of RDF Triplestores can be e.g. RDF4J, GraphDB or Apache Jena TBD.

3.3.1 SPARQL

Similarly to SQL for relational databases, triplestores have own query language optimized for data retrieval. It is a semantic query language formulated by W3C. SPARQL is used to execute queries on top of RDF triplestores that result in graph patterns supporting aggregation, subqueries and other graph algorithms [23].

The language either queries a regular triple or a user is interested in a more complicated query that implements a pattern matching in the graph. That produces a query results corresponding to all subgraphs of the base graph.

Basic query forms are enumerated in a following list [23]:

- SELECT - queries and searches values as they were stored in the store. Returns a relation.
- CONSTRUCT - returns a graph constructed based on a template provided to query.
- ASK - test existence of a solution.
- DESCRIBE - retrieves a graph data about specified resources. The result is not standardized and depends on the triple store implementation.

What makes SPARQL so powerful in comparison to SQL is that it can firstly access relational databases with an appropriate middleware as well and furthermore, the query execution is not bound to a single database, instead it can operate across distributed sources of non-uniform data [24].

A very simple SPARQL query illustrating a syntax of selection is shown in Listing 3.5. The select[9] shows a query *retrieving names of two persons X, Y where person X knows person Y and if person Y has defined a nickname, it is retrieved as well.*

Listing 3.5: SPARQL SELECT

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY ?nickY
WHERE
  { ?x foaf:knows ?y ;
       foaf:name ?nameX .
    ?y foaf:name ?nameY .
    OPTIONAL { ?y foaf:nick ?nickY }
  }
```

3.3.2 JOPA

Despite SPARQL flexibility explained earlier, it still manipulates statements only from an RDF point of view and does not reflect modern paradigms of Object Oriented Programming (OOP), thus the query results remain in a relation. The maintenance of application data and their alignment with an underlying ontology requires developing another layers providing an extra abstractions [25].

Researchers from the KBSS Group[10] from the Czech Technical University have developed a JOPA framework inspired by Java Persistence API (JPA) well known for its Object-relational Mapping where the classes with properties defined as Java classes are, with a help of annotations, automatically persisted and queried from relational databases.

The JOPA focuses on object-ontological mapping (OOM) which guards an integrity of Java model with respect to an underlying ontology [26]. It profits from a similarity to JPA API thus the developers are familiar with the concepts of mapping and a migration is smoother. Classes and their properties are annotated as well, only instead of JPA's *@Entity* and *@Column* it is replaced by *@OWLClass* and *@OWLDataProperty* or *@OWLObjectProperty*. These annotation are mapping the model to an OWL representation and manage integrity according to schema.

Furthermore, JOPA adopts a concept of *EntityManager*, *PersistenceContext* or *@Transactional* annotations. The solution is designed in a modern way that counts with possibilities that a new stores will appear, thus it declares the OntroDriver API [26] that, when implemented, allows connection to a new triplestores without a need to change upper layers of persistence logic. An entry point communicating with OntoDriver is the EntityManager that, similarly to JPA, implements standard CRUD database operations or allows an execution of a native SPARQL queries together with a custom logic for extraction of result sets.

[9]https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#select
[10]https://kbss.felk.cvut.cz/web/kbss/home

On top of that, the framework supports inference through *@Inferred* annotation. When a field is annotated with *@Inferred* it becomes read only and its value is automatically set by a semantic reasoner. What needs to be counted with on the other side when modelling the ontology is the fact that JOPA currently does not support multiple inheritance thus the classes can inherit only from a single super class being an *@OWLClass* [25].

To conclude, JOPA offers a high-level API that, despite being tied only to Java, presents a framework allowing fast and easy-to-maintain option how to connect server-side application with RDF stores.

Chapter 4

Existing Ontologies

The decision which ontology to use influences the whole knowledge base and a resulting application thus it must be chosen wisely. Subsection 3.1.3 mentions a recommended approach for designing ontologies where one of the steps states that analyst should consider reusing an existing ontology. This chapter thus briefly mentions ontologies focusing on reliability in aerospace, aviation and factory automation industry, analyses each of them and summarizes what are the meanings of their classes and their intended usage.

4.1 NASA JPL's Fault Management Ontology

The ontology to begin with is a model proposed by NASA JPL analyst Jean-Francois Castet that was created as a part of *JPL's Fault Management ontology*. The paper [5] explains how the ontology works in a model-based environment for carrying out reliability analyses for a spacecraft domain. Given the model, both FTA and FMEA can be generated by a set of transformation algorithms, converted by plugins to a desired format and finally examined in MagicDraw visualiser[1].

Figure 4.1 depicts an excerpt from JPL's model which extends JPL's Behavior ontology. Note that the figure shows only an excerpt thus not all later mentioned concepts are present. While the blue parts represent the fault management ontology, the black ones are inherited from the behavior one. The important concepts to focus on are *Component (not in figure)*, *Function*, *ElementBehavior*, *ViolationExplanation*, *CauseExplanation* and *ModeContext*.

In a simplified way, these concepts are inputs to a transformation algorithms that generate resulting methodology outputs. As an example, *FailureMode* used in FMEA can be derived by a combination of *ViolationExplanation*, *ElementBehavior*, *CauseExplanation* and *ModeContext*. Similar transformations can be applied during a construction of FTA hierarchical structure as can be seen in Figure 4.2. Violation Explanations (VE) and Behaviors (BE) are combined together and with Cause Explanations (CE) to form an FTA OR gate. By amending the dependencies between effects, gate is capable of automatically

[1] https://www.nomagic.com/products/magicdraw

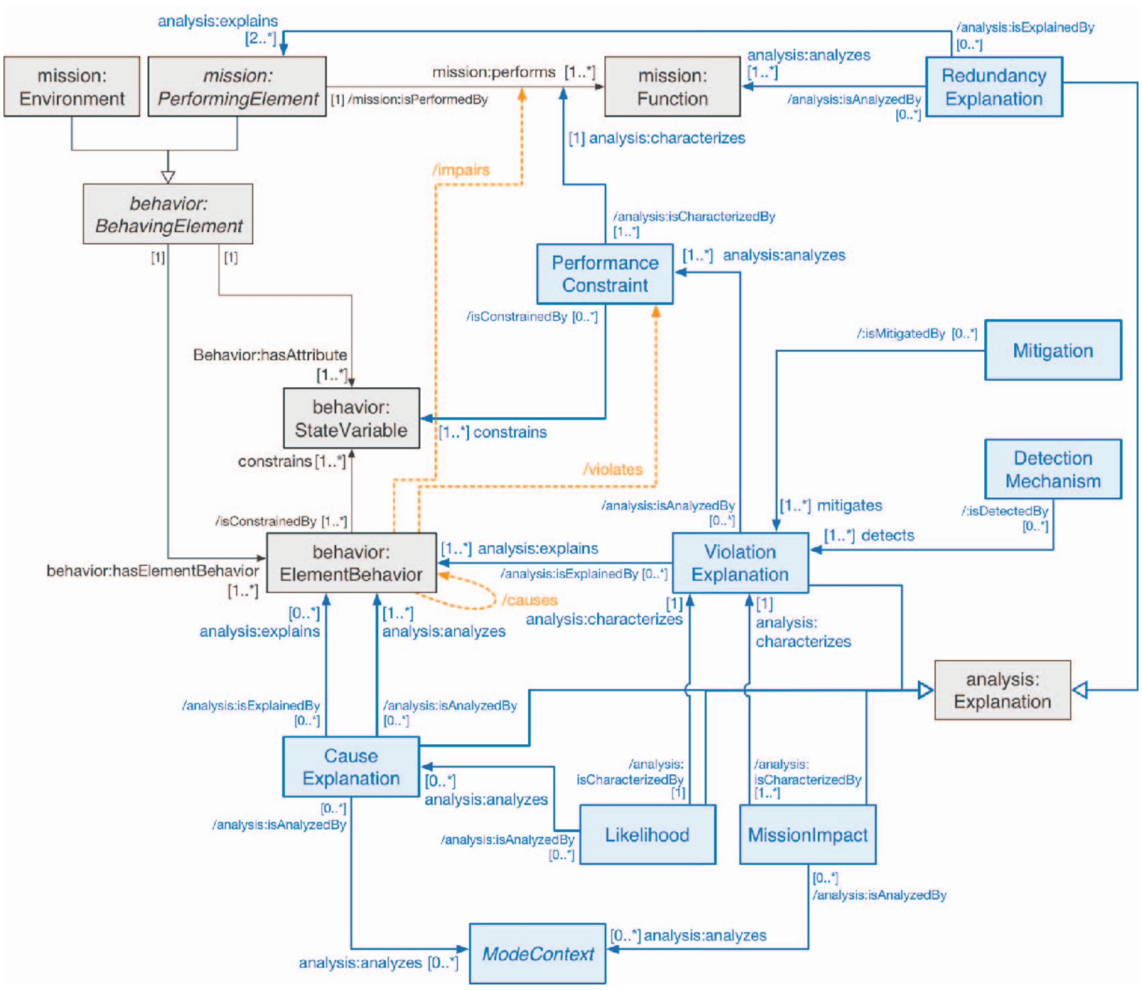


Figure 4.1: Fault Management Ontology Excerpt [5]

changing its type [5]. Much more transformations are executed to create the resulting tree, but only a few are mentioned to get a grasp how it works.

The Fault Management ontology is a well thought ontology with a target audience in spacecraft industry, yet it can be with small modifications reused in an aviation domain. Its advantage lies in its compatibility with both FMEA and FTA analysis. What needs to be counted on though is that despite the model is well thought, some relationships are fairly complex for simpler use cases and some concepts could even be omitted, e.g. *Mission* which is an example of spacecraft-domain related entity. Nevertheless, it proposes an interesting relationships and might serve as an inspiration when defining an application ontology.

4.2 Computer-Aided Fault Tree Ontology

This ontology was presented as a part of Emerging Technology and Factory Automation (EFTA) in 2014. It explicitly focuses on an automatically generating an FTA model given an ontological basis and builds its ideas on a work of A. Majdara from 2009 where he has presented similarly named paper "A New Approach for Computer-Aided Fault

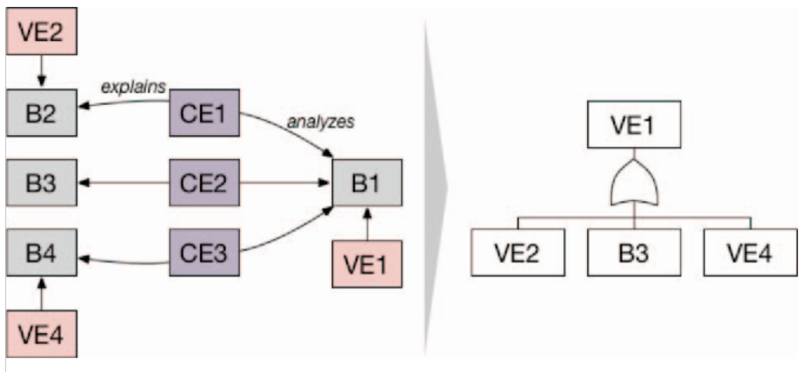


Figure 4.2: Deduction of OR Gate [5]

Tree Generation" [27]. The Majdara's work introduces an algorithm that is given *Components, Function Tables and State Transitioning Tables* and constructs an FTA. The ontological model [28] migrates the idea of tree construction to an ontological world.

The first concept used in the algorithm is a *Component* that, as the name suggests, defines a component in the system. Components are connected forming a network that represents a system. Secondly, each component is assigned a *Function Table* that has *inputs*, *functionality condition* and *outputs*. It shows which outputs are produced if a specific inputs are given and an internal functionality condition holds. Last but not least, a chosen components are assigned a *State Transition Table* that might remind UML State Diagram in a tabular representation. It starts with a *First state*, *Command*, *Functionality Condition* and ends based on these information in a dedicated *Final State*. An example of both types of tables is shown in Table 4.1 and Table 4.2 demonstrating a simple manual switch [28].

Input	**Functionality Condition**	**Output**
0	-	0
-	Close	0
1	Open	1

Table 4.1: Function Table for Manual Switch [27]

To briefly outline the algorithm, firstly a top event to examine is chosen. An associated component is then found and the process iterates through all remaining components within system boundaries by a set of jumps into and from both table types. Once all combinations are explored and all gates and events are fully developed, it stops. To finalize the process, the resulting tree is updated by several modifications such as re-encountered events removal to match FTA recommendations [27].

The advantage of EFTA solutions stands out in its separation of system modeling and automated tree generation. Furthermore, the tables are easily extensible and provide an additional overview of the data which is always welcome. On the other side, it focuses only on FTA and does not have any cross-over to FMEA. Furthermore, filling in informa-

First State	**Command**	**Functionality Condition**	**Final State**
Open	Close	OK	Close
Open	Close	Fail-to-Close	Open
Open	Open	-	Open
Close	Open	OK	Open
Close	Open	Fail-to-Open	Close
Close	Close	-	Close
Open	No command	-	Open
Close	No command	-	Close

Table 4.2: State Transition Table for Manual Switch [27]

tion about complex system in the tables is very time demanding if done correctly, as was shown in Table 4.2 for a simple manual switch.

4.3 FMEA Ontology

The last model considered is an FMEA ontology proposed by Simona Bolčeková in her diploma thesis focusing on a reliability analysis of mechanical and lubrication system of an aircraft engine. She has shown that an FMEA conducted on an unstructured data has issues with information consistency, shareability and reusability [1]. It was overcome by a usage of ontology that is based on previously mentioned NASA JPL's ontology from Section 4.1. With a small modifications, it was adjusted to better match FMEA naming conventions to keep it aligned with aviation domain terminology.

The model is depicted in Figure 4.3 and shows the conceptual structure of the classes and relationships. As mentioned earlier, this proposal is tailored to an FMEA technique.

