Evaluation of the smartIflow approach to model-based safety analysis

Oliver Schmid
University of Applied Sciences Ulm
Intelligent Systems
schmol03@thu.de

Abstract

Safety Analysis, the verification of system designs, is a task that can be of critical importance. Traditional approaches such as Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA) are reaching their limits due to growing system complexity. The more recent model-based approaches to safety analysis seem to have fewer problems with complexity due to the use of automated model checking techniques. One approach to model-based safety analysis is smartIflow [1]. In this paper, the smartIflow approach is evaluated based on the pressure tank system design, as it is described in the Fault Tree Handbook [2]. Part of this evaluation is the examination of the generated FTA and FMEA results. Further, it will be described how an explosion of the state space affects different analysis tasks, and how it can be mitigated by the means of smartIflow.

Keywords: *Safety Analysis, smartIflow, Model-based Safety Analysis, Model Checking, FMEA, FTA*

1 Introduction

In system development, the verification of the system design and the verification of reliable operation is a task of critical importance. One reason for the importance of safety analysis is the cost of correcting errors in the design. Those costs increase as the development process progresses and system complexity increases. Apart from monetary reasons, rigorous implementation of safety analysis is a absolute necessity, since for instance in the aviation domain, human lives depend on the correct and complete verification of system designs. A possible solution to address increasing complexity and to increase the reliability of the system verification are model-based approaches to safety analysis, such as smartIflow. In this paper, the smartIflow approach to model-based safety analysis will be evaluated, in order to find out, whether it is a suitable solution to verify real designs.

Structure To convey the basic idea behind Safety Analysis and approaches to performing it, we will discuss Safety Analysis and the traditional approaches to Safety Analysis (Fault Tree Analysis and Failure Mode and Effect Analysis) in section 2. In addition to that, section two introduces Model-Based Safety Analysis, the general approach behind smartIflow. A detailed description of smartIflow, and its approach to Model-Based Safety Analysis, is provided in section 3. The Evaluation of the smartIflow approach to Model-Based Safety Analysis takes place in section four. Therefore, section 4 contains a description of the Pressure Tank System, which will be used as a case example. Following this case example, we will apply an existing process to conduct model-based safety analysis. This process includes the modeling of the nominal system, the definition of derived safety requirements and the modeling of faults. At the end of this process, we will describe and analyze the generated results. In the Results chapter 5, we will describe whether the generated analyses are sufficient and as expected. Furthermore, we will discuss whether the means regarding the state space management in smartIflow are sufficient.

2 Safety Analysis

The general goal of safety analysis is to prevent system failures to attain a mandatory level of safety. Safety analysis is thus concerned with analyzing the structure of systems and identifying potential causes of error. The traditional approaches to safety, namely Failure Mode and Effect Analysis and Fault Tree Analysis, stem from the 20th century and were developed by or on the behalf of the U.S. military. As a part of the "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment" (ARP4761), they are used in conjunction with other methods to ensure a necessary level of safety in the aviation domain. In both of these approaches, systems are analyzed by a team of professionals applying specific methodologies. A potential drawback of these approaches is that the identification of potential causes of failure, and thus the reliability of the system, depends on the skills of the analysts performing the safety analysis. Furthermore, the effort of the traditional approaches increases strongly with increasing system complexity, since they are executed manually. A more recent approach is Model-Based Safety Analysis, in which a model of the system are used in conjunction with a description of the expected system behavior to find potential failures. Model Checking and Model-Based Safety Analysis is considered a subset of formal methods and aim to prove the validity of a system's functionality with mathematical accuracy [3].

2.1 Failure Mode and Effect Analysis (FMEA)

FMEA is a method for identifying and assessing different failure modes and effects. While the failure mode describes the way in which a component or systems fails, the failure effect describes the consequences of the failure for different levels. The Process to conduct a Failure Mode and Effect Analysis [4, 5] involves the assembly of a team of domain experts. This team of experts identifies potential failure modes, associated failure causes and failure effects. For each failure mode a Risk Priority Number (RPN) is calculated based on the severity, the frequency of occurrence and the ease of the detection of the failure. This metric is then used to address identified issues and calculated once again to measure the effect of the improvements [5]. Since the safety analysis in FMEA begins at the component level and ends with the analysis of effects at system level, FMEA is considered an inductive approach to safety analysis.

2.2 Fault Tree Analysis (FTA)

The fault tree, the most important part in a FTA, is a diagram illustrating how various components (and their failure) affect the system and its overall operational state. The top event (TOP) of a fault tree represents the considered failure and is connected via logical gates to basic events that represent individual component failures. FTA is about analyzing this fault tree using qualitative and quantitative methods. Quantitative approaches focus on calculating probabilities related to system failures, i.e. the probability of a system failure based on the probabilities of basic component failures [6]. Qualitative approaches on the other hand focus on analyzing the structure of the failure tree. A known method for doing so is the identification of minimal cut sets, i.e., the smallest possible subsets of the failure tree leading to the considered failure (TOP event) [6]. Other qualitative approaches include the identification of minimal path sets and common causes of failures [7]. In contrast to FMEA, FTA starts from a failure at the system level and analyzes its cause downwards to the component level. It is therefore considered a deductive approach to safety analysis.

2.3 Model-based Safety Analysis (MBSA)

Conducting a model-based safety analysis involves describing a system in terms of a system model and testing it against a predefined specification using a model checker.

Model The system model is a description of the system and its components. Models are usually subject to a certain level of abstraction, which depends on the goal of the modeling task [8].

It thus reflects safety-relevant characteristics of the system and is designed by a development engineer. Parts of the model may be taken from models created in context of other system design tasks. The modeling itself is performed using a modeling language that is compatible with the Model Checker. Modeling languages, such as the ones used by AltaRica or smartIflow, allow the description of components and their behavior in languages that include syntax elements known from existing programming languages. Furthermore, the two languages also allow the Utilization of object-oriented paradigms such as inheritance.

Specification "Specification models are used to communicate the goals and requirements of a system among different engineering teams" [8]. Specifications therefore consist of expressions describing the expected system behavior and are usually defined using temporal logic, namely either Linear-Time Temporal Logic (LTL) or Computation Tree Logic (CTL). In both LTL and CTL, time is described discretely [9], i.e., the specification describes the general ordering of occurring states (s_0 is followed by s_1), rather than using an actual time measurement in seconds or the like. LTL and CTL utilize logic operators (e.g. conjunction, disjunction, negation) to describe logical conditions denoted by ϕ, as well as temporal operators. Temporal operators describe that a logical condition (ϕ) has to hold

- in the next state ("next time"): $X\phi$
- on the entire path ("globally"): $G\phi$
- eventually ("finally"): $F\phi$
- until some other condition holds ("until"): $\phi\ U\psi$
- until some other condition maybe holds ("weak until"): $\phi\ W\psi$

[1, 9, 8]. LTL allows the definition of expected linear sequences of events, meaning in a visualization of system states it considers a single path of (future) events. CTL on the contrary is a branching time logic [9] and therefore allows the description of multiple possible future paths the system can end up in. Computation trees can be "[...] obtained as tree unfoldings of discrete transition systems [(e.g. finite state machines)] and represent the possible infinite computations arising in such systems" [9]. CTL expressions always consist of two parts: The path quantifier and previously described temporal conditionals. The path quantifier indicates whether

- there must exist at least one path that satisfies the temporal condition (E)
- the temporal condition must hold on all paths (A)

[8].

