Fault Tree Synthesis from Capella Models

Manzi Aimé Ntagengerwa

University of Twente, Enschede, The Netherlands
m.a.ntagengerwa@utwente.nl

Abstract

Model-Based Systems Engineering (MBSE) plays a critical role in the design and analysis of Cyber-Physical Systems (CPSs), where early detection of potential system failures is essential. This paper presents a methodology for inferring Fault Trees (FTs) from models created in Capella, a widely adopted MBSE tool. This transformation goes through a knowledge graph (KG) as an intermediate step. The approach facilitates risk assessment during the early stages of system development by generating preliminary FTs from high-level design artifacts. As the system design evolves, these FTs are incrementally refined, allowing for iterative risk analysis and design improvement. We argue that automated FT generation from Capella models enhances the MBSE workflow by integrating risk assessment in the early design stages.

1 Introduction

Cyber-Physical Systems (CPSs) bring together software, hardware, and physical processes – often running concurrently and interacting in real time. Designing these systems is a complex task that involves many disciplines. Each group of experts brings its own modeling language: mechanical engineers use blueprints, physicists use equations, electrical engineers use circuit diagrams, and so on. This is because these experts from different domains all have different conceptual models of the part of the CPS that they design. And not only this; even within a discipline, experts will often express the same concepts in different languages. This makes it difficult to reason about the system as a whole – especially when components interact and operate concurrently.

Model-Based Systems Engineering (MBSE) aims to bridge this gap by providing structured, traceable system models. In this work, we use the Capella workbench (based on the Arcadia method) [1], which supports stepwise refinement from high-level requirements down to component-level architecture. Capella is particularly well-suited to managing the layered abstractions and interdependencies in CPSs, including concurrent behaviors.

Alongside correctness, safety and reliability are key concerns. Techniques like Fault Tree Analysis (FTA) [2] are commonly used for risk assessment, but fault trees (FTs) are typically created manually, based on expert discussions. This is tedious, error-prone, and hard to keep in sync with changing system models—especially in early design phases, where iteration is fast and frequent.

We propose a method for the automatic synthesis of FTs from Capella models, through the use of knowledge graphs (KGs). A KG is a graph data structure that represents a network of facts. KGs support reasoning and querying, and their structure can be defined with an ontology (a conceptual model of the domain). One of the KG's key strengths is the integration of facts from different sources by providing a standardized format.

This approach enables early-stage risk analysis that evolves with the design. The methodology supports the process of iterative refinement, where insights from FTA can guide design decisions to make the CPS more robust.

2 Methodology

Figure 1 shows the transformation and envisioned context of iteratively refining Capella models based on synthesized FT insights. In this workflow, the engineer creates a Capella model of a CPS. They then generate FTs from this Capella model. Analysis of these FTs provides risk metrics, which inform the design decisions the engineer takes.

The transformation of a Capella model to FTs takes two steps. The first step is transforming a subset of the Capella model to a knowledge graph (KG). The second step is transforming this KG into FTs.

For step 1 (transforming Capella *models* to KGs), we must identify a valid mapping from Capella *concepts* to the ontological concepts shown in Figure 2.

For the second step of the transformation we leverage our previous work, which focuses on the transformation from KGs to FTs [3]. The transformation result of step 1 needs to conform to the structure shown in Figure 2 in order to apply this existing work for step 2.

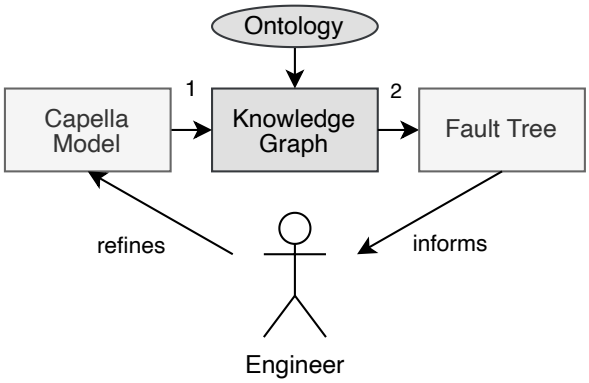

Figure 1: The proposed workflow of FTA-informed Capella model refinement.

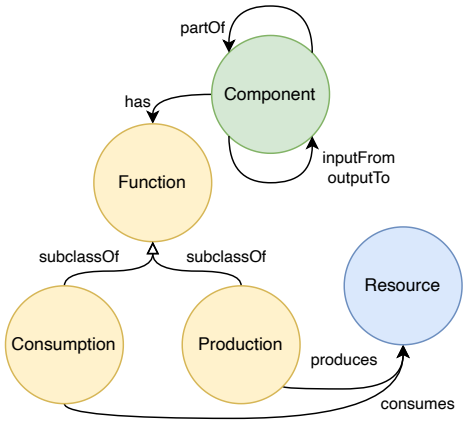

Figure 2: The KG's ontology (conceptual model) [3]

The ontology only describes concepts that are common in CPS design (**components**, **functions**, **resources**). Faults and failure can be inferred from only this compositional and functional information. Similar concepts are described in Capella as *Notional Components*, *Functions* and *Exchanged Items* [4]. We extract the elements to be mapped from Capella files with `capellambse` [5] and use `SPARQL` [6] queries to construct the KG.

In the early design stages, the Capella model is very high-level. Commonly, large black-box subsystems and their composition are specified first. For instance, an aircraft may be modeled as a fuel subsystem, an engine and propellers, without specifying further internal details. Meaningful FTs can already be generated from this high-level design. As these black-boxes get iteratively refined into subcomponents and subfunctions, the generated FTs also become more detailed. For example, the fuel subsystem might be refined to consist of a fuel tank, a fuel sensor and a fuel pump. The composition of subcomponents and subfunctions is analogous to the composition of sub-FTs.

References

[1] Pascal Roques. MBSE with the ARCADIA method and the capella tool. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[2] Enno J.J. Ruijters and Mariëlle I.A. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 2015.

[3] Manzi Aimé Ntagengerwa, Georgiana Caltais, and Mariëlle Stoelinga. Fault tree synthesis from knowledge graphs. In *RAMS-Europe*. IEEE, 2025. (Accepted for publication).

[4] Jean-Luc Voirin. Arcadia reference: Engineering data model. Technical report, Thales, 2023.

[5] DB InfraGO AG. Python-capellambse, 2019. [Accessed 04-06-2025].

[6] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language, 2013. [Accessed 04-06-2025].