Each *FMEA* focuses on a *Component* that can have several *Functions* and *FailureModes*. The *FailureMode* then together with its mitigation, RPN and a propagation path represents a single line in the table whose structure was commented in Section 2.3.

While this approach is tied only to FMEA, it simplifies the JPL's ontology introduced first, removes its complexity and is extensible by FTA-specific concepts only with a minor modifications needed. The extensibility was presented in a thesis focusing on usage of conceptual models for FTA [29].

To briefly conclude just introduced existing ontologies, it is clear that each one has a specific target usage and it needs to be counted on. NASA JPL's is primarily used in spacecraft engineering and thus offers even more complex solution that an aviation industry requires. It also covers general reliability engineering, not only FMEA and FTA. Computer-Aided Fault Tree Ontology focuses on a fully automating FTA construction given a model-based environment with function and state transition tables. It demands more work to input the data but on the other hand automatically re-generates resulting tree which is explicitly derived from the tables. Needless to say, it only covers FTA and

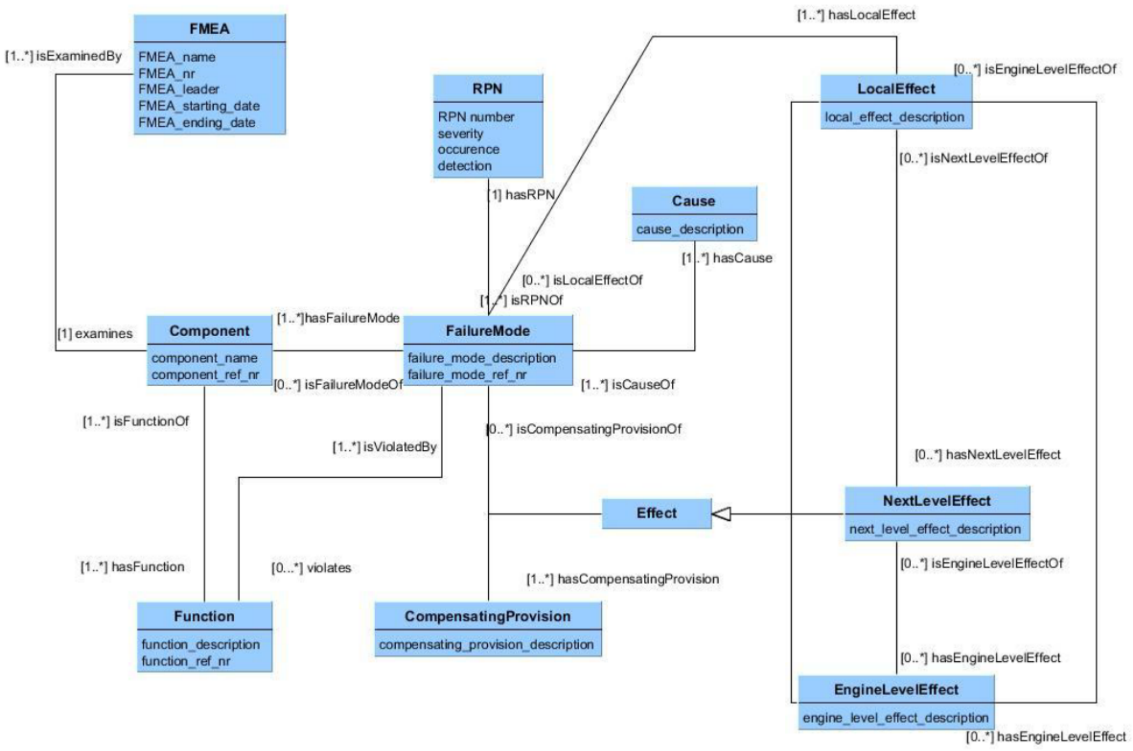


Figure 4.3: FMEA Ontology [1]

extension for other methods is complicated given how tied to tables is it. Last but not least, the ontology of Ms. Bolčeková aims on FMEA audience and significantly improves issues resulting of unstructured data. Similarly to Computer-Aided Fault Trees, it covers only FMEA, but yet keeps an open space for extensibility to other methodologies.

The conceptualization used in application is suggested and explained later in Chapter 7.

Chapter 5

Graphical Libraries

Moving towards the software part of this paper, the technologies available for reliability investigation methods, namely the graphical part of the FTA, need to be investigated. This chapter first declares capabilities required from the libraries, then presents chosen candidates and finally compares them with respect to criteria.

The application targets web browser for its portability and overall compatibility independent of operating system. The visualisation of tables necessary for FMEA can be easily composed by a standard HTML elements and thus needs to further investigation.

The trees on the other side are not native and need to be delegated to an external library.

5.1 Required Capabilities

The capabilities can be divided in three categories.

Firstly, capabilities needed to provide and visualise all the FTA domain functionality described in Section 2.2, such as render every type of event and gate shapes matching recommended visual guides and to be able to create a hierarchical structure including connector lines. To achieve that, the library has to be open for full customization of rendered shapes.

Secondly, the solution should support features from user experience spectre which simplify work with the editor. That primarily means tree visual transformation as dragging the rendered elements, zoom in and out and scroll to a selected point. To support the shapes organization, it should come up with its native automated shapes layout. Last but not least, the library should allow export of the drawn structure.

Lastly, from the developer perspective, it is important to look into the future and use library which is free to use, established on the market and properly documented.

The requirements are further justified by features prioritization in Chapter 7.

5.2 Candidates

This section enumerates the candidates as they were used for prototyping and testing during the evolution of the development process. Their advantages and disadvantages are pointed out and finally, the candidate selection is justified.

As the whole application is developed in JavaScript and ReactJS, the libraries for canvas manipulation targeting ReactJS were considered in the first place. The technological aspects of the application are further developed in Chapter 7 and Chapter 8.

5.2.1 React-Konva

React-Konva is a ReactJS port of Konva.js[1] which is a popular HTML5 2D canvas library that is, as the previous suggests, capable of drawing vector graphics inside HTML canvas element.

The library provides a lower level API thus the developer needs to build everything from scratch. What it means is that all components of the tree would have to be composed from rectangles, lines and custom shapes that would need to be defined on a coordinates basis. The overall modelling on a programmatic level would be constructed of initial acquiring of a canvas to draw onto. Next, place layers necessary to draw elements, after that compute the coordinates where to place the elements and compute overlaps, intersection and overall layout. Finally, the developer of the library needs to implement how to respond to user events such as manipulating the shapes, dragging, changing their positions, zooming-in and -out and generally intercept operations over the drawn image.

That being said, the user of the framework is given a complete control over the rendering process and the possibilities are unbound, so basically anything can be implemented, on the other side there is no explicit support for implementing diagram editors thus the complexity of the codebase grows rapidly as every single feature such as connecting shapes need to be implemented and further updated manually.

After initial prototyping while employing this framework, it was omitted and replaced by a following candidate to simplify the development and to delegate diagramming behavior to the library.

5.2.2 JointJS

JointJS[2] is a JavaScript framework focusing mainly on a construction of diagrams and implementing their editors. What necessarily needs to be mentioned in the first place is that the framework consist of two parts - JointJS Core and Rappid. JointJS Core is an open source framework that implements core functionality of the library and provides an API for canvas manipulation, rendering of shapes, connectors, their management, dragging, position updates etc. Rappid on the other side is a commercial paid extension building on top of JointJS Core adding further functionality, e.g. undo/redo, control panels etc.

[1]https://konvajs.org

[2]https://www.jointjs.com/opensource

Rappid is a nice extension that simplifies the work with the library, nevertheless the core framework is capable of everything that Rappid is if the functionality is implemented manually by the developer.

To similarly describe the development process as for the React-Konva, the JointJS Core builds on top of BackboneJS and jQuery libraries and implements a set of built-in components for composition of a diagram. These components are extensible and offer a full customization of design and behavior, yet come up with their defaults, so it can be reused if no further functionality is necessary. The solution was created as a standard JavaScript library with no ports to ReactJS, but the integration was fairly simple.

The main building block is *Paper* that holds a reference to canvas to draw onto and takes care of a rendering logic. A necessary complement to that is *Graph* being backed by Backbone models that represent the elements being put on the canvas. It holds a set of *Elements* that are, as was mentioned, extensible models of shapes that can be composed and connected together and provide basic diagramming behavior. For the purposes of the application, the *Paper* and all beneath is placed in ReactJS wrapper that reacts to user events. The application has its own set of overridden shapes enriched by the FTA visual components, namely types of the FTA event and corresponding gates below.

That being said, JointJS matched all the requirements established in the beginning of this chapter, even though some of them require more work. Another advantages of the framework lies in its well documented features, several demo applications showcasing different types of diagrams and its user community which helps when there are issues to overcome. The authors are active in an open source community and respond well to user feedback which promises a good potential support in case issues arise.

The JointJS library was chosen as an ideal candidate and its capabilities were further improved by an employment of *Dagre*[3] and *Graphlib*[4] libraries that were used as an extension to provide missing functionality of directed acyclic graph (DAG) layout necessary for tree alignment. The combination of JointJS + Dagre + Graphlib brings in an automated arrangement of the shapes which improves visual overview over the tree.

[3]https://github.com/dagrejs/dagre
[4]https://github.com/dagrejs/graphlib

Chapter 6

Similar Applications

The theoretical background of this thesis will be concluded by an overview of similar applications focusing on a combination of FTA and FMEA reliability methodologies. To remind, the goal of this thesis is to design a solution where both techniques could be conducted within one product, thus the overview will put emphasis on applications that are capable of combining both in the first place. Nevertheless, a single purpose solutions will be analysed as well to provide valuable insights into what features are provided, how they are solved and what could be improved. Per each product, its platform dependencies will be investigated and if publicly available, whether it is based on semantic web technologies or not.

These insights will serve as a requirement basis for the proposed solution and will then be summarized and enriched in Chapter 7 by requirements of aviation reliability domain experts involved as consultants of the thesis.

As there are many existing solutions for the case, only the ones that were the most relevant according to this thesis were analysed in depth and are thus mentioned here.

6.1 ReliaSoft XFMEA

XFMEA[1] is a desktop reliability application developed by ReliaSoft company that focuses on an implementation of reliability tools. XFMEA supports all types of FMEA described in Section 2.3 and multiple other risk assessment methods apart from RPN.

The data can be imported, exported and are stored in a relational database thus it is assumed they are not using semantic technologies for the data representation. Given the imported data, the software is capable of automatic RPN calculation, displaying graphs and diagrams about failures and provides advanced filtering. The failures can be covered by an automatically generated set of test cases to assure reliability. The resulting tables can be exported to Excel spreadsheet or to be stored in ReliaSoft SEP Web Portal together with generated reports.

Where the application stands out is in its connectivity to other reliability methods together with a dashboard where all the reports can be presented.

[1]https://www.reliasoft.com/products/xfmea-failure-mode-effects-analysis-fmea-software

6.2 SoftExpert FMEA

SoftExpert FMEA[2] is a web application that covers all FMEA types similarly to XFMEA.

The data can be imported, exported to Excel spreadsheets, but the internal structure is not mentioned. The software also automatically computes RPN, shows graphs, reports and also offers other possibilities apart from traditional table to visualize the data - hierarchical tree views and filtered lists.

On a user friendliness side, it filters the failure modes, prefills text fields for severity, occurrence and detection values and is capable of notifying users with preventive and corrective actions to be performed.

The advantage to notice here is the version for website which improves portability of the solution and alternate view over the data, but it lacks integration with other tools thus it needs to be combined with FTA software of other companies. The problem that obviously arises is the consistency of the data while being transferred between applications.

6.3 IQASystem FMEA

The software, or to be more precise - the plugin, from IQASystem[3] stands out from the other solutions by design - it works as an Excel plugin.

The plugin is installed to Excel and then offers several templates to begin with. The data import and export is obvious as the analyst stays in Excel all the time. A general workflow with the plugin works in 3 steps.

Firstly, either choose or configure custom template, then conduct the analysis and finally perform the risk evaluation.

Given that the plugin relies on Excel, it can easily calculate RPN similarly to previous applications. The place where it dominates is the speed how the analyst can start conducting the analysis. The creation of the template is fast and data editing as well. The plugin yet suffers from the portability offered by SoftExpert FMEA and from unavailability of implementing complex features due to dependency on Excel.

6.4 Relyence FMEA

The last FMEA application to mention here is the one from Relyence[4] because the company has a similar area of focus as ReliaSoft. Likewise ReliaSoft, Relyence puts emphasis on reliability methodologies products and provides binding between separate tools.

As previously mentioned solutions, Relyence FMEA imports and exports the data in Excel format. Unfortunately, there are no information about internal structure. Aside

[2]https://www.softexpert.com/produto/fmea

[3]https://www.iqasystem.com

[4]https://www.relyence.com

from a standard FMEA functionality, Relyence supports customization of the analysis workflow that can be enriched by approvals and notifications about changes.

Everything is displayed on a dashboard that collects all reliability reports. Speaking of sharing information, FMEA can be automatically transformed to FTA for which they also offer own solution that will be discussed later. While there are no documents how the transformation is done, it should be mentioned that NASA FTA Handbook [6] clearly states that constructing FTA from FMEA is a bad practice and produces bad results.

To avoid criticism, the solution is implemented as a web application and is fully optimized for mobile devices as well.

6.5 ReliaSoft BlockSim

Fault Tree Analysis[5] from ReliaSoft is a part of larger package called BlockSim which is a solution for system modelling and Reliability, Availability, Maintainability (RAM) analysis[6].

As being the counterpart to XFMEA, BlockSim also runs as a desktop application, but it does not allow data import. It is further shown that it is a common problem of FTA products.