Model Checker Testing whether the system model M fulfills a specification Φ ($M \models \Phi$?) is the objective of the model checker [1, 10]. The model checker must therefore examine all possible states the system can end up in and check whether the specification given by the user is violated at any point. If the model checker finds such a case, in which the system states violate the specification, this so-called counterexample is returned to the user [8]. By analyzing the counterexample, the user can trace how the system got into the faulty state. In automata-theoretic approaches to verification, such as the smartIflow approach, the model checker uses the system model to build up some kind of finite automaton such as a state machine. The generated automaton is then used to search for paths that are acceptable for the state machine but not allowed by the specification term. To do so, the automaton is "unwinded" into a tree-like structure representing possible system states. Summed up, the Model Checker can check a Model against a specification by applying a "[...] automata-theoretic approach to verification, [...] [that] reduce[s] questions about systems and their specifications to questions about automata" [8].

State Explosion Problem The state space explosion problem describes the exponential relationship between system behavior and the number of possible states the system can enter [11]. State space explosion becomes problematic, when the computing resources required to represent the system states exceed the computer resources that can be reasonably allocated. Therefore, it is necessary to utilize methods that reduce the number of considered states.
One such method mitigating the increase of state space is partial order reduction. Partial order reduction utilizes knowledge about the order of events to identify sequences of states that lead to the same state regardless of their order. By knowing about sequences that lead to the same state, the number of paths to be checked can be reduced [12].

Failure Modeling Failure Effect Modeling (FEM) and Failure Logic Modeling (FLM) are ways to account for component failures in safety analysis. Failure Effect Modeling describes the effects of a failure by modeling the effects of the failure on other components by adjusting output values accordingly. Therefore, using FEM means that the system model must include information about, for example, energy flows that can be used to propagate failures throughout the system. This means, that actual flow dependencies have to be modeled, what might lead to increased complexity and a thus increased need for computational resources for the simulation.
When Failure Logic Modeling is applied, a fault model is created which contains information about the way components fail. An example for a tool that uses this approach are Hierarchically Performed Hazard Origin & Propagation Studies (HiP-Hops). HiP-Hops start from a system model that includes a description of how components fail and how their failure impacts other components. Based on this description, fault trees are then calculated and analyzed to create an FTA and FMEAs [13]. Lisagor summarizes as follows: "The key idea behind both methods [(Hazard and Operability Analysis and HiP-HOPS)] is the specification of causal relationships between deviations of component outputs on the one hand and internal failures of the components and input deviations on the other" [14]. Therefore, in FLM failures are represented by describing how a component is affected by the failures of the component itself and the failures of related components.

Orders of Magnitude and the modeling of Power Networks When quantitative values are described in orders of magnitudes, they are described by converging them to the closest reference value. These rough representations of numbers already allow the consideration of some physical effects, as it is often already sufficient to know that one value dominates another value.
In "Qualitative Order of Magnitude Energy-Flow-Based Failure Modes and Effects Analysis" [15] a qualitative approach to modeling (e.g. electrical) power networks is presented. By using a order of magnitude representation, the components behavior and failure effects associated with flows can be depicted without the representation of real values and actual physical conditions. The representation of power networks as proposed in said approach is based on generalized physical variables, namely effort (e.g. voltage, pressure or force), flow (e.g. current, volumetric flow rate or velocity) and resistance [15]. Components are represented by their resistance and thus characterized by their power absorption. The system under consideration is therefore modeled in the form of a graph labeled with those component-describing resistance values. Said graph is then used to simulate the system by "[...] [deriving] effort across each resistance and the flow through each resistance given a power (effort or flow) source in the network" [15]. To derive the effort and flow values of the components, the network is reduced to a "single equivalent resistance" value by applying series and parallel (SP) reduction techniques and the star-delta transformation where SP reduction is not applicable [15]. The flow values for the components are then calculated based on the previously calculated single equivalent resistance value and an overall flow value [15].

3 smartIflow

The smartIflow (State Machines for Automation of Reliability-related Tasks using Information FLOWs) modeling language and associated tools were developed with the goal of automating the safety analysis process [1]. Hönig et al. characterize smartIflow as a modeling language with a rather high level of abstraction, that uses undirected connection modeling, allows failure effect modeling and failure logic modeling, it also enables the partial reuse of design models [1].

Model Description In smartIflow components of a system are described in the form of object-oriented classes describing their properties and behavior. An example of a component modeled in smartIflow can be found in Listing 2. In the following, the individual smartIflow sections, that are used to describe components will be discussed:

- Components: The components section is used to instantiate potential subcomponents.
- Ports: Ports are used to connect components with each other. Port definitions include the definition of a data type, specifying the values that can be transmitted. Component-connections can be defined as directed connections by using a generic type to mark them as either Input or Output. See Listing 1 for an example of a port definition.
- Variables: State variables are a necessity to depict state specific behavior and are thus defined in the variables section. The definition of a state variable includes the specification of the (user-defined) data type, the variable name and an initial value [16].
- Events: Components can be affected by internal or external effects, such as user interactions or component failures. Those user or event-driven interactions are described in the events section.
- Behavior: State-dependent behavior is described by using set and connect functions. The set function allows publishing properties at specific ports (e.g. set powered to false in a mechanical port, see 1), the connect function is used to connect ports (e.g. electrical ports). To associate the correct behavior with the current state, if statements are used to detect the current state.
- Transitions: The transitions section is about the description of state changes. Transitions are described using when-clauses, that take a Boolean condition as a parameter and allow code to be executed when that condition becomes true.
- EventHandlers: While state changes caused by internal (with respect to the system) events are handled by the transitions section, the EventHandlers section handles state changes caused by external events. An example for an external event would be intervention by the user or a failure of the component.

[17].

```
port Mechanical {
    Properties:
        powered = {True, False};
}
```

Listing 1: Definition of a Port-type using the smartIflow modeling language.