It fullfills general requirements and rules mentioned in Section 2.2, but on top of that improves the reliability technique by adding a failure frequency and a several custom gate types:

1. NOT, NAND, NOR

   A set of gate to create an inversion of the original type.

2. Voting gate

   The gate is parametrized by a number of inputs and necessary quorum of inputs that need to be true for the gate to succeed. It has roots in combinatorics.

3. Sequence Enforcing Gate

   The gate succeeds only if the events happen in a specified order.

4. Load sharing and Standby Gates

   Complex gate types employing dependencies between input events and time-dependent system reliability.

That being said, BlockSim offers all required functionality and on top of that adds customized gate types. The main advantage however lies in a close integration with ReliaSoft infrastructure and in a fact that BlockSim itself allows analyst to develop RBDs right next to FTAs which simplifies the view over the overall system and improves data consistency.

[5]https://www.reliasoft.com/products/blocksim-system-reliability-availability-maintainability-ram-analysis-software

[6]a method for estimating system production availability

6.6 ITEM FTA

Another desktop representative of FTA software is a solution developed by Itemsoft[7]. After the trees are constructed, they can be evaluated by both of qualitative and quantitative analyses.

On a qualitative analysis side, the most important are the *cut sets* that represent a combination of leaf events that lead to a top event failure. The cut set is said *minimal* in case that, when any of events is removed, it is no longer a cut set. They are a helpful tool to show where the system is the most vulnerable, but are not necessarily required.

From the quantitative parameters, a standard top event failure probability is evaluated.

Similarly to BlockSim, it offers everything that is needed to perform FTA, but its UI is outdated and composed from the old Windows UI components and furthermore, it lacks connectivity to other reliability tools.

6.7 Edraw FTA

Edraw Fault Tree[8] is quite different from the previous FTA solutions. It likewise operates as a desktop application, but was designed to be more lightweight and puts focus on the graphical part of the analysis.

It lacks any support for the analysis itself, but provides all the event and gate shapes and emphasises the visual cleanliness. It might be useful for creation of nicely looking diagrams, but cannot be used for an aviation domain with high requirements for safety and reliability.

6.8 Relyence Fault Tree

The last application to analyse will be a counterpart to Relyence FMEA. The Relyence Fault Tree[9] is the only representative of web applications for FTA being mentioned here. It is fully integrated with the reliability tools from the company.

As mentioned earlier, the FTA can be constructed by a transformation from the FMEA and furthermore, it enables users to create an event library from where the events can be later retrieved if necessary. It supports both types of analyses (qualitative, quantitative) and can be integrated into Relyence dashboard.

Together with its portability, integration with other company products, risk assessment support and overall connectivity the software presents a lot of features to get inspired by.

[7]https://itemsoft.com/fault_tree.html
[8]https://www.edrawsoft.com/fault-tree-diagram-software.html
[9]https://www.relyence.com/products/fault-tree/

6.9 Summary

The analysis of the applications shows a selection of representatives of both reliability methodologies whereas it was shown that they are targeting desktop platforms or improve the portability by targeting web. They also differed in approach to data sharing between analyses. As was explained in Section 2.4, the technique should be combined together to achieve the best results thus the integrity of the data needs to be prioritized.

The problem was approached in a best way by both ReliaSoft and Relyence companies that are focusing on reliability in general and developed a tool set designed to work together.

What must not be omitted though is that all the presented solutions are available only as a paid commercial version and none of those was built on top of semantic web technologies.

The key functionality and innovative concepts is summarized in an upcoming Chapter 7 and will serve as a requirement basis for the thesis.

Chapter 7

Analysis

This chapter focuses on analysis of a solution that will be the resulting software prototype of this thesis. Firstly, it begins by summarizing similar applications and proposes application requirements consulted with domain experts involved in this paper. The requirements are prioritized according to the MoSCoW method explained later. An application ontology is then proposed according to required functionality and reliability methodologies specifications mentioned in Chapter 2 to be ready for future improvements. The analysis chapter is concluded by proposing and explaining the system infrastructure.

7.1 Requirements

The major goal of this thesis is to implement a web application focusing on FTA and FMEA reliability analysis and to be based on semantic web technologies. As could be seen in Chapter 6, existing applications can be divided into 3 groups where there are the ones that are FTA focused only, FMEA focused only or the last hybrid type where the solution does not offer both methods combined in one tool, but is powered by an infrastructure binding everything together.

None of the presented tools was, according to available documentation, based on semantic web technologies which can also be supported by the fact that most of them were developed primarily for desktop. Targeting the desktop indicates missing support for portability and forcing users to install the app physically on the computer which is something being abandoned lately in favor of web applications.

Except ReliaSoft (Section 6.1, Section 6.5) and Relyence (Section 6.4, Section 6.8) there was a lack of interchangeability between the methods and their model-base. ReliaSoft offered a connection to FTA (BlockSim), but only on a level of sharing similar data. Relyence on the other side proposes transformation tools to automatically construct FTA from FMEA, but the details on implementation were not provided and, as was mentioned, FTA Handbook [6] clearly discourages from doing so.

BlockSim has shown a nice example of model-based environment first presented in this thesis in Section 4.2 where the model was used as an information basis for generating a computer-aided FTA. BlockSim targets the consistency issue by offering also a RBD editor in the same application thus the overall system view is consistent with accordingly generated FTAs.

The functionality above will be used as inputs for functional requirements for the resulting application.

7.1.1 MoSCoW Prioritization

A requirements prioritization helps to determine the scope of the project and to keep both parties satisfied with resulting functionality. To perform the prioritization, MoSCoW method will be used. It proposes 4 groups of requirements whereas the ones being in the same group have the exact same priority [30].

- Must have (M)

  Requirements that are critical for the project to be considered successful. Without all being implemented, the overall result is incomplete and not usable.

- Should have (S)

  Features which should be implemented if there is time left. They add significant value.

- Could have (C)

  Features that are nice to have, but their added value is less significant than "Should have".

- Won't have (W)

  Also called a *wish list*. These features are not possible to implement due to various reasons (time, money, complexity, etc.), but can be considered in future. The method name *MoSCoW* was derived from the group naming.

7.2 Functional Requirements

Functional requirements define what functionality the system offers to user, but does not state how it should be achieved. The scope will be on a higher level and the functionality will be annotated as *<MoSCoW Priority> FR-<number>*, e.g. M FR-1, and the template used for the definition will be unified by following the template *"Application will allow users ..."*.

By defining and prioritizing functional requirements, it is possible to more accurately determine the scope of the project.

- **M FR-1** - Application will allow user registration, authentication and update password.

- **M FR-2** - Application will allow users to construct FTA in a visual editor.

That more in depth means that the user will be able to perform following operations aiming to create the tree:

1. Create, Edit, Delete Event
2. Assign Gate to Event
3. Link Events

- **M FR-3** - Application will allow users to use following event types - *INTERMEDIATE, BASIC, EXTERNAL, UNDEVELOPED, CONDITIONING*.
- **M FR-4** - Application will allow users to use following gate types - *AND, OR, XOR, PRIORITY_AND, INHIBIT*.
- **M FR-5** - Application will provide users a visual representation of the tree.
- **M FR-6** - Application will allow users to export the tree to an image.
- **M FR-7** - Application will provide users with quantitative analytical methods in form of probability assignment.

  Without providing either qualitative or quantitative analytical methods, the FTA is basically only on image. The probability of the top event and the corresponding propagation from the leaves helps analyst to identify weak spots in the system that can be corrected with a low effort yet having a big impact on the overall probability.
- **M FR-8** - Application will allow users to define the system.

  The system in this terminology represents a system of components and their functions, as will be explained later. The user should be allowed to define partonomies between components to represent the system's hierarchical structure.
- **M FR-9** - Application will allow users to assign a failure mode to an event.

  The failure mode defines how a component and its function can fail. It is a main building block in FMEA.
- **S FR-10** - Application will allow users to transform FTA to FMEA.

  The FMEA transformation expects the possibility to define *Failure Modes*, *Mitigation*, *Effects (Local, Next, Final)* and *RPN*.
- **S FR-11** - Application will allow users to export FMEA to Comma-separated Values (CSV) format.
- **S FR-12** - Application will allow users to reuse events in the tree.

  By doing so, the trees will indirectly support *TRANSFORM-IN* and *TRANSFORM-OUT* gates by composing the tree out of smaller ones by reusing their root as child.

- **C FR-13** - Application will allow users to import data of system partonomies, components and their functions.

  The systems are already defined on a level of UML diagrams and need to be imported in the system. It may be done either manually or automatically, but should require minimal effort.

- **C FR-14** - Application will allow users to define system structure by integrated editor.

  As mentioned in FR-13, the systems are already defined in UML diagrams or other modelling languages. While UML defines a serialization format, the transformation to graph databases would be complicated and still would not cover all possible formats. By integrating the editor directly in the application, the transformation step could be omitted.

- **C FR-15** - Application will allow users to construct an overall FMEA from all the trees at once.

  The requirement FR-10 expects a transformation in a one-to-one relation manner, but analysts usually need to aggregate multiple FMEA tables together to have an overview over the whole system.

- **W FR-16** - Application will allow users to collaborate on the analyses with other users, to share the diagrams and grant access.

  The collaborative work and restriction of access right to other users is a common requirement in applications where diagrams and analyses are created, but the emphasis is put on model based analysis and improved data interoperability rather than user collaboration thus it will be omitted.

- **W FR-17** - Application will allow users to update FTA by operational data.

  Data being collected during airplane operation are commonly evaluated with respect to corresponding FTA analyses. The application would provide possibilities to update the model by precomputed operational probabilities.

- **W FR-18** - Application will allow users to define system by functional and state transition tables.

  As explained in Section 4.2, the system can be defined as a model-based environment and FTAs can then be computed on top of that as so-called *"computer-aided fault tree"*.

- **W FR-19** - Application will provide users with qualitative analytical methods in form of minimal cut sets.

7.3 Non-Functional Requirements

Non-Functional requirements will be annotated similarly to functional ones as *<MoSCoW Priority> NFR-<number>*, e.g. M NFR-1. The outcome after prioritization will be qualities and constraints that the application needs to have and respect.

- **M NFR-1** - Application will be based on semantic web technologies.

  Semantic web technologies are the major topic in this thesis thus the application will be developed on top of that.

- **S NFR-2** - Application ontology should be based on UFO.

  The UFO is a well build "metamodel" for ontologies that gives us a guidance how to conceptualize the problem correctly and effectively while using well known terminology.

- **M NFR-3** - Application ontology will be designed to support FMEA.

  Yet the thesis puts emphasis on FTA, the model should be designed so that it allows usage of FMEA with the same data.

- **S NFR-4** - Application should use JOPA.

  While graph databases can be approached without OOM, JOPA introduced in Subsection 3.3.2 improves readability and simplifies additional layer of abstraction which makes it an ideal candidate.

- **M NFR-5** - Application will target browsers.

  As Chapter 6 mentions, targeting the browsers improves portability of the data and their accessibility.

- **S NFR-6** - Application should be written in ReactJS.

- **S NFR-7** - Application diagrams should be rendered by JointJS graphical library.

  JointJS library fulfills all the requirements on graphical libraries defined in Chapter 5 thus should be used as the main visualization technology.

7.4 Proposed Ontology

In Chapter 4, existing ontologies emphasising reliability analyses domain were introduced. It began with NASA JPL's Fault Management Ontology, continued with Computer-Aided FTA Ontology and the topic was concluded by FMEA Ontology by Ms. Bolčeková.

To summarize the insights from the chapter, the NASA ontology provides terminology for the whole fault management domain, but only an excerpt was analysed. The model primarily focuses on spacecraft missions rather than aviation domain which adds on complexity in concepts like *Mission*, *MissionImpact* etc.

NASA JPL's ontology served as an inspiration and basic model for the one from Ms. Bolčeková. She has extracted FMEA features and simplified the relations for aviation domain. Her primary concepts are *Component, Function, FailureMode, RPN and associated Effects*. Despite the model puts emphasis on FMEA for airplane engine, the ontology itself is not impacted by this fact too much and there are only minor changes necessary. The changes have been highlighted in the thesis of Ms. Adamcová [10].

The last one to summarize is the Computer-Aided FTA Ontology whose focus, on the other side, lies in automatic FTA generation. The approach to achieve that is very different and originates in model-based environment by definition of *Function* and *State Transition Tables*. It specializes only on FTA and the structure is not simply extensible by FMEA features.

7.4.1 Concepts

As mentioned in Section 7.2, the main emphasis in the tool being just analysed will be put on an FTA construction with a possibility to convert the data to FMEA table from the same model.

The proposed ontology was built with respect to UFO from Subsection 3.1.4 and its main concepts of *endurant*, *perdurant*, *qualities* and *dispositions*. The inspiration was taken mainly from the ontology of Ms. Bolčeková and her FMEA which was enriched by FTA concepts to fulfill the requirements. Needless to say, it was developed as an *application ontology* thus it aims to provide semantically correct data targeting primarily this application, yet the structure still should be reusable elsewhere if required.

The model being ready for both FTA and FMEA is depicted on Figure 7.1 and its concepts are described in the following list:

- *System*

  A system is an aggregating concept that holds the representation of the mechanical system together by a set of *Components* being composed to form a dependency structure. It opens application possibilities to construct the overview directly without using UML. It is an *object* in UFO terminology.

- *Component*

  Component belongs to a system and can be linked to other components in an aggregation manner. It can have *Functions* and associated *FailureModes*. Both relations *has function* and *has failure mode* are sub relations of the inverse relation of *inheresIn*, *bears*. The component being a mechanical part of the system (thus *moment*) can fail completely or only in a manner of failure of some of its functions.

- *Function*

  Function defines one capability of the component and what behavior the component can offer. The existence of function is conditioned by the existence of component and thus cannot exists without it which makes it *a moment* of component *object*.