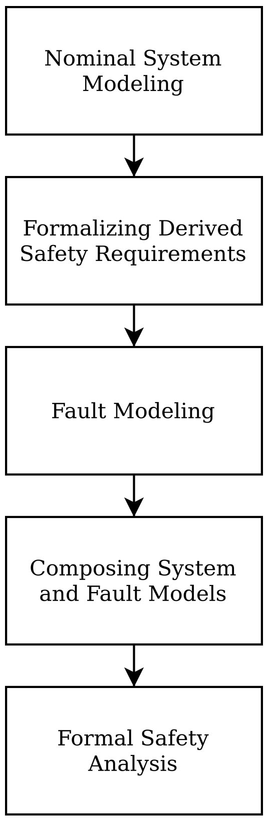
Model Checking smartIflow performs model checking by using system models in combination with a explicit-state model checker and a CTL specification.
The model checking process includes an initialization phase, an unfolding phase (sometimes referred to as simulation phase) and a requirement verification phase. In the initialization phase, components are initialized to their default state values. Starting from this initial state, the possible system states are unfolded step-wise. Each unfolding-step consists of the following sub-steps and yields a "[...] set of event-state pairs covering all possible subsequent states" [1].

- Network reconfiguration: During the network reconfiguration step, the component-specific Behavior section is processed. This means, that connections defined by the connect functions are created and based on the set functions, properties are published using the Ports of the component [1, 18].
- Flow direction determination: Determination of the flow directions using a qualitative solver and propagation of port properties [1, 18].
- Event processing: The event processing step is about the internal and external events defined in the Transitions or EventHandlers section respectively [1, 18].

Based on this procedure, smartIflow builds up a tree-like graph representing possible system states. Generating this graph can quickly become a resource-intensive task, leading to the State Explosion Problem mentioned earlier. To mitigate this problem, smartIflow provides means to narrow down the generated graph by allowing the specification of an Event Trigger Specification. This Specification enables the Model Checker to stop the unfolding of the graph based on another user-provided Formula [18].
The Model Checker traverses the previously unfolded graph in the Requirements Verification Phase to check it using the user-provided CTL specification. If the Model Checker detects any violations of the user-provided specification, it returns all violations as counterexamples [1].

4 Safety Analysis of the Pressure Tank System with smartIflow

To evaluate the smartIflow approach to model-based safety analysis, the MBSA process as described by Joshi et al. [10] is applied. Therefore, the first step is the modeling of the system and component behavior, i.e. the description of the system model. Afterwards, the expected system behavior is defined, i.e. the specification reflecting safety requirements. Subsequently, component faults and their effect on other components are modelled as part of the Fault Modeling step.



Nominal System Modeling
Formalizing Derived Safety Requirements
Fault Modeling
Composing System and Fault Models
Formal Safety Analysis

Figure 1: Model-Based Safety Analysis Process, as presented by Joshi et. al [10].

The initially created system model is then extended to include these faults (Composing System and Fault Models step) by injecting them into the model definition itself or by automatically applying a separately defined fault model. In the last step (Formal Safety Analysis), the actual analysis is conducted.
Before the description of the applied MBSA process, the pressure tank system is first introduced as it is described in the Fault Tree Handbook [2].

4.1 Pressure tank case example

A pressure tank is a system used to generate and maintain a desired pressure level at an output valve. In the pressure tank system, pressure is stored in a pressure tank (7), released through an outlet valve (6) and generated by a pump (9). The pump itself receives power from an electrical motor (8). The motor is part of a circuit we call motor circuit in this paper. This part of the circuit is marked yellow in figure 2. Whether the motor in the motor circuit is powered, is controlled by Relay K2 (3) which is part of the upper circuit, we call control circuit (marked blue in figure 2). By activating Switch S1 (1), Power is applied to Relay K1 (2) which then closes its contacts. Since closure of Relay K1 contacts are an alternative way to power the relay to a constant button press, Relay K1 (2), Relay K2 (3) and the Timer Relay (4) now constantly receive power from the lower end of the power supply (Relay K1 is "self-latched" [2]). As soon as Relay K2 (3) receives power, it closes the motor circuit and therefore engages the motor (8) powering the pump (9), which will then pump fluid into the pressure tank (7). After the pump pumps fluid into the tank for 60 seconds, the pressure switch (5) is activated since it detects excess pressure. The description in the handbook do not mention, whether that depends on the current fluid or pressure level of the tank. It is therefore assumed, that the current fluid or pressure level is not relevant for the time it takes to fill the tank e.g. because the tank is drained and pressure released before the pump starts pumping. The pressure switch then interrupts the connection to the K2 (2) relay, therefore de-energizes the components in the control circuit (2, 3, 4) and thus stops the motor (8).

Emergency shutdown The purpose of the timer relay (4) is to stop the system in case the pressure switch fails. If the Timer Relay is constantly powered for 60 seconds, it will interrupt the connection to Relay K1 (2) and resets the system as a result.

Assumptions The pressure tank case example comes with the following assumptions [2].

- The pressure switch opens its contacts as soon as the maximum pressure is reached and otherwise closes its contacts.
- The Timer Relay resets its timer after every activation.
- The outlet valve drains the tank in negligible time.
- The reservoir can provide an unlimited amount of fluid.

4.2 Nominal System Modeling

At the start of a modeling task, it makes sense to make a distinction between (flow) connections that must be represented by actual simulation of flows and connections that can be simulated by using (e.g. Boolean) properties. For the fluid connections (reservoir to tank), it is already sufficient to use the port properties to indicate whether fluid is running. The reason for this is that the system description doesn't mention any throughput limitations of the valves. Also, the amount of fluid required to pressurize (or damage) the tank is not specified in terms of volume, but instead a time constraint. When it comes to the modeling of the circuit, it is necessary to determine whether it is necessary to take different resistance or voltage levels into account. This would be the case, if, e.g. a short circuit is to be modeled. As the descriptions of the pressure tank system do not contain any information about resistance levels or the like, it would be in theory possible to model the contained circuit completely by using properties instead of flow connections. One indication that it is advisable to model the system by modeling actual flow connections is the fuse contained in the motor circuit, that prevents other contained components from getting damaged due to excessively high voltage

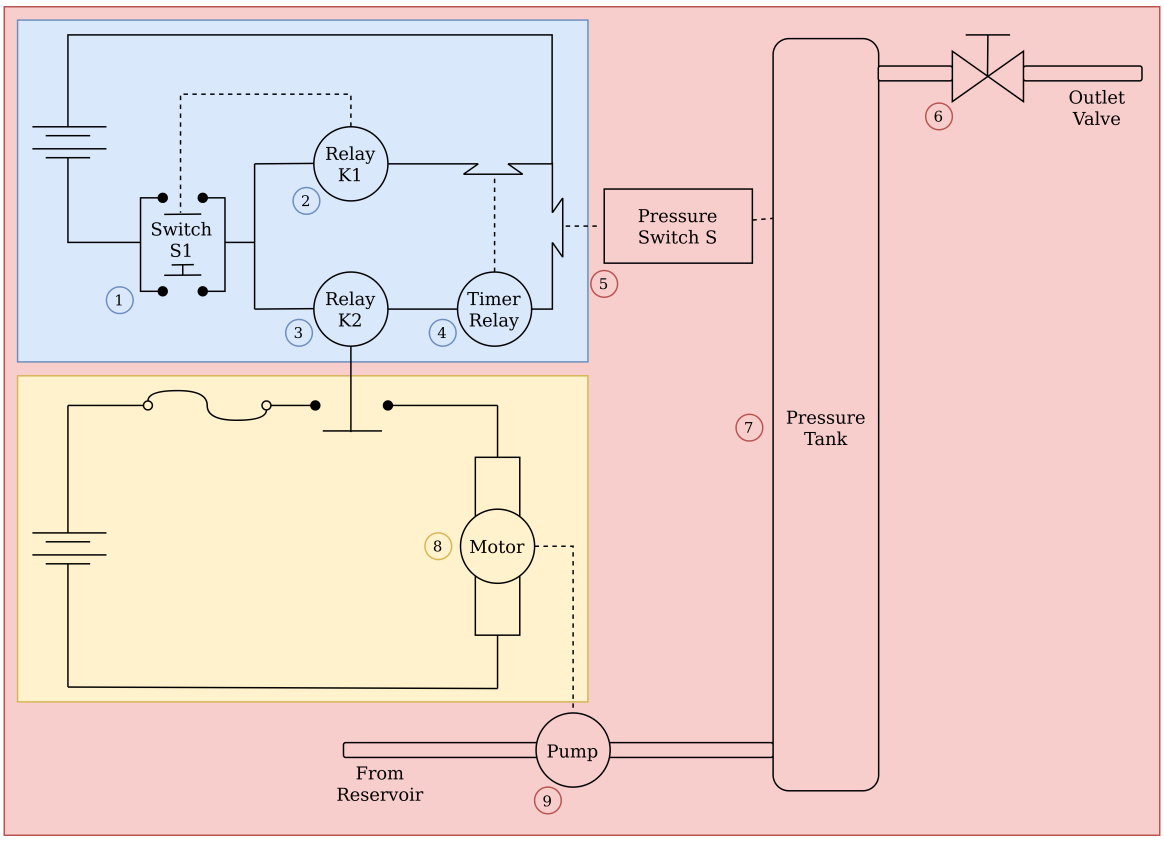


Figure 2: Diagram of the pressure tank system, depicted according to the model presented in Fault Tree Handbook by Haasl et. al [2].

levels. Since the descriptions of the system do not contain information about the behavior or possible failures of the fuse, this is negligible here. The primary reason for which we will model the circuit using flow connection is to determine whether smartIflow can calculate the energy flows correctly.

In software development, the Don't Repeat Yourself (DRY) principle describes, that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system" [19]. The purpose of DRY principle is to make code as reusable and maintainable as possible by applying concepts such as inheritance or composition. Due to the object-oriented nature of smartIflow, it makes sense to apply this principle by examining the components for commonalities in properties and behavior. One similarity when it comes to the electrical components is their ability to receipt electricity. The ElectricalComponent class is defined, to describe these characteristics (see figure 2). Further, many of the electrical components forward received power by connecting their electrical ports. The ForwardingElectricalComponent class therefore extends the previous mentioned class by a behavior section describing how electrical components forward electricity (see listing 3). Differentiating between these classes makes sense, since for instance, the electrical pressure switch does not forward electricity if it has not been pressed. The use of inheritance makes component descriptions clear and concise, as in the motor example (see figure 4).

Temporal sequence modeling In smartIflow, temporal sequences can be modeled by describing the order in which states occur in terms of magnitudes of time. Consequently, it is necessary to think about what behavior is tied to which duration. One thing to keep in mind, is that if no duration is explicitly specified, smartIflow assumes a duration of one order of magnitude of time. Therefore, this duration is used as a baseline and implicitly assigned to all processes, that are executed in milliseconds. An example for such a process would be the speed in which a relay closes its contacts.

The description found in the handbook describes, that it takes 60 seconds to pressurize the tank. To represent this behavior, the next OM of time is defined as one minute. This means, the pump has to pump for two OM of time until the tank is considered full. If the pressure tank still receives pressure after being fully pressured, it becomes damaged. A logical conclusion would be therefore to change the status of the pressure tank to damaged after another OM of time. To prevent this, the pressure switch opens its contacts and therefore de-energizes the upper circuit, we call control circuit. Since the pressure tank might be subject to a component failure and therefore might fail to open its contacts after the tank is pressurized, the system designer included the timer relay as a backup system. The handbook describes how the timer relay opens its contacts after being continuously powered for 60 seconds. That would be the same time, at which the pressure tank is considered full. Assuming the timer relay becomes active at the same time as the tank is considered full, it would not allow the pressure switch to serve it's intended purpose because the timer relay would always pre-empt it. Therefore, the time assigned to the timer relay is one OM of time higher, than the pressure tank needs to be considered full (e.g. one minute and two seconds). The timer relay therefore opens its contacts after receiving power for three OM of time. As the timer relay must have the chance to prevent the pressure tank from rupturing, the pressure tank is considered damage after receiving pressure for four OM of time (e.g. one minute and ten seconds).

4.3 Derived Safety Requirements

The TOP Event of the fault tree presented in the Fault Tree Handbook is labeled with "rupture of pressure tank after the start of pumping" [2]. Based on this TOP event, the safety requirement "The pressure tank must not be damaged after pumping hast started" can be derived. To model the temporal dependency in the description of the top event, a CTL formula has to be defined, that describes that the status of the tank is not allowed to become damage at any future path if the pump started pumping. In addition to that, a safety requirement is described, according to which the pressure tank must not be damaged in general, i.e. independently of pumping.

```
TankNotDamagedAfterPumpStarted :=
    AG((pump.isPowered == True
    -> tank.status != Damaged));

TankNotDamaged :=
    AG(tank.status != Damaged);
```

Listing 5: CTL formulae describing, that the tank cannot become damaged after the pump started pumping and that the tank cannot become damaged at all.

However, this additional requirement should already be fulfilled by the previous requirement, since damage to the tank is caused by excess pressure produced by the pump, unless there is a failure of the tank itself or if a tank has been installed that is not suitable for the use case. For both requirements, a severity rating of 10 on a scale from 1 to 10 is used. The reason for this rating is, that a rupture of the tank might lead to injuries of potential bystanders. A rupture further means the tank becomes incapable of serving its original purpose and therefore has to be repaired or replaced. For an evaluation of an actual system with multiple safety requirements, the different levels of severity rating should be defined, such as in [20]. As the system at hand is subject to two safety conditions of similar importance, this is not considered necessary here.