- *FailureMode*

  The existence of failure mode is similarly conditioned by the existence of component. It further influences the function, but note that *influences* is an application optimization of the model and does not originate in UFO. The failure mode represents a way how the component can fail and which function will be influenced by the failure. It is manifested by *FaultEvent* where the *has effect* relation is a sub relation of UFO *isManifestedBy*.

- *Mitigation*

  Mitigation can be assigned to failure mode proposing a way how to minimize impact of its effects. The *is mitigated by* is a sub relation of *bears*.

- *FaultEvent*

  As was mentioned, the fault event is a manifestation of the failure mode and defines in depth how the component can fail. The fault event is a first of FTA oriented concepts of the model. It is a building block to construct the tree structure by being a root of *FaultTree* and yet being associated with other fault events linked by *is caused by* relation. The fault event specifies type of an event rather than the real failure thus it is an UFO *event*.

  The event is assigned probability as depicted in Figure 7.1.

- *FaultTree*

  Fault tree, as was mentioned in fault event, represents the FTA itself by referencing the root fault event and the rest of the tree is composed of events which cause directly or indirectly the root fault event. The tree provides an access to perform the FTA reliability analysis over the fault tree's events with assigned occurrence probability. It is similarly an *event*.

- *Failure Modes Table*

  The failure modes table is a beginning of the FMEA extension to the model. The table is derived from the tree and consists of *FailureModesRows*.

  An FMEA table is a collection of rows each of which represents a causal chain where a low level component event leads to a top level failure. In UFO the table can be represented as a collection of events, and the row is a complex event composed of a causal chain of events.

- *Failure Modes Row*

  A representation of FMEA table row as shown in Figure 2.4. The row has, by the nature of the model, access to all columns necessary to visualise an FMEA row, namely - *Component, Function, Failure Mode, (Local, Next Higher-level, Final) Effects, RPN, Mitigation*. Each row is assigned an *RPN* for risk assessment and prioritization during the transformation process.

The transformation algorithm will be properly explained in Chapter 8.

- *RPN*

  RPN simply represents a risk priority number assigning severity, occurrence and detection (as shown in Figure 7.1) to each table row.

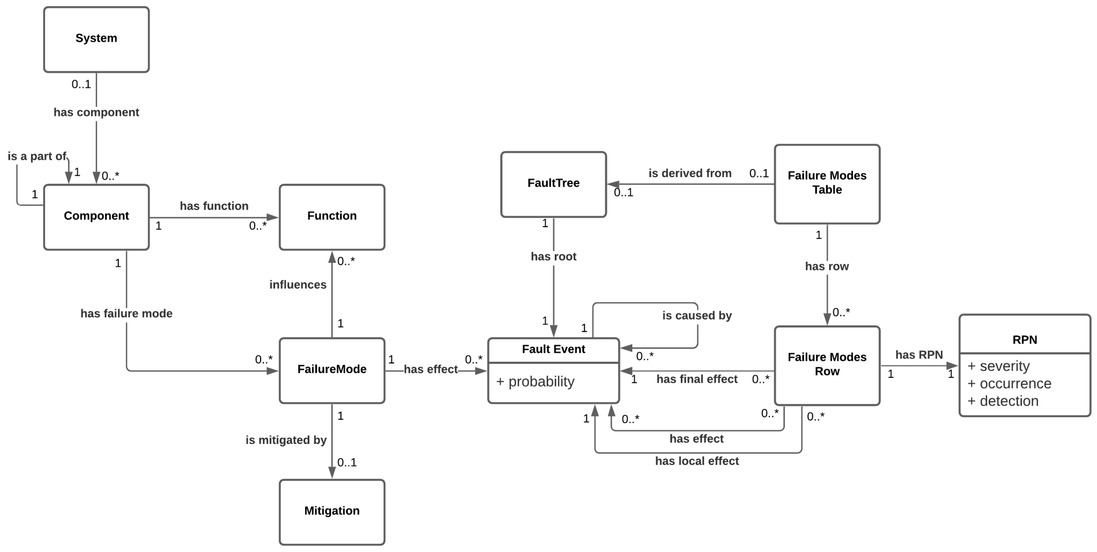


Figure 7.1: Proposed FTA, FMEA Ontology

As is explained later, the ontological model that was just presented offers a flexibility of firstly constructing the system, after that build a fault tree and finish the initial analysis by converting the trees to FMEA tables. By being able to perform all three steps in the same application, it is expected that the data consistency will be significantly improved as it minimizes migration errors.

Furthermore, the proposed model can be used for both FTA and FMEA analyses in either a context of investigation of an incident or in a risk assessment context during the system design phase. In the case of incident analysis, the model will consist of instances of the proposed ontology, e.g. specific systems and components which instantiate the system and component classes. In the context of risk assessment during the design phase the model will consist of classes, e.g. describing components and system types. The thesis focuses on the latter use case.

While the presented ontology is conceptually correct, the JOPA framework formally works with instances of an ontology rather than concepts themselves. The ontology for JOPA thus needs to be derived from the meta-model in Figure 7.1 to create sub-classes and sub-properties. The JOPA ontological model will then have an identical conceptual structure representing a meta-model of the original ontology and it will be mapped to the meta-model conceptualization. The generated instance data by JOPA then correspond to the description of the event and object types.

7.5 Architecture

As could be seen in an analysis of existing solutions and also in the application requirements, the resulting solution should provide portability and be accessible without requiring an installation. The data representation should be tied to semantic web technologies, powered by an ontological conceptual model and the model should profit from storage optimizations of graph databases.

To put it in other words, the web application should allow users to create and manage both FTA and FMEA analyses each in a dedicated analysis editor. The editor then communicates with the server side of the solution that fulfills purpose of validation of semantical correctness of published data and takes care of connection to a persistent storage.

The architecture of the application is depicted in Figure 7.2. The figure depicts the architecture all the way to a component level, briefly mentioning technologies that are relied on and describes the main communication protocol between frontend and backend part.

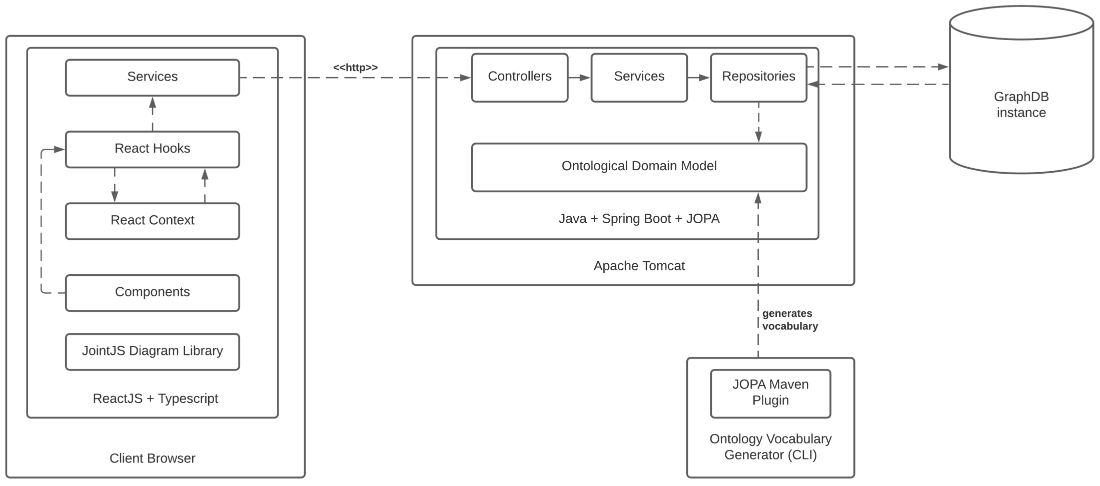


Figure 7.2: Application Architecture

The frontend part of the application should, as the requirements state, be written in ReactJS which will also make use of JointJS library for diagram components manipulation and will rely on React Hooks to manage application state. Type safety will be ensured by employment of TypeScript. FMEA table is server-side computed and rendered by a powerful Data Grid API[1] from Material UI library.

The backend part exposes an HTTP interface to access database resources. To do so, it can simply compose out of 3-layers - controller, service and repository. The repository

[1]https://material-ui.com/components/data-grid

layer will employ JOPA OOM to hide the complexity of SPARQL queries and will thus help with persistence context management, caching and model definition. JOPA model vocabulary will be automatically generated from Turtle file defining the ontology in a subproject relying on KBSS-developed plugin for Maven.

While it is fancy to use microservice architecture nowadays, it is understandable from the problem definition that the backend only provides an access to graph database with a simple level of abstraction on top. A classic *client-server* architecture is thus sufficient.

As a triplestore was chosen GraphDB[2] which is provided either in enterprise, standard or free version. The free version has no significant deficiencies for the purpose of this thesis, is ready to be deployed anywhere using Java and to conclude, it has already a running instance on a hosting server provided by KBSS group closely cooperating on the project.

[2]https://www.ontotext.com/products/graphdb

Chapter 8

Implementation

The Chapter 7 has determined the project focus, what functionality is necessary and which is a "nice to have" or infeasible at the moment. It has further explained the system architecture and briefly mentioned the technologies this thesis should stand on.

This chapter begins by explanation of application ontology meta-model used for JOPA and its relation to UFO. Next, it continues by describing non-functional requirements that significantly influenced the implementation of functional requirements later. The chapter states how they were fulfilled for both the requirements.

The fulfillment of the (non-)functional requirements will be denoted similarly as when they were defined thus by referencing it in a *FR-<number>* or *NFR-<number>* format.

8.1 Transforming JOPA Meta-Model to UFO

To continue in a discussion from Section 7.4, UFO cannot be mapped directly to the meta-model used in the application. The instances of meta-model classes are types while the instances corresponding to UFO classes are individuals. The application works with meta-model and it creates instances of the meta-model using JOPA. The transformation to UFO is depicted in Figure 8.1. The upper part of the diagram describes the schema and lower shows an instance. The left side of the diagram shows a selected relation from the meta-model and the right side the corresponding UFO model.

To construct an ontology compliant with the UFO model, instances of the classes in the meta-model need to be transformed to sub-classes in the UFO model. Similar is for relations, e.g.:

- *component* → *«ufo:Object» component*
- *function* → *«ufo:Disposition» function*
- *has function* → *«ufo:bears» has function*

The stereotypes are used to indicate that a class is sub-class of given UFO classes. Same holds for properties. This pattern holds for all the classes and properties being

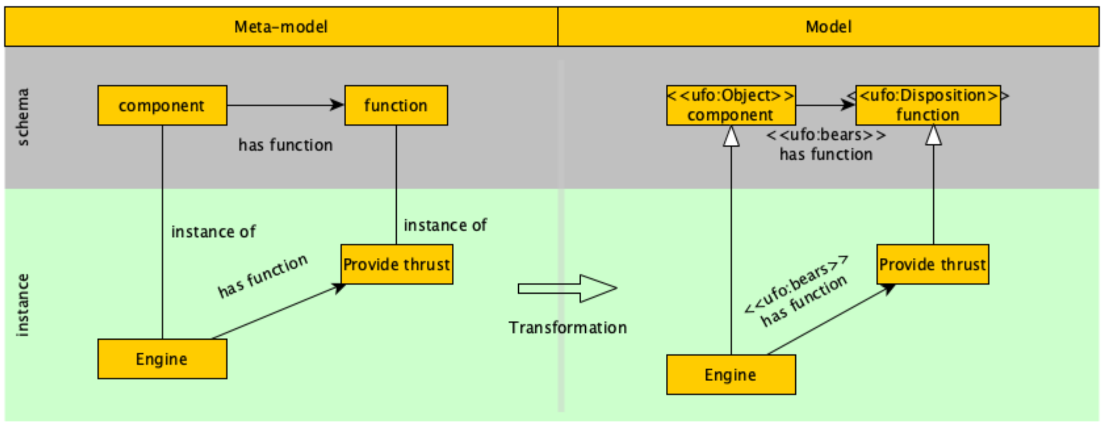


Figure 8.1: JOPA Meta-model to UFO Model Conversion

mapped to UFO.

8.2 Non-Functional Requirements

Non-Functional requirements are generally defining how the system should perform and which metrics it should achieve. As was mentioned earlier, it does not put pressure on provided functionality, rather on achieved performance, response time, availability, etc.

The subsections will focus on non-functional requirements being related together.

8.2.1 Semantic Technologies

To briefly remind the points mentioned already when discussing application architecture, the backend part is written in Java, powered by Spring Boot and all is built together by Gradle.

The application model is defined on two levels *(NFR-1)*. Firstly, there is a subproject that relies on Maven, has an access to a Turtle ontology definition and, when compiled with *maven-jopa-plugin*, it generates *Vocabulary* class for Java, is linked to classpath and contains the terminology to use. Secondly, an application ontological model is written in Java with a help of JOPA's *OWLClass*, *OWLDataProperty* and *OWLObjectProperty* annotations *(NFR-4)* that requires an *iri* specification whereas the generated *Vocabulary* comes in place. An excerpt from ontological model is shown in Listing 8.1.

Listing 8.1: JOPA Model Definition

```
@OWLClass(iri = Vocabulary.s_c_FaultEvent)
@Getter
@Setter
public class FaultEvent extends AbstractEntity {

    @ParticipationConstraints(nonEmpty = true)
    @OWLDataProperty(iri = Vocabulary.s_p_hasFaultEventType)
    private EventType eventType;

    ...
}
```

Controllers of the application are tailored by Spring Boot and communicate with the frontend solution in JSON-LD format (details in Subsection 3.2.4) to make use of the semantic data.