4.4 Fault Modeling

The fault modeling for the evaluation is based on the descriptions found in the Fault Tree Handbook [2]. Those descriptions come with some limitations concerning the modeling of faults:

- Neglection of all plumbing and wiring failures.
- Neglection of all secondary failures (failures of components in environments they are not qualified for) except "tank rupture after the start of pumping".
- "[...] [L]imit of the resolution at the "component failure level"", meaning that primary failures (failures of components in environments they are qualified for) are represented in the form of a circle in the diagram. It is assumed that those are not traced further.

To depict the failure mode of a component in this project, a state variable named "failureMode" is created. Those are of type enum with either "Nominal" or "Defect" as a value to indicate whether the component is damaged. To simulate failures, an event (of type failure) in the Event section and an event handler in the EventHandlers section are defined, to change the previously defined state variable to the faulty state. Previously defined behavior sections have to be updated, in order to consider the possibly faulty behavior based on the state of the failure mode state variable.
Since the system descriptions do not include information about failure probabilities, detection probabilities and the severity of failures, those values are chosen arbitrarily, as defined in Table 1.

Pressure switch fault A possible fault concerning the pressure switch is, that it does not detect the pressure of the pressure tank and therefore fails to break the connection of the control circuit. This fault can be modeled, by adding a state variable to the pressure switch, indicating whether the pressure switch is in a nominal or defect state. By defining an event of type failure (named "Defect" in the example), the state variable can be switched into defect mode through a corresponding entry in the EventHandlers section. Based on the state of this state variable, the behavior of the pressure switch is augmented, to only react to the received pressure signal when the state variable indicates nominal behavior.

K2 relay fault Another fault describes a component fault in the K2 relay in which it closes its contacts for more than 60 seconds. To model this fault, the same steps as for the modeling of the pressure switch fault are repeated. Therefore, a state variable is created indicating whether the component is defect. As for the pressure switch, this variable can be updated using an event and an associated EventHandler. The simulation of the fault can be modeled by augmenting the when-statement in the transition section managing the

```
class ElectricalComponent {
    Ports:
        Electrical powerConnection1;
        Electrical powerConnection2;

    Variables:
        // Whether the component is powered.
        Boolean isPowered = False;

    Transitions:
        // Update state variable, if power is received.
        when (flow.om(powerConnection1, powerConnection2) != null){
            isPowered = True;
        }
        // Vice versa.
        when (flow.om(powerConnection1, powerConnection2) == null){
            isPowered = False;
        }
}
```

Listing 2: The ElectricalComponent is an generalization of a electrical component. It therefore contains two electrical ports to connect it to other components, a state variable indicating whether the component is powered and a when-statement in the transition section to update the state variable based on the received electrical flow.

```
class ForwardingElectricalComponent extends ElectricalComponent{
     Behavior:
        // Always forwards power.
        connect { r = 1 } (powerConnection1, powerConnection2);
}
```

Listing 3: The ForwardingElectricalComponent is another generalization derived from the ElectricalComponent and describes how electrical components forward electricity.

```
class Motor extends ForwardingElectricalComponent{
    Ports:
        Mechanical mechanicalOutput;

    Behavior:
        // The motor produces mechanical power if it is powered.
        if(isPowered == True) {
            set(mechanicalOutput, { powered = True });
        }
        // Vice versa.
        if(isPowered == False) {
            set(mechanicalOutput, { powered = False });
        }
}
```

Listing 4: The description of the motor inherits its behavior with respect to its nature as an electrical component from the ForwardingElectricalComponent (3) class, which in turn inherits from ElectricalComponent (2).

Failure	Failure probability	Corresponding occurence level	Detection probability
Pressure switch	$1e-4$	1	3
Relay	$3e-5$	3	4
Switch	$3e-5$	2	5
Timer relay	$1e-4$	2	2
Pressure tank (rupture)	$5e-6$	4	5
Pressure tank (secondary)	$5e-6$	4	5

Table 1: Arbitrarily chosen probabilities for the occurrence and detection of failures.

disconnect of the relay by an additional condition. This condition makes sure the relay does only open its contacts when it is in nominal mode and therefore does not open in the defect (jammed) mode.

S1 switch fault A fault concerning the S1 switch is continuous activation of the switch, meaning it constantly powers the upper control circuit. In conjunction with a failure of the pressure switch, this failure can lead to rupture of the tank. For this failure, once again model a state variable indicating the failure mode, a event of type failure and a corresponding EventHandler have to be defined. The continuous activation itself can then be modeled by checking whether the component is in nominal state before the previoulsy mentioned state variable is set to false.

Pressure tank A fault of the pressure tank itself mentioned in the Handbook is a rupture of the tank "due to improper selection of installation (wrong tank)" [2]. Another fault of the pressure tank is the only considered secondary failure "from other out-of-tolerance conditions (e.g. mechanical, thermal)" [2]. Both of those faults can be modeled, by adding new failure events ("TankRupture" and "SecondaryFailure") and EventHandlers, that switch the state of the tank to damaged if the event occurs.

4.5 Composing System and Fault Models

smartIflow allows the definition of failures directly as a part of the component descriptions, and injects the fault model automatically. From a user's perspective, for this reason, we do not need to worry about it.

4.6 Safety Analysis

The smartIflow workbench allows us to view the previously modeled system in the component view (see figure 7), in which the various components can be arranged freely. The generated state graph can be viewed in the state view (see figure 8), further the different component and system states can be inspected using the variable and ports watchlists (see 10, 9). The most interesting results in respect to safety analysis are the results of the generateable FMEA and FTA analysis (see figure 3, 4, 5). Note, that smartIflow does not generate the depicted fault trees directly, but allows the user to export the generated minimal cut sets in different formats, including the OPEN-PSA format. Using arbre analyste [21], a software tool for FTA that is compatible with the OPEN-PSA format, the generated fault trees can be visualized, freely arranged and exported into various image formats.

Generated FTA As suspected in 4.3, the fault trees (see 3, 4) do not differ due to the similarity of the safety requirements. By analyzing the fault trees it can be seen, that three different combinations of component failures and three individual failure events lead to a violation of the safety requirements, depicted as the TOP events of the fault trees. In the following descriptions, we call such combinations of two component failures "double failures". In addition to those three double failures, it can be observed, that a failure (e.g. contacts jammed) of the K2 relay can lead to the TOP event, as well as failures of the tank itself. It can be concluded, that a failure of the K2 relay and failures of the tank itself are the biggest safety risks. This is because, they are directly connected to the OR-gate under the TOP event and therefore lead to a violation of the safety requirement, even if they occur individually. A conclusion of the analysis is therefore that it makes sense to pay special attention to the selection and installation of the tank. Further, it makes sense to provide the function of the K2 relay redundantly. One way to do so, would be to use two relays in series connection instead of just one. A second relay would still be able to cut the connection to the motor if the contacts of the other relay fail to open. Another takeaway is, that the failure of the pressure switch is part of each observed double failure.

Generated FMEA The FMEA results displayed in figure 5 were generated using the CTL-based and state-based effect finders (see [22]). For the generation, a event trigger specification limiting the amount of faults per path to two is used. In the generated results it can be observed, how the relay and the tank both lead to end effect violating the safety requirements (CTL formulae). It can also be observed how a failure of the pressure switch and the K2 relay impact other components by their failure. The listed higher level effect values allow us to directly observe, how the failure of the K2 relay leads to a violation of the safety requirements. The RPN helps us identify faults, that pose a high risk if they occur. For instance, it can be observed, how a failure of the tank itself (e.g. due to installation of the wrong tank) is a major risk. Since FMEA analysis typically do not consider double failures, one relies on the FTA for a thorough analysis. It was previously no-

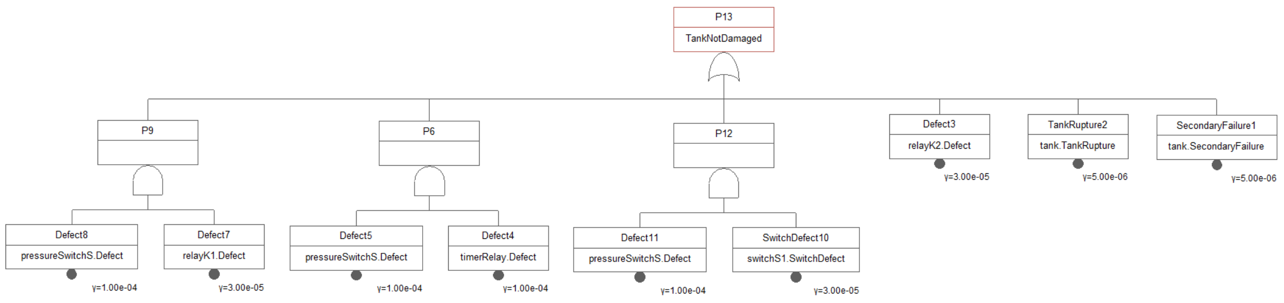


Figure 3: Fault Tree generated by smartIflow and visualized with arbre analyste for the previously described CTL specification, stating that the tank should not be damaged.

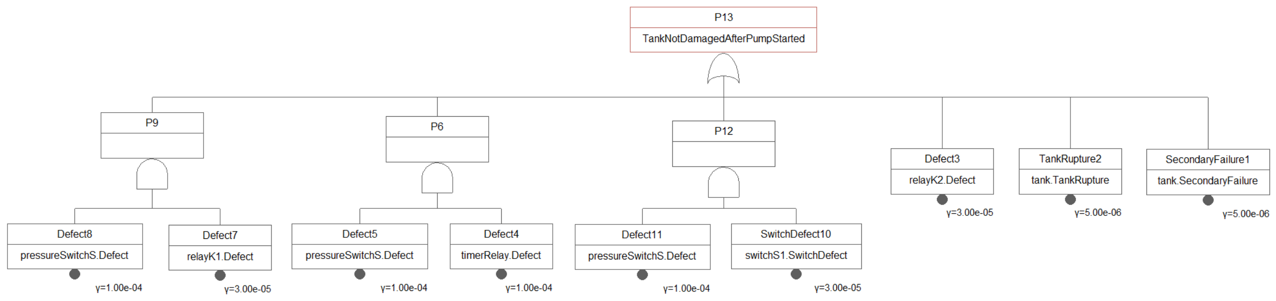
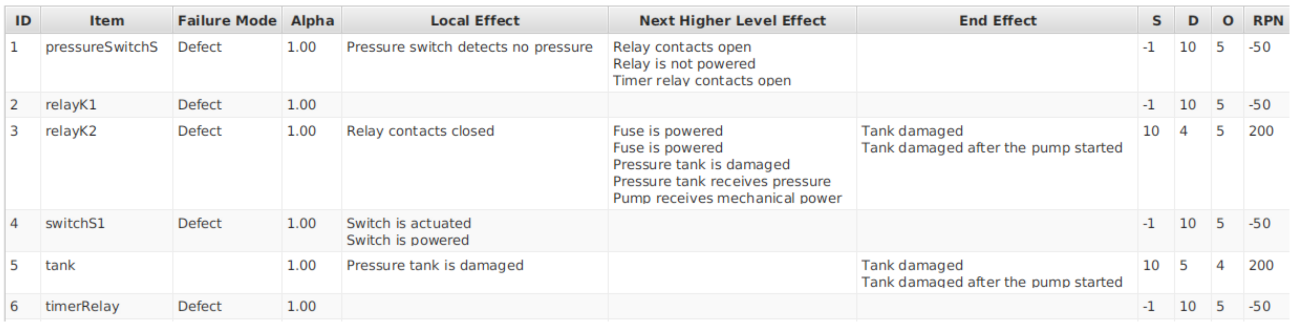
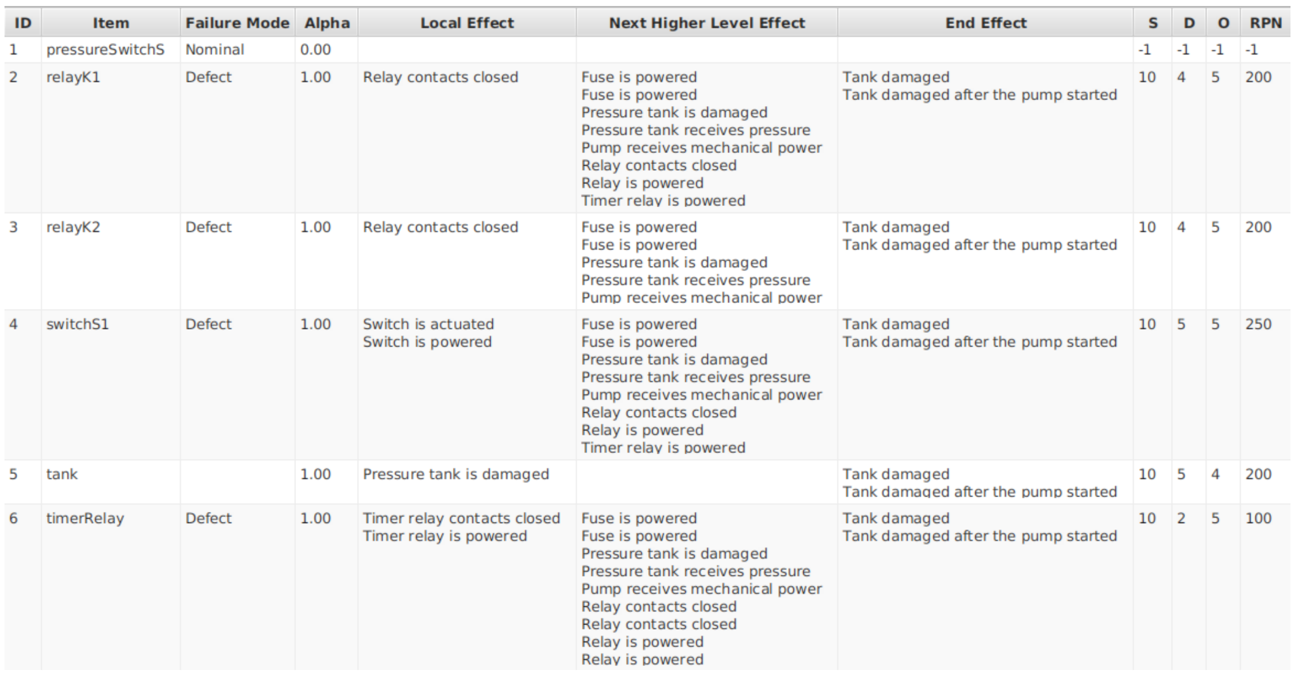