While the service layer provides a standard business logic manipulating the data and implements tree traversal algorithms, the data access layer (repositories) manage the connection to underlying GraphDB database through JOPA *EntityManager*, *PersistenceContext* and by executing native queries. The connection is configured externally in application properties together with an ontological driver to be used which is an important advantage of JOPA solution that it can, without any model modifications, replace an underlying database by only switching the driver. The application relies on *SesameDataSource* driver that comes in with JOPA distribution package.

The inspiration by UFO *(NFR-2)* was already introduced in Section 7.4 and the readiness for FMEA *(NFR-3)* is explained more in depth in explanation of functional requirements implementation.

8.2.2 Application Frontend

The remaining non-functional requirements on the system were primarily focusing on the target platform and a specification of technologies.

After discussions with domain experts, the conclusion was made that the solution should be developed as a web application *(NFR-5)* due to a growing demand for semantic web technologies and accessibility without installing desktop software.

It was further decided that the codebase will be written in ReactJS and types ensured by Typescript *(NFR-6)* as its commonly used framework and furthermore, it is used for projects in KBSS group which will simplify possible adjustments by other members.

To conclude fulfillment of non-functional requirements, the frontend solution uses JointJS library *(NFR-7)* for diagram manipulation. As was discussed in Subsection 5.2.2, the library is written for pure Javascript projects and does account with ReactJS framework on top.

The communication of sequence calls between JointJS and ReactJS to render a shape on a diagram canvas is depicted on Figure 8.2. When ReactJS decides that *Editor Canvas* needs to be rendered, the canvas relies on ReactJS hook *useEffect* to invoke a code after it has initially been displayed. When that happens, an empty JointJS *Graph* is created to represent the diagram data. Then the *Paper* parameters are specified and it is given the *Graph* whose data to render. After it is linked together, diagram and shape behavior is specified on *Paper* so that it responds to user clicks, opening of context menus and changing shape position.

Once that is completed, canvas shapes are about to be rendered by passing the *Graph* reference to them. Each shape is similarly to *Editor Canvas* wrapped in ReactJS Component and manage the diagram shape in its *useEffect* hook where they firstly create the shape instance and then add the instance to the *Graph*. By doing so, the *Paper* is able to detect updates in the *Graph* and draws the shapes on the canvas.

By this approach, a library originally developed for Javascript only was migrated to ReactJS framework while keeping the capabilities of both solutions.

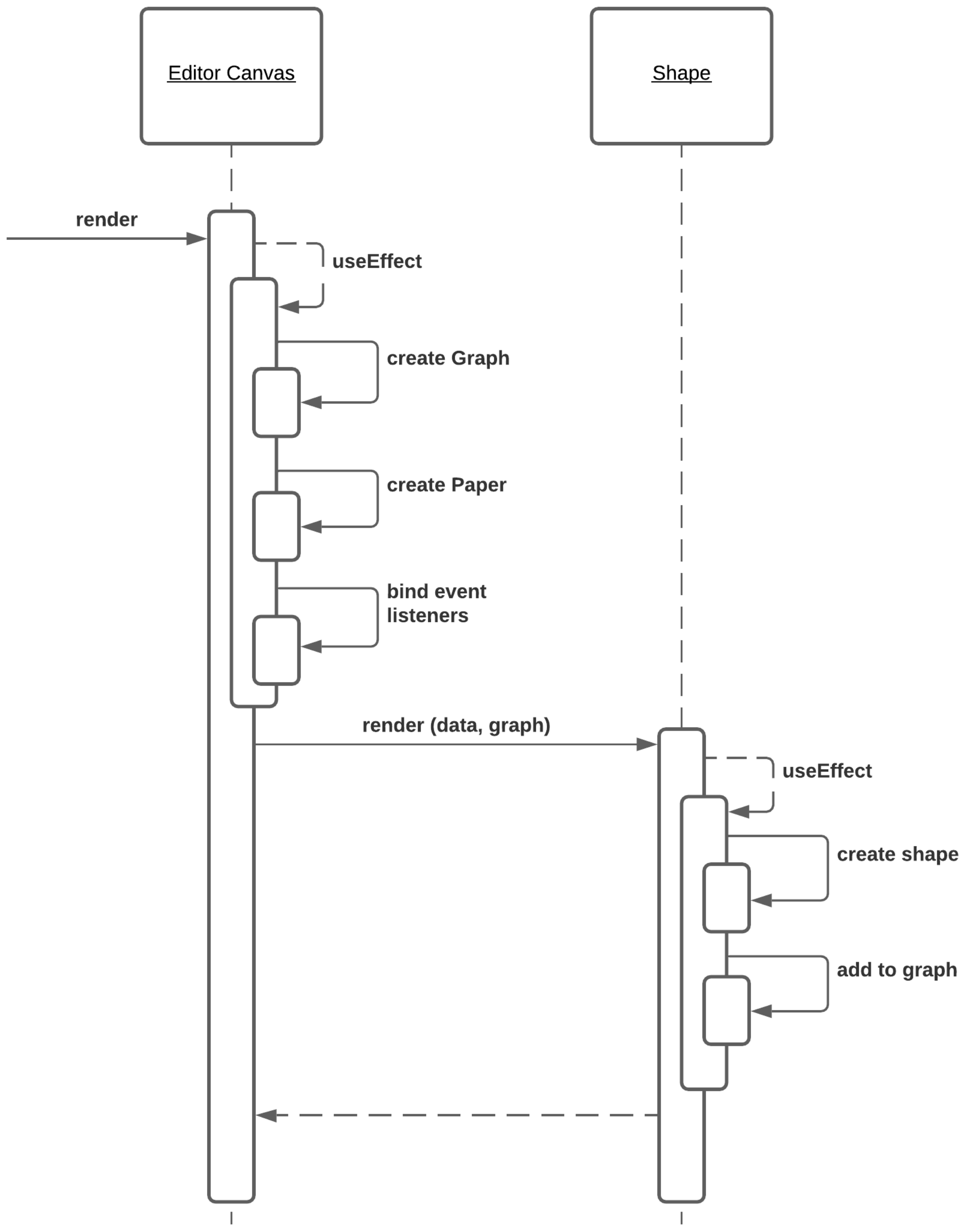


Figure 8.2: JointJS Management in ReactJS

8.3 Functional Requirements

The implementation of functional requirements was more complex than of the non-functional ones and thus deserves more explanation to be provided. Due to those reasons, the section follows a similar structure as when they were defined, enumerate the functionality in a list and explain in detail how each of them was achieved and why such approach was chosen.

- **M FR-1** - Application will allow user registration, authentication and update password.

  The application is deployed with a public access on KBSS server for the domain experts to test. As the solution is primarily focusing on implementing a prototype of FTA construction and FMEA conversion on top of ontologies, an advanced user access granularity *(FR-15)* such as document ownership, sharing with others or live collaboration was dropped for now.

  From a server side perspective, registered user information are held in GraphDB database as well and user session is managed by a usage of JSON Web Tokens (JWT) which is sent in every request. User can further update its password if necessary.

- **M FR-2** - Application will allow users to construct FTA in a visual editor.

  The FTA can be created on a visual editor drawing the shapes on a canvas. Each tree is created with its name and with a definition of top event. The underlying structure of the tree is then dependent on the tree root. The events are created via a context menu of the event under which they should be connected. If the event has other type than *INTERMEDIATE*, the child creation is prohibited. This reflects a top-down approach of the construction process.

  The events can further be edited including its *name*, *description*, *type* or *probability*. If the event is of type *INTERMEDIATE*, user can also specify a gate type to place below - the default one was chosen *OR*.

  Figure 8.3 shows an overall view of the tree editor and is further referenced to describe selected functionality. From what has been discussed until now, it can be seen that it consists of two major parts - the graphical editor itself and a sidebar on the right allowing configuration of selected event.

- **M FR-3** - Application will allow users to use following event types - *INTERMEDIATE, BASIC, EXTERNAL, UNDEVELOPED, CONDITIONING.*

  While analysing the tree, the user can take advantage of all above mentioned event types. The type of an event in FTA is mainly to simplify a visual understanding of the tree and to deduct the meaning without a necessity to read the event details. From a functional point of view, they do not change behavior of the system when performing tree analyses as much as gates do.

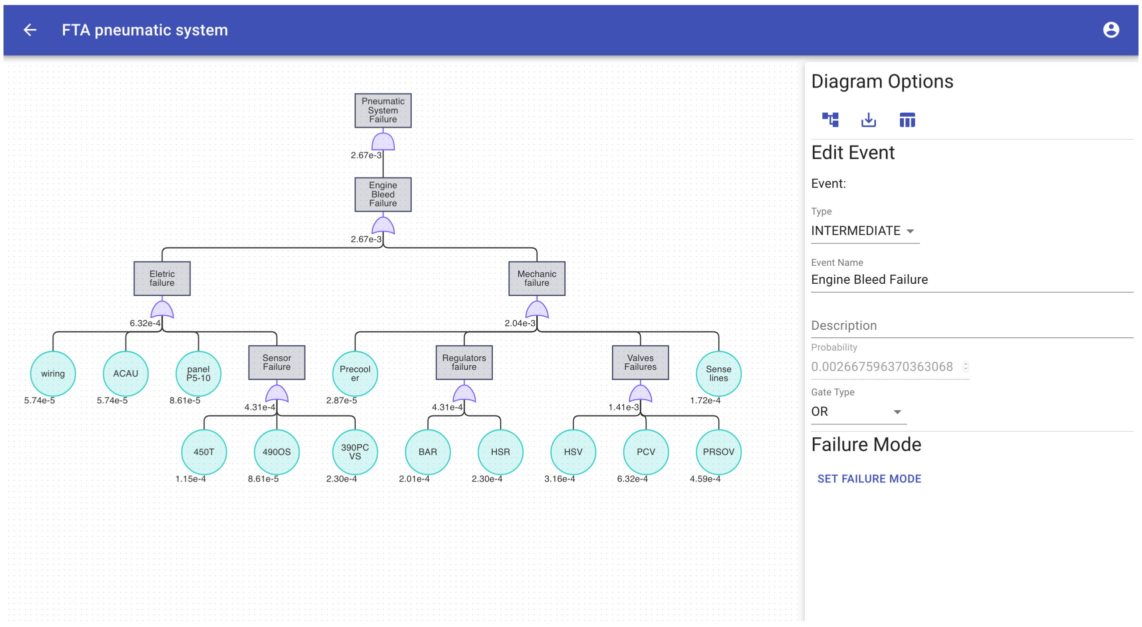

FTA pneumatic system

Diagram Options

Edit Event

Event:

Type: INTERMEDIATE

Event Name: Engine Bleed Failure

Description

Probability: 0.002667596370363068

Gate Type: OR

Failure Mode

SET FAILURE MODE

Figure 8.3: FTA editor

- **M FR-4** - Application will allow users to use following gate types - *AND*, *OR*, *XOR*, *PRIORITY_AND*, *INHIBIT*.

  Speaking of gate types, they add a significant value to analysis by amending the probability propagation within the tree. The editor supports all of the gate types as stated in the requirement *FR-4* - *AND*, *OR*, *XOR*, *PRIORITY_AND*, *INHIBIT*.

  There are several approaches how the FTA can handle the gates. Firstly, FTA uses the gates only for a visual representation and does not provide quantitative analysis tools at all. Secondly, events are treated in a binary fashion such that they either do not happen or they do which could be translated to probabilities with fixed values 0 and 1. Lastly, the events have an arbitrary probability within 0 to 1 range.

  The choice of event probability approach influences how the gates treat the propagation. It was decided that application will work with the last option which introduces better granularity and allows analyst to see how a change of a leaf event probability influences the top event.

  The following list analyses behavior of all the gate types[1] in the system.

  – AND Gate probability calculation

$$\prod_{i=1}^{N} P_i \tag{8.1}$$

  – OR Gate probability calculation

[1] Reference definition taken from http://www.crgraph.com/Fault_Tree_Analysis.pdf

$$1 - \prod_{i=1}^{N}(1 - P_i) \tag{8.2}$$

- XOR Gate probability calculation

  XOR gate is typically defined for two inputs only where it outputs 1 if only one input has value different from 0. The multiple-input XOR has no exact specification and can be understood in two ways. Either it outputs 1 if it contains odd inputs with value different from 0, or it outputs 1 in case that one and only input from all is different from 0. The second option reminds classic two-input XOR the most and thus the application implements multi-XOR by applying the XOR gate on two inputs in iterations. By doing so, the inputs are reduced to a single number.

- PRIORITY_AND Gate probability calculation

  Despite *PRIORITY_AND* gate is mentioned in the NASA FTA handbook [6] as in other sources, there is no strict definition how the gate should behave. Some applications implement the gate in a way that *gate fails in case that input events all fail in a specified sequence*. This definition denotes that there exists an order between input events of the gate and also a time dependency between them to distinguish which failed first.

  While there have been studies how to approximate *PRIORITY_AND* probability [31], the application uses own approach to specify a sequence of input events in the editor and assign a *sequence probability* as denoted in Formula 8.3 with symbol P_{seq}. Such solution was emphasised by the fact that events in the application have no information about time and thus the decision how probable the sequence is was transferred to analyst.

  When user chooses a sequence in which events should fail and *sequence probability*, then the probability is calculated as a simple multiplication of *sequence probability* by an output of *AND* gate. If no sequence probability is specified, the gate outputs probability of value 0.

$$P_{seq} \prod_{i=1}^{N} P_i \tag{8.3}$$

- INHIBIT Gate probability calculation

  INHIBIT gate is similar to *AND* gate with the difference that it is being used together with *CONDITIONING* event which is visually plugged-in from a side of the gate to highlight that there is another event influencing the gate which needs to be counted on, but cannot be changed.

• **M FR-5** - Application will provide users a visual representation of the tree.

A visual representation of the tree has been shown in Figure 8.3. The visualisation is rendered by JointJS as explained earlier and automatically updated by ReactJS.

- **M FR-6** - Application will allow users to export the tree to an image.

  The tree can currently be exported to Portable Network Graphic (PNG) format. The export is triggered by user clicking export button in editor sidebar menu. When requested, application serializes the data that are sent to server which converts the data to PNG by Apache Batik library.

- **M FR-7** - Application will provide users with quantitative analytical methods in form of probability assignment.

  As mentioned above, each event can be assigned a probability. It can be assigned to all event types except *INTERMEDIATE* one whereas it is derived from its children. The propagation happens in a bottom-up manner from the leaf events through the gates all the way to the root event. Application structure has been designed thinking ahead of possibilities that a new gate type might be introduced and thus it manages the gate calculations by a *strategy* pattern[2].

- **M FR-8** - Application will allow users to define the system.

  The system definition is a base model for both FTA and FMEA analyses. It gives the analyst an overview how the system looks like, what are the partonomies between components and what function do the components fulfill. The application has the ontology designed for such purpose and supports storing of component partonomies (currently only 'is part of') and definition of component functions.

  Further functionality regarding the system definition is described in requirement *FR-14*.

- **M FR-9** - Application will allow users to assign a failure mode to an event.

  To remind, the failure mode is a main building block of a row in FMEA table. By assigning a failure mode to an event, the application can derive a component and functions that are influenced.

- **S FR-10** - Application will allow users to transform FTA to FMEA.

  Having understood the functionality and algorithms before, it is possible now to proceed to an explanation how the FMEA is converted from FTA. The application has a definition of system, its components, component functions and linked failure modes. The failure mode is further connected to an event. The FMEA row then, by our approach, corresponds to a single path from a leaf to root.

  When the analyst decides to create FMEA from FTA, the application calculates paths from the tree root to all its leaves. The algorithm relies on a Breadth First Search (BFS) algorithm that was modified to remember the paths. All the paths are then shown to the analyst to choose which intermediate events to keep. Leaf and root are always fixed, but *next higher-level effects* can be skipped if a desired level of detail in table is lower.

[2]https://refactoring.guru/design-patterns/strategy

The leaf event is used as a *local effect* and *failure mode*, *component* and *functions* are also taken from the failure mode assigned to the leaf event. When a component has multiple functions that are influenced by the failure mode, the FMEA row is duplicated in the table per function. The middle events selected by the analyst are used as *next higher-level effects* and the root is used as *final effect*.

Each path is then assigned an RPN that can be edited later if necessary. If the failure mode of the leaf event is assigned mitigation, it is displayed right next to RPN in the row as well.

The server then creates a new FMEA table for the FTA diagram and stores all its paths as table rows. The table data returned back from server are visualised in DataGrid component from Material UI which has an advantages that both columns and rows are configurable as JSON thus only server side changes are necessary if content changes (add, rename, remove columns) are requested.

An example comparison of FTA and corresponding FMEA table is described in Chapter 9.

- **S FR-11** - Application will allow users to export FMEA to Comma-separated Values (CSV) format.

  The table is exported by requesting the FMEA table from the server in CSV format. The structure is similar as when visualised in the frontend application.

- **S FR-12** - Application will allow users to reuse events in the tree.

  A reusing of the events has been an interesting topic with the domain experts as well as there are no correct or wrong answers how to do so. Aviation systems consist of several components which might share same subcomponents such as valves, wiring etc. Failure of such subcomponents will negatively influence all of the components above thus they should be same in all the subtrees.

  There are several options how to implement the reusing:

  1. Prohibit reusing
  2. Use event as template
  3. Reuse event without its subtree
  4. Reuse event with its subtree

  Providing a background earlier on the topic in Subsection 2.3.1 of subcomponents and their similarity across the whole system, it was decided to use the last option of event reusing with its subtree. When a user decides to place a new event under a currently selected one, he or she is given an option to reuse an existing event.

  A current implementation in the application prohibits placing an existing event in the tree where it is already used to make sure user does not create a cycle from the subtrees. This implementation has been shown in Chapter 9 as incorrect because aviation systems commonly use symmetric subcomponents where e.g. an airplane has engine #1 and engine #2 which are identical.

A solution here would be to validate only whether the event has not already been used within a path where it is about to be inserted. Such validation would prevent circular dependencies where an event is one of its own children and so on. Unfortunately, when such functionality was implemented, the application stumbled on limitations of JOPA library which are further developed in Section 9.3.

To conclude event reusing, this functionality partially substitutes *TRANSFER_IN* and *TRANSFER_OUT* gates from the FTA specification.

- **C FR-13** - Application will allow users to import data of system partonomies, components and their functions.

  Initially, it was expected that the application could import the data about the system from files where they are stored. Nevertheless, as the data were inconsistent, often missing and hard to understand, it was proposed implementing an integrated system editor directly in the application which will be further explained in next requirement commentary.

- **C FR-14** - Application will allow users to define system structure by integrated editor.

  In Section 2.4, the duplicates in the system definition were identified as one of the major issues in the analysis process. Commonly, the analyst receives or creates the system definition in UML. As could be seen in Chapter 6, the applications usually do not combine system definition and reliability analyses which leads to a necessity to transport the data from one application to another. By doing so, migrating data about the system definition introduces duplicates, inconsistent or missing data in a new application.

  The solution thus contains an integrated system editor which allows user to create components, link them together in a *is part of* manner and assign their functions.

  Chapter 9 further describes system diagram on an example of pneumatic system partonomies.

- **C FR-15** - Application will allow users to construct an aggregating FMEA from all the trees at once.

  The last functionality according to requirements implemented for the purpose of application prototype is aggregation of FMEA tables. FTA are oriented on the top event representing a failure and do not reflect information about the system partonomies. FMEA is on the other side primarily interested in components and functions.

  To avoid losses of the system partonomy information by being focused on one tree only, the application implements a functionality where a large (aggregating) FMEA can be created from all the trees. The conversion algorithm works similarly with the change that it has multiple root events and if a subtree is reused in another larger tree, its root will not be used as a *final effect* but will appear as one of *next higher-level effects* instead.

The requirements *FR-16* to *FR-19* were not implemented within this thesis but might be considered in future.

Chapter 9

Testing

Chapter 7 has declared what the application scope is and Chapter 8 has provided details how application requirements were implemented together with background details and encountered issues.

In this chapter, the functionality is reviewed by both automated testing and a verification with domain experts. The chapter is closed by mentioning major issues that were discovered during the development and verification process and present possible solutions.

9.1 Automated Testing

Server side application developed in Java and Spring is tested by automated integration tests.

As the controller layer only delegates the calls from the API to service layer with an employment of deserialization library jb4jsonld[1], there is no critical functionality to test.

The testing thus emphasises the service layer manipulating the data and providing additional value. Repository (data access layer) again delegates calls to *EntityManager* from JOPA thus the tests are covering only more complicated cases.

The client side of the application heavily relies on canvas manipulation and for that reason, it was tested by a combination of manual testing and by a verification on real data.

9.2 Domain Experts Verification

The application has been developed in a close collaboration with domain experts from Faculty of Transportation Sciences on Czech Technical University in Prague. The experts were specifying the requirements, providing a domain knowledge and advice during the development. It was thus a straightforward step to verify the overall results with them on real aviation data.

[1]https://github.com/kbss-cvut/jb4jsonld

There were 5 domain experts in total where 2 of them are the supervisors having experience in aviation safety and reliability from aviation organizations and 3 students being guided by them. The application was reviewed in cooperation with the students.

The proposed verification process was performed in three steps where each step was created by one of the experts. The analysis was thus combined by three different experts cooperating on one system together which reproduces a real live scenario. The overall goal was to perform an analysis of a pneumatic system of Boeing 737 Next Generation airplane. The input data (system diagram, FTA) come from a bachelor thesis of K. Mündel [32] who was one of the participating experts. The FTA and system diagram from the thesis are depicted in Figure 9.1 and Figure 9.2.

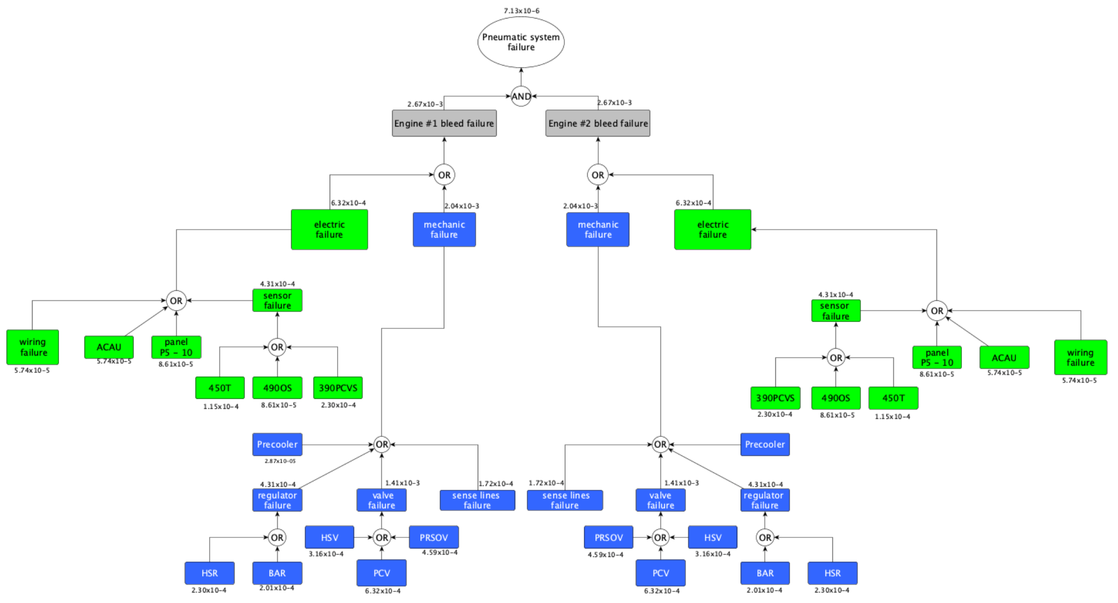


Figure 9.1: FTA of Pneumatic System [32]

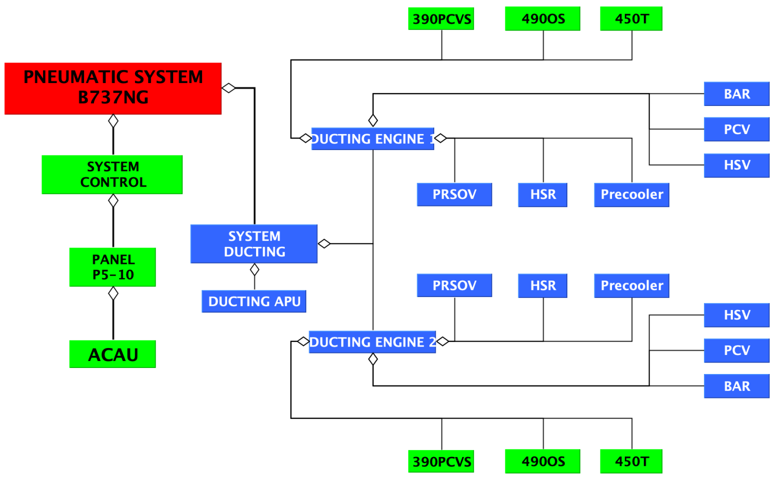


Figure 9.2: System diagram of Pneumatic System [32]

As the FTA shows, the *Pneumatic system failure* event is caused by two events *Engine* *bleed failure*. The bleed failures are caused similarly by *electric failure* and *mechanic failure* and represent symmetric branches in the tree.

9.2.1 Electric Failure

The verification began by modeling the electric failure FTA and an appropriate conversion to FMEA. Firstly, the domain expert constructed FTA of electric failure depicted in Figure 9.3. The comparison with the original FTA shows that resulting probabilities were calculated correctly and the FTA layout and distribution of events is similar.

An initial part of system partonomies (overall result is visible in Figure 9.7) was then defined where the expert has added only *System Ducting* component which was assigned *450T function*.

Given the inserted data, FTA event *450T* was linked with a failure mode influencing the *System Ducting* component and FTA was converted to FMEA table. The FMEA table in Figure 9.4 depicts a result of conversion showing all the paths from the leaves to root event and RPNs estimated by the analyst. It can be noticed that none of *next higher-level effects* was excluded from FMEA and that the table contains all the leaves as *local effects*.