Figure 4: Fault Tree generated by smartIflow and visualized with arbre analyste for the previously described CTL specification, stating that the tank should not be damaged.



ID	Item	Failure Mode	Alpha	Local Effect	Next Higher Level Effect	End Effect	S	D	O	RPN
1	pressureSwitchS	Defect	1.00	Pressure switch detects no pressure	Relay contacts open Relay is not powered Timer relay contacts open		-1	10	5	-50
2	relayK1	Defect	1.00				-1	10	5	-50
3	relayK2	Defect	1.00	Relay contacts closed	Fuse is powered Fuse is powered Pressure tank is damaged Pressure tank receives pressure Pump receives mechanical power	Tank damaged Tank damaged after the pump started	10	4	5	200
4	switchS1	Defect	1.00	Switch is actuated Switch is powered			-1	10	5	-50
5	tank		1.00	Pressure tank is damaged		Tank damaged Tank damaged after the pump started	10	5	4	200
6	timerRelay	Defect	1.00				-1	10	5	-50

Figure 5: Screenshot of an FMEA generated by smartIflow.



ID	Item	Failure Mode	Alpha	Local Effect	Next Higher Level Effect	End Effect	S	D	O	RPN
1	pressureSwitchS	Nominal	0.00				-1	-1	-1	-1
2	relayK1	Defect	1.00	Relay contacts closed	Fuse is powered Fuse is powered Pressure tank is damaged Pressure tank receives pressure Pump receives mechanical power Relay contacts closed Relay is powered Timer relay is powered	Tank damaged Tank damaged after the pump started	10	4	5	200
3	relayK2	Defect	1.00	Relay contacts closed	Fuse is powered Fuse is powered Pressure tank is damaged Pressure tank receives pressure Pump receives mechanical power	Tank damaged Tank damaged after the pump started	10	4	5	200
4	switchS1	Defect	1.00	Switch is actuated Switch is powered	Fuse is powered Fuse is powered Pressure tank is damaged Pressure tank receives pressure Pump receives mechanical power Relay contacts closed Relay is powered Timer relay is powered	Tank damaged Tank damaged after the pump started	10	5	5	250
5	tank		1.00	Pressure tank is damaged		Tank damaged Tank damaged after the pump started	10	5	4	200
6	timerRelay	Defect	1.00	Timer relay contacts closed Timer relay is powered	Fuse is powered Fuse is powered Pressure tank is damaged Pressure tank receives pressure Pump receives mechanical power Relay contacts closed Relay contacts closed Relay is powered Relay is powered	Tank damaged Tank damaged after the pump started	10	2	5	100

Figure 6: Screeenshot of an FMEA generated by smartIflow with the pressure switch set to defect by default.

ticed, that all observed double failures include a failure of the pressure switch. This knowledge can be applied by setting the state variable of the pressure switch indicating its failure mode to defect, in order to gather some further insights. In Figure 6 it can be seen, how risky an outage of the different components is. It can be noticed, that besides the tank itself, a failure of the switch is a big risk with regard to the rupture of the tank. By looking at the individual values, it can be figured out that this is due to high detection- and occurrence levels.

5 Results

This chapter is about the discussion of the generated safety analysis results and includes a description, how the expansion of the state space affects different analysis tasks.

5.1 FTA in smartIflow

Using the smartIflow language and workbench, it is possible to generate the previously described fault trees for the safety requirements of the pressure tank system. Through analyzing the generated fault tree, it is possible to detect different fault-scenarios leading to the violation of the safety requirements. In order to read the fault tree, the reader must be familiar with the AND and OR symbols, but otherwise the presentation is minimalistic and easy to understand. The identification of events that are easily responsible for a system failure is quite intuitive, since the reader only needs to look for nodes that have the shortest paths to the TOP event. In the previous chapter, it was possible to put previously generated knowledge to use and to come up with some recommendations that would make the system more secure.

Compared to the fault tree featured in the Fault Tree Handbook, the generated fault tree is wider but less deep. This is due to the fact, that the Handbook does not contain the same failure event multiple times, but rather combines them using OR gates. Which version is preferred depends on the viewer. The flatter version might be more accessible, especially for less experienced users because of its less complex structuring. Besides the structure, the Fault Tree in the Handbook contains describing blocks and other elements which are not part of the generated FTA. An extension of the generated fault tree with such additional descriptions through an expert would be possible by exporting the minimal cut sets into one of the standardized formats and importing it into a suitable third party tool. Overall, the fault tree contains the previously modeled faults and, accordingly, the faults which are represented by diamonds in the manual which do not represent secondary faults. Therefore, smartIflow is a suitable solution for the automatic generation of fault trees of the presented pressure tank system.

5.2 FMEA in smartIflow

As for the FTA, it is also possible to generate the desired FMEA results. A disadvantage of the FMEA is that the double faults are not directly apparent from it, since they are not included on the immediate results described earlier. In combination with the FTA results, it is possible to identify the double failures and gain additional insights by setting a failure mode to defect by default. These results are of particular importance, as they allow the Identification of components that can

Allowed faults	Amount of states	Transition system	Time FMEA	Time FTA 1	Time FTA 2
2	651	$0.14s$	$< 1s$	$< 1s$	$< 1s$
4	1889	$0.39s$	$< 1s$	$< 1s$	$< 1s$
6	2087	$0.45s$	$167s$	$50s$	$136s$
8	2087	$0.44s$	$-$	$12m$	$125m$
10	2087	$0.44s$	$-$	$-$	$-$
12	2087	$0.48s$	$-$	$-$	$-$

Table 2: This table features time measurements for different safety analysis tasks for different amount of considered faults. Trials which took longer than 40 Minutes or failed due to a crash of the program are depicted by $-$. For the generation of the FTA, the event-based minimal cutset generator is used. FTA 1 is generated for the CTL stating, that the tank is not allowed to be damaged. FTA 2 is generated using the CTL describing, that the tank cannot be damaged after the start of the pumping.

cause the most damage in the event of a failure. In general, the obtained results correspond to the results expected from a Failure Mode and Effect Analysis.