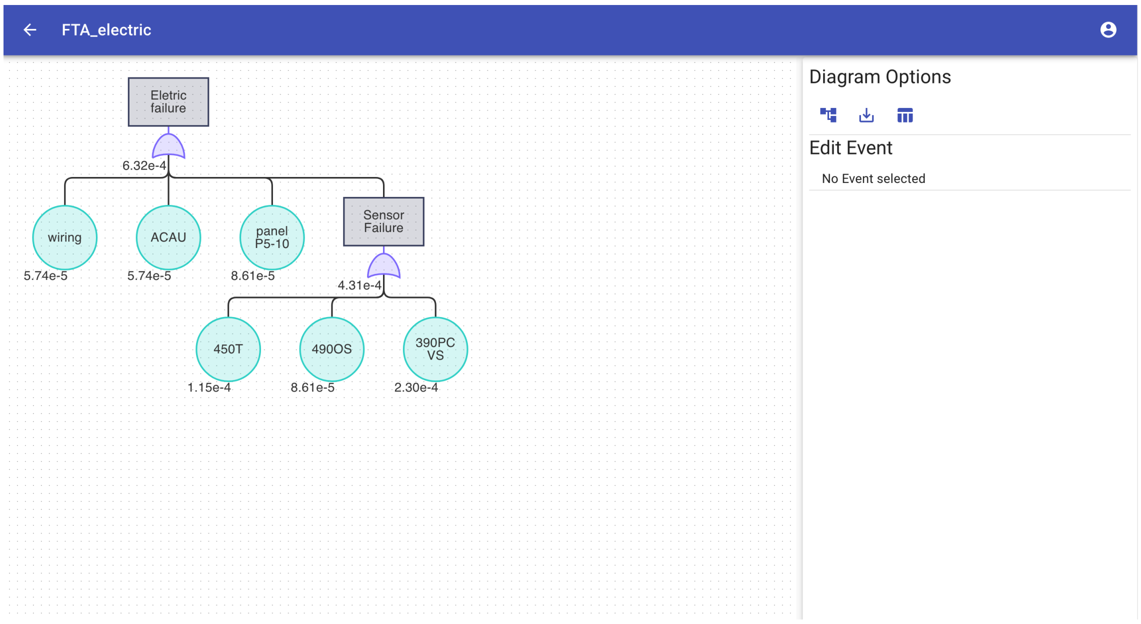
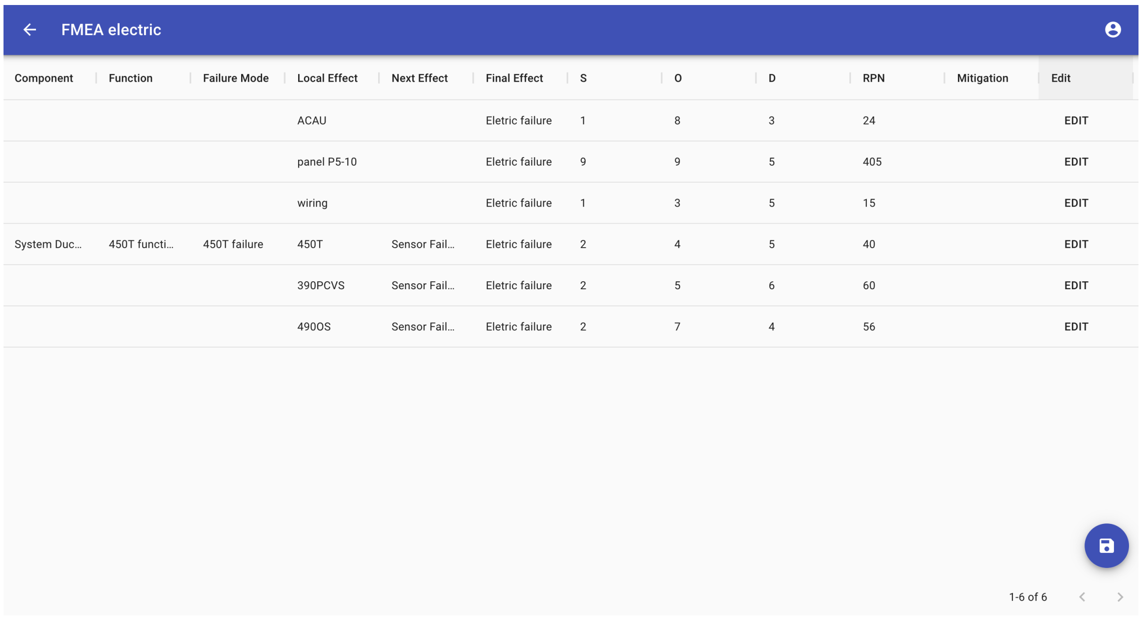


Figure 9.3: FTA Electric Failure

9.2.2 Mechanic Failure

The goal of the second analyst was to finalize the diagram of partonomies and to create FTA of *mechanic failure* which is a second branch necessary to create an overall analysis of pneumatic system.

The system partonomies were finalized as can be seen in Figure 9.7. As the application supports only *is part of* relation, the *ENGINE #2* lacks subcomponents as they are exactly same as for *ENGINE #1*. After the partonomies were complete, FTA of *mechanic*



FMEA electric

Component	Function	Failure Mode	Local Effect	Next Effect	Final Effect	S	O	D	RPN	Mitigation	Edit
			ACAU		Eletric failure	1	8	3	24		EDIT
			panel P5-10		Eletric failure	9	9	5	405		EDIT
			wiring		Eletric failure	1	3	5	15		EDIT
System Duc...	450T functi...	450T failure	450T	Sensor Fail...	Eletric failure	2	4	5	40		EDIT
			390PCVS	Sensor Fail...	Eletric failure	2	5	6	60		EDIT
			490OS	Sensor Fail...	Eletric failure	2	7	4	56		EDIT

1-6 of 6

Figure 9.4: FMEA Electric Failure

failure was entered through FTA editor and the results were again similar to the input data.

Similarly to tree dedicated to electric failure, the FTA of mechanic failure (Figure 9.5) was converted to an FMEA table (Figure 9.6).

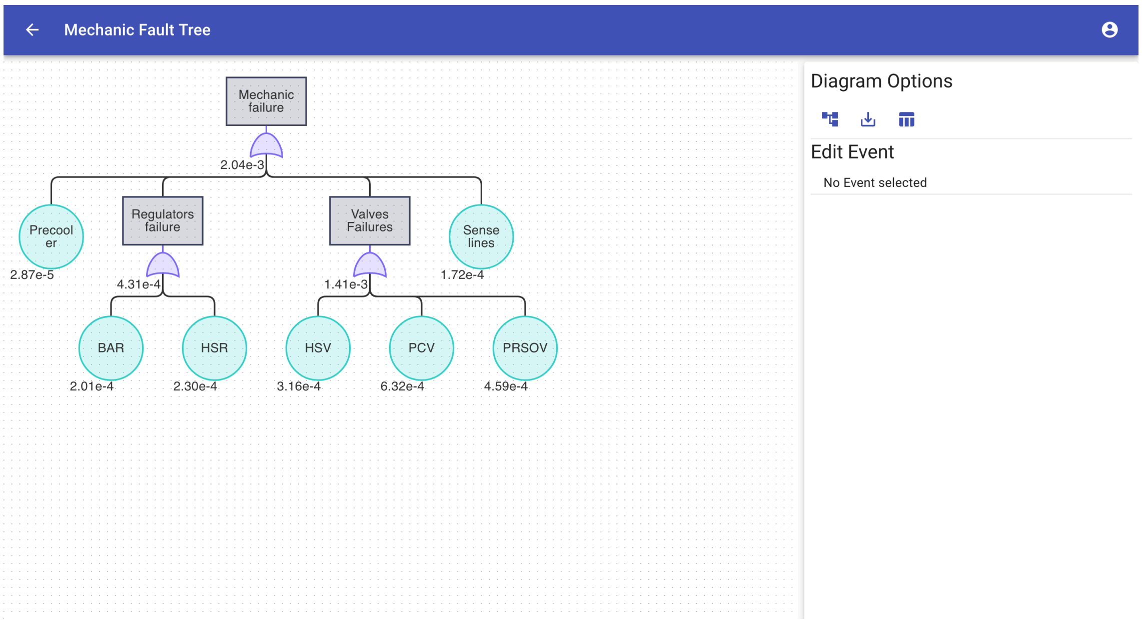
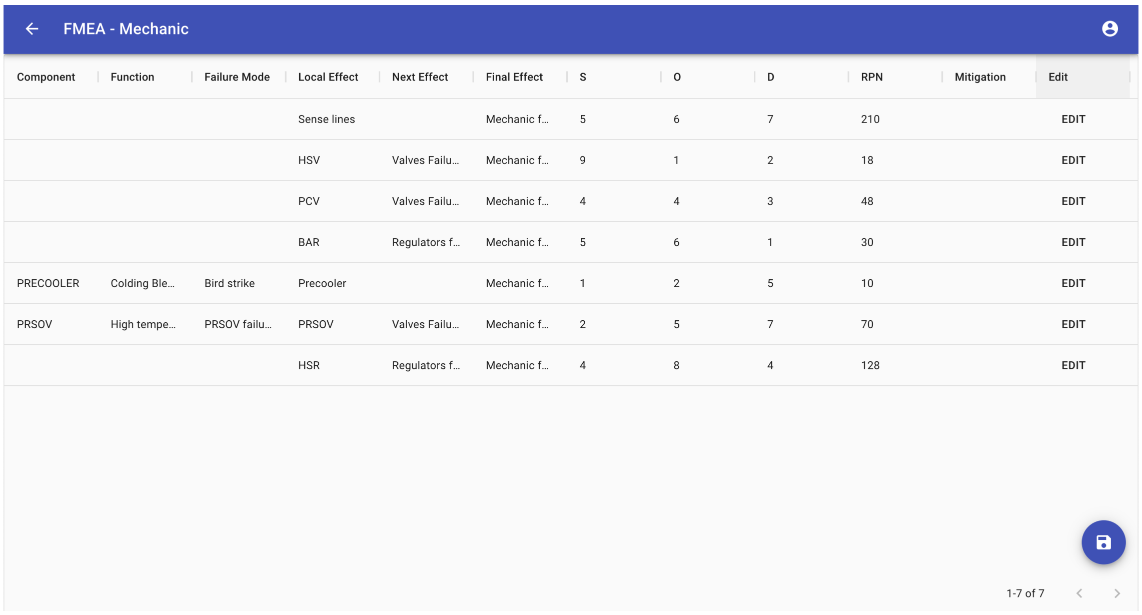


Figure 9.5: FTA Mechanic Failure

9.2.3 Pneumatic System

To finalize the verification, the last domain expert has taken over previously created diagrams and used a functionality to reuse existing events to create FTA composing them


FMEA - Mechanic

Component	Function	Failure Mode	Local Effect	Next Effect	Final Effect	S	O	D	RPN	Mitigation	Edit
			Sense lines		Mechanic f...	5	6	7	210		EDIT
			HSV	Valves Failu...	Mechanic f...	9	1	2	18		EDIT
			PCV	Valves Failu...	Mechanic f...	4	4	3	48		EDIT
			BAR	Regulators f...	Mechanic f...	5	6	1	30		EDIT
PRECOOLER	Colding Ble...	Bird strike	Precooler		Mechanic f...	1	2	5	10		EDIT
PRSOV	High tempe...	PRSOV failu...	PRSOV	Valves Failu...	Mechanic f...	2	5	7	70		EDIT
			HSR	Regulators f...	Mechanic f...	4	8	4	128		EDIT

1-7 of 7

Figure 9.6: FMEA Mechanic Failure

together. As the FTA input from Figure 9.1 shows, *pneumatic system failure* consist of two events regarding the *engine bleed failure* that are consisting of both *electric failure* and *mechanic failure* which prompted to reuse existing FTAs.

The analyst has created both engines' failures and plugged existing subtrees below. Unfortunately, as mentioned in Chapter 8, the application does not support reusing a subtree twice in the same tree due to technical issues caused by JOPA. The issues will be explained in Section 9.3. Nevertheless, FTA of pneumatic system failure was entered in the application with its one branch and the results were correct. The overall tree of pneumatic system is depicted in Figure 8.3 shown in Chapter 8.

The testing was then concluded by creating FMEA table aggregating all the FTA trees together in a one big table as can be seen in Figure 9.8.

9.2.4 Feedback

While the verification with domain experts helped to discover minor issues in the application, the overall feedback was positive. The application works as expected and the quantitative analysis calculates probabilities correctly. The analysts appreciated the combination of editors for system partonomies and FTA and then the convertibility of FTA to FMEA. Compared to commercial applications, the data were consistent and no redundancies as identified in the thesis of Ms. Bolčeková [1] were introduced. They further said that the analysis was significantly faster than with no dedicated solution and that they would like to continue using the solution in future.

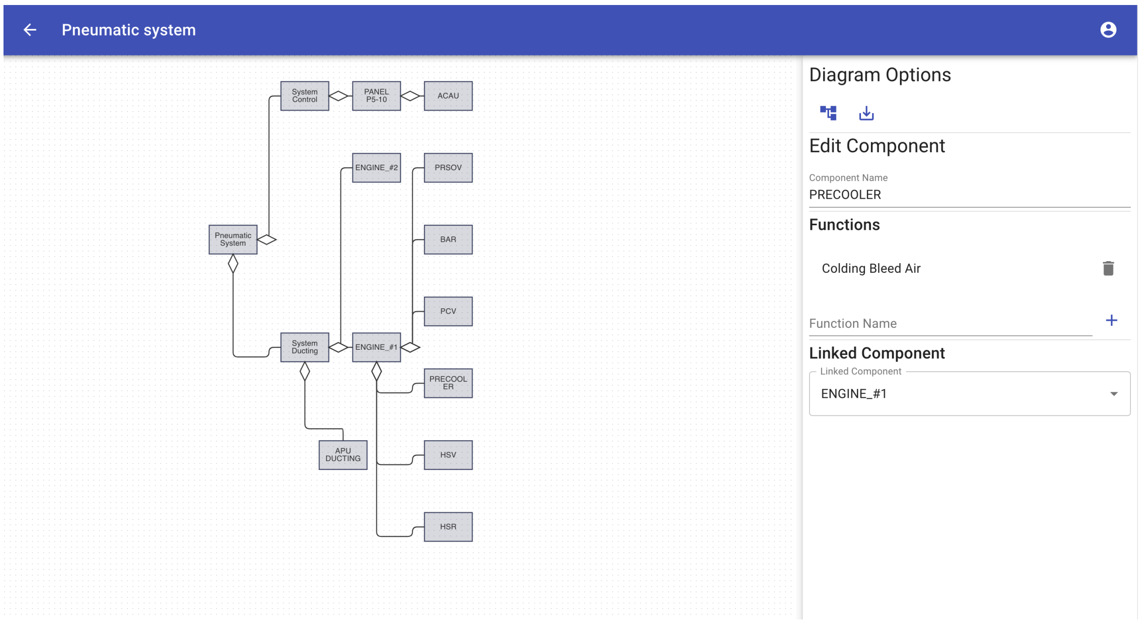


Diagram Options

Edit Component

Component Name
PRECOOLER

Functions

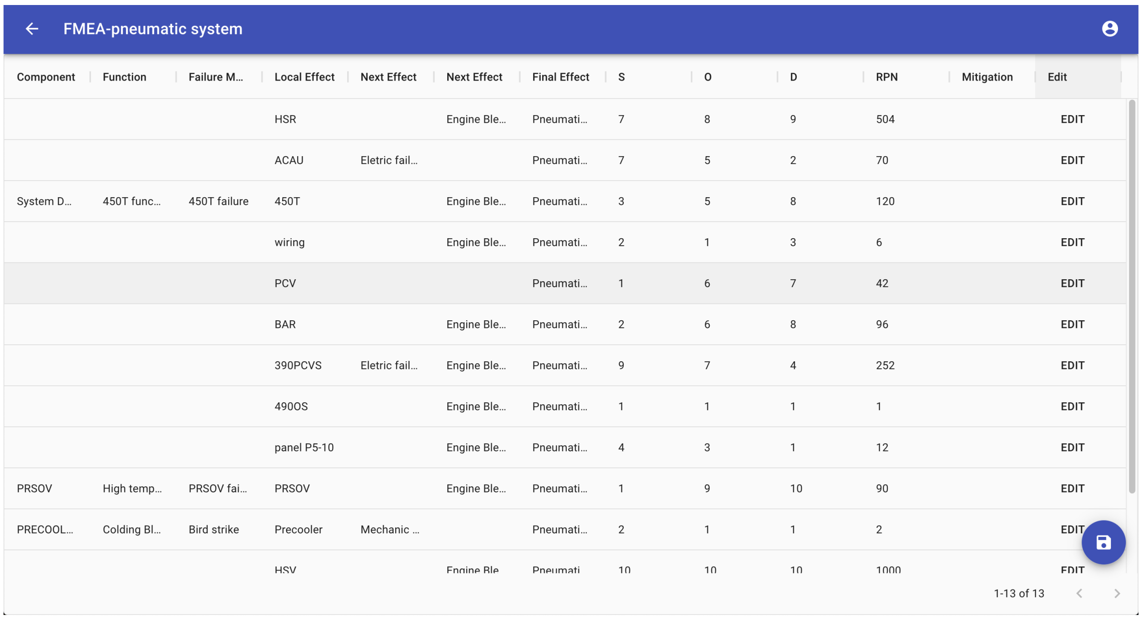
Colding Bleed Air

Function Name

Linked Component

Linked Component
ENGINE_#1

Figure 9.7: Pneumatic System Diagram



FMEA-pneumatic system

Component	Function	Failure M...	Local Effect	Next Effect	Next Effect	Final Effect	S	O	D	RPN	Mitigation	Edit
			HSR		Engine Ble...	Pneumati...	7	8	9	504		EDIT
			ACAU	Eletric fail...		Pneumati...	7	5	2	70		EDIT
System D...	450T func...	450T failure	450T		Engine Ble...	Pneumati...	3	5	8	120		EDIT
			wiring		Engine Ble...	Pneumati...	2	1	3	6		EDIT
			PCV			Pneumati...	1	6	7	42		EDIT
			BAR		Engine Ble...	Pneumati...	2	6	8	96		EDIT
			390PCVS	Eletric fail...	Engine Ble...	Pneumati...	9	7	4	252		EDIT
			490OS		Engine Ble...	Pneumati...	1	1	1	1		EDIT
			panel P5-10		Engine Ble...	Pneumati...	4	3	1	12		EDIT
PRSOV	High temp...	PRSOV fai...	PRSOV		Engine Ble...	Pneumati...	1	9	10	90		EDIT
PRECOOL...	Colding Bl...	Bird strike	Precooler	Mechanic ...		Pneumati...	2	1	1	2		EDIT
			HSV		Engine Ble	Pneumati	10	10	10	1000		EDIT

1-13 of 13

Figure 9.8: FMEA Aggregation

9.3 JOPA issues

The application delegates communication with GraphDB to JOPA library developed by KBSS group. While the dependency worked correctly, there were two issues discovered in library that slowed the development process down and one of them still limits application functionality.

9.3.1 Caching of Referenced Objects

First discovered issue relates to an application structure and a way how cache is managed by *EntityManager* in JOPA. The model often references other objects without defining *CascadeType* option. When a referenced object is then updated by another entity, a place where the object is referenced from retrieves an old version of the data introducing inconsistency in the information.

Luckily, this was resolved by turning off the JOPA cache which slows the application, but allows it to work correctly before it will be fixed in the library.

9.3.2 Querying Symmetric Data

A tree structure in the application is built by first referencing a top event which then has a link to its children and the approach continues similarly all the way to tree leaves.

As Figure 9.1 depicts, aviation system commonly consists of identical branches where the subtrees are imported and connected below an event.

Unfortunately, when modelling a symmetric tree structures in JOPA, the data are persisted correctly but the *EntityManager* then retrieves only one branch and the other one is left empty. This issue has been reported just recently and has not been fixed yet thus it limits a functionality of event reusing in the tree.

Due to these reasons, the solution currently allows user to reuse an event in the tree only if it has not yet been defined in the tree or in any of imported subtrees.

Chapter 10

Conclusion

The paper focused on an implementation of application to perform reliability analyses in an aviation domain. The goals were to select either FTA or FMEA and to develop a fully functional solution focusing on one of the techniques and to be extensible for the second method. The application should have been based on semantic web technologies and target web browsers.

A common drawback of existing applications is that they do rely on an unified model for the analyses that need to be combined together, which was shown to be erroneous when the analyst often migrated the data from one application to another. Such migrations have introduced data inconsistencies, duplicates and typos.

The issues mentioned above have served as a motivation to propose a new application that would target these drawbacks. The solution has primarily focused on an implementation of FTA technique and has employed an ontology that was based on existing ontological models that were extended and slightly amended to allow interchangeability of FTA and FMEA on a model basis. The ontology was then integrated in the server side solution with a help of JOPA OOM library communicating with GraphDB triplestore.

On the contrary to server side, a frontend application written in ReactJS was consuming data in JSON-LD format and supported the analyst by providing firstly an editor for system partonomies, then FTA editor for tree construction and lastly automatic tool for FTA to FMEA conversion which is then shown in a classic tabular form.

While most of the functionality was implemented, the application still lacks requirements marked as *won't have* from Chapter 7 which would improve the resulting contribution to reliability analysis tools. That primarily means user collaboration that is common when analysing complex systems, updating the trees by operational data to reflect real life usage and to provide qualitative analytical methods such as minimal cut sets.

Last but not least, the solution has discovered unexpected behavior in JOPA framework that temporarily limits its functionality regarding reusing of events across the trees.

That being said, the application was evaluated by domain experts involved in the development process of this paper. They have tested the solution on a real system data from an aviation domain. The verification process involved three analysts that worked separately to simulate cooperation on a single analysis. Their feedback has shown that the all-in-one approach (system partonomies, FTA and FMEA) powered by semantic web technologies significantly improved data consistency and overall usability of the tool.

10.1 Future Work

In the future, the application will continue to be used by analysts to test it on a daily basis. It will further extend its functionality to fully support the second analysis method and the FTA technique will be enriched by an employment of qualitative evaluation methods such as minimal cut sets for an identification of events combination causing the top event failure. The whole system would finally be introduced an analytical dashboard highlighting the important metrics.

Bibliography

[1] S. Bolčeková. *Reliability Analysis of Mechanical and Lubrication System of Aircraft Engine*. Czech Technical University in Prague, Jun 2019. URL `http://hdl.handle.net/10467/83427`.

[2] R. Holub and Z. Vintr. *Spolehlivost letadlové techniky: elektronická skripta*, page 233. VUT FSI, 2001. Translated to English.

[3] Jet Propulsion Laboratory. *Reliability Analyses Handbook*. Jet Propulsion Laboratory, Jul 1990.

[4] D. R. Vieira, M. L. Rebaiaia, and M. C. Chain. *The Application of Reliability Methods for Aircraft Design Project Management*. American Journal of Industrial and Business Management, 2016.

[5] J. Castet, M. Bareh, J. Nunes, S. Okon, L. Garner, E. Chacko, and M. Izygon. Failure analysis and products in a model-based environment. In *2018 IEEE Aerospace Conference*, pages 1–13, 2018. doi: 10.1109/AERO.2018.8396736.

[6] M. Stamatelatos and W. Vesley. *Fault tree handbook with aerospace applications*. NASA Office of Safety and Mission Assurance, 2002.

[7] Fault tree analysis - what are fault tree symbols. `https://www.smartdraw.com/fault-tree/`. Accessed: 2020-12-02.

[8] D. H. Stamatis. *Failure mode effect analysis: FMEA from theory to execution*, page 224. ASQC Quality Press, 2003.

[9] An example of fmea table. `https://www.researchgate.net/figure/An-example-of-FMEA-table_fig4_220838473`, Jan 2004. Accessed: 2020-12-02.

[10] M. Adamcová. *Hodnocení spolehlivosti metodou FMEA s využitím ontologického inženýrství*. České vysoké učení technické v Praze. Vypočetní a informační centrum., Aug 2020. URL `http://hdl.handle.net/10467/90652`.

[11] T. Gruber. Ontology (computer science) - definition. `http://tomgruber.org/writing/ontology-definition-2007.htm`, 2009. Accessed: 2020-12-06.

[12] N. Noy and D. Mcguinness. Ontology development 101: A guide to creating your first ontology. *Knowledge Systems Laboratory*, 32, 01 2001.

[13] What is an rdf triplestore? https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-triplestore/, 2012. Accessed: 2020-12-07.

[14] Antonio De Nicola and Michele Missikoff. A lightweight methodology for rapid ontology engineering. *Commun. ACM*, 59(3):79–86, 02 2016. ISSN 0001-0782. doi: 10.1145/2818359. URL https://doi.org/10.1145/2818359.

[15] R. Falbo. Sabio: Systematic approach for building ontologies. In *ONTO.COM/ODISE@FOIS*, 2014.

[16] G. Guizzardi. *Ontological Foundations for Structural Conceptual Models*. PhD thesis, 01 2005.

[17] G. et al. Guizzardi. Towards ontological foundations for the conceptual modeling of events. In Wilfred Ng and Juan C. Storey, Veda C.and Trujillo, editors, *Conceptual Modeling*, pages 327–341, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-41924-9.

[18] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Formal Ontology in Information Systems. Proceedings of FOIS'98*, pages 3–15. IOS Press, 1998.

[19] Semantic web. https://www.w3.org/standards/semanticweb/, 2015. Accessed: 2020-12-06.

[20] M. Obitko. Semantic web. https://www.obitko.com/tutorials/ontologies-semantic-web/semantic-web.html, 2007. Accessed: 2020-12-06.

[21] Rdf 1.1 primer. https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/, 2014. Accessed: 2020-12-06.

[22] Owl quick reference guide. https://www.w3.org/2007/OWL/wiki/Quick_Reference_Guide, 2012. Accessed: 2020-12-06.

[23] Sparql 1.1 query language. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/, 2013. Accessed: 2020-12-07.

[24] What is sparql? https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/. Accessed: 2020-12-07.

[25] M. Ledvinka, P. Křemen, , and B. Kostov. Jopa: Efficient ontology-based information system design. In H. Sack, editor, *The Semantic Web: ESWC 2016 Satellite Events*, pages 156–160. Springer, 2016.

[26] Jopa - kbss. https://kbss.felk.cvut.cz/cs/jopa. Accessed: 2020-12-08.

[27] A. Majdara and T. Wakabayashi. A new approach for computer-aided fault tree generation, 2009.

[28] A. Venceslau, R. Lima, L. A. Guedes, and I. Silva. Ontology for computer-aided fault tree synthesis. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–4, 2014. doi: 10.1109/ETFA.2014.7005334.

[29] F. Drastich. *FTA Conceptual Model and its Utilization for Aircraft Reliability Evaluation*. Czech Technical University in Prague, Aug 2020. URL http://hdl.handle.net/10467/90677.

[30] S. Hatton. Choosing the right prioritisation method. In *19th Australian Conference on Software Engineering (aswec 2008)*, pages 517–526, 2008. doi: 10.1109/ASWEC.2008.4483241.

[31] J. B. Fussell, E. F. Aber, and R. G. Rahl. On the quantitative analysis of priority-and failure logic. *IEEE Transactions on Reliability*, R-25(5):324–326, 1976. doi: 10.1109/TR.1976.5220025.

[32] K. Mündel. *ATA36 Pneumatic System Reliability and Maintenance for B737NG Operators*. Czech Technical University in Prague, Sep 2020. URL http://hdl.handle.net/10467/90373.

Appendix A

List of Abbreviations

FT	Fault Tree
FTA	Fault Tree Analysis
FMEA	Failure Modes and Effects Analysis
RBD	Reliability Block Diagram
PRA	Probabilistic Risk Assessment
RPN	Risk Priority Number
SPOF	Single Point of Failure
NASA	National Aeronautics and Space Administration
JPL	Jet Propulsion Laboratory
UML	Unified Modeling Language
JOPA	Java OWL Persistence API
W3C	World Wide Web Consortium
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
OWL	Web Ontology Language
URI	Uniform Resource Identifier
JSON-LD	JSON for Linking Data
KBSS	Knowledge-based and Software Systems Group
EFTA	Emerging Technology and Factory Automation
DAG	Directed Acyclic Graph
CSV	Comma-separated Values
UFO	Unified Foundational Ontology

JWT	JSON Web Token
RAM	Reliability, Availability, Maintainability analysis
BFS	Breadth First Search

Appendix B

List of Figures

Appendix C

Attached Files

The attached files contain source codes for both frontend and backend applications.

File	Description
`fta-fmea`	Source codes of backend application.
`fta-fmea-ui`	Source codes of frontend application.

Table C.1: List of attached files.