5.3 State Space Management in smartIflow

To assess whether the means of smartIflow are sufficient to avoid an explosion of the state space, it makes sense to observe the performance of the smartIflow workbench for different safety analysis tasks. Therefore, measurements were taken (see table 2) for different safety analysis tasks using an Intel Core i7-9700k CPU at 3.70 GHz (base clock speed) and 16 GB RAM on a Windows 10 machine. Those measurements are taken using event trigger specifications, used to limit the unfolding of paths to different amounts of failure events. It can be observed, that the state tree for the system is unfolded in reasonable time, even if the amount of considered failure events is not limited. For the generation of FMEA and FTA results, it can be observed, how complexity increases with increasing state space. It can be observed that the calculation of analyses with 8 or more allowed faults per path becomes very complex. In the attempts to generate such analyses with 8 or more failures, the CPU load increases strongly and the workbench sometimes crashed. For the pressure tank system, this limitation is less problematic, since the system under consideration contains only six faults. When it is required to detect complex failure constellations, crashes of the workbench could become problematic, as one might accept an increased runtime in order to detect all possible fault combinations leading to the violation of the safety requirements. Besides that, that especially for systems which are of similar size as the case example used here, the event trigger specifications are a useful mean, that allows the generation of the desired analysis with reasonable time and effort.

6 Conclusion

In this paper, the smartIflow language and associated workbench were evaluated. It was demonstrated, how the smartIflow language and tools can be used to model and analyze system designs, such as the pressure tank case example. For the practical modeling part, it was possible to leverage the object-oriented nature of smartIflow and therefore managed to keep model descriptions minimal and concise. This was achieved by introducing generalization classes, from which other classes can inherit.
When it comes to analyzing modeled system designs, it was shown, that smartIflow allows the generation of FTA and FMEA results automatically based on the one-time modeled system and the descriptions of the safety requirements. It was observed how the amount of required computational resources grow rapidly, as the state space grows. By utilization of the smartIflow event trigger specifications, this problem can be mitigated by limiting the length to which a path is traced to a fixed number of fault conditions. It can be concluded, that this might become a problem if the identification of complex fault constellations is necessary but otherwise constitutes a suitable solution.
In summary, it is possible to learn the smartIflow language and the functionality of the associated workbench in reasonable time. For the preparation of the presented analysis, it was necessary to contact the developers, or researchers to clarify some questions. With a more extensive documentation and more examples, smartIflow will be a powerful tool for the rigorous performance of safety analysis tasks.

7 Declaration of independent work

I hereby declare that this seminar report is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

References

[1] Philipp Hönig, Rüdiger Lunde, and Florian Holzapfel. Model based safety analysis with smartiflow. *Information*, 8(1), 2017.

[2] D F Haasl, N H Roberts, W E Vesely, and F F Goldberg. Fault tree handbook. pages VIII–1 pp., 1 1981.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT press, 2008.

[4] Mohamed Ben-Daya, Salih Duffuaa, A Raouf, Jezdimir Knezevic, and Daoud Ait-Kadi. *Handbook of Maintenance Management and Engineering.* Springer London, 01 2009.

[5] Kapil Dev Sharma and Shobhit Srivastava. Failure mode and effect analysis (fmea) implementation: a literature review. *J Adv Res Aeronaut Space Sci*, 5:1–17, 2018.

[6] Michael Stamatelatos, William Vesely, Joanne Dugan, Joseph Fragola, Joseph Minarick, and Jan Railsback. Fault tree handbook with aerospace applications. 2002.

[7] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15-16:29–62, 2015.

[8] Orna Kupferman. *Automata Theory and Model Checking*, page 107. Springer International Publishing, Cham, 2018.

[9] Valentin Goranko and Antje Rumberg. Temporal Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, Summer 2022 edition, 2022.

[10] A. Joshi, S.P. Miller, M. Whalen, and M.P.E. Heimdahl. A proposal for model-based safety analysis. In *24th Digital Avionics Systems Conference*, volume 2, pages 13 pp. Vol. 2–, 2005.

[11] Armin Biere. Tutorial on model checking: Modelling and verification in computer science. In *International Conference on Algebraic Biology*, pages 16–21. Springer, 2008.

[12] Faranak Nejati, Abdul Azim Abdul Ghani, Keng-Yap Ng, and Azmi Jaafar. Handling state space explosion in verification of component-based systems: A review. *CoRR*, abs/1709.10379, 2017.

[13] Yiannis Papadopoulos, Martin Walker, David Parker, Erich Rüde, Rainer Hamann, Andreas Uhlig, Uwe Grätz, and Rune Lien. Engineering failure analysis and design optimisation with hip-hops. *Engineering Failure Analysis*, 18(2):590–608, 2011.

[14] Oleg Lisagor. *Failure logic modelling: a pragmatic approach.* PhD thesis, University of York, 2010.

[15] Neal Andrew Snooke and Mark H Lee. Qualitative order of magnitude energy-flow-based failure modes and effects analysis. *Journal of Artificial Intelligence Research*, 46:413–447, 2013.

[16] Philipp Hönig and Rüdiger Lunde. A new modeling approach for automated safety analysis based on information flows. In *Proceedings of DX'14, 2014*, 01 2014.

[17] Christian Müller, Philipp Hönig, and Rüdiger Lunde. Evaluation of smartiflow based on the wheel brake system from arp4761. *IFAC-PapersOnLine*, 51(24):1255–1262, 2018. 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2018.

[18] Philipp Hönig, Rüdiger Lunde, and Florian Holzapfel. Formal verification of technical systems using smartiflow and ctl. In *Proceedings of the 2nd International Conference on Applications in Information Technology (ICAIT-2016), 2016*, 01 2016.

[19] Thomas David and Hunt Andrew. *The Pragmatic Programmer.* Addison Wesley, 2019.

[20] *MIL-STD-1629A, MILITARY STANDARD: PROCEDURES FOR PERFORMING A FAILURE MODE, EFFECTS, AND CRITICALITY ANALYSIS.*

[21] Emmanuel Clement, A Rauzy, and Thierry Thomas. Arbre analyste: un outil d'arbres de défaillances respectant le standard open-psa et utilisant le moteur xfta. *Congrès Lambda Mu 19 de Maîtrise des Risques et Sûreté de Fonctionnement, Dijon, 21-23 Octobre 2014*, 2014.

[22] Christian Müller, Rüdiger Lunde, and Philipp Hönig. Generation of a failure mode and effects analysis with smartiflow. In *Proceedings of the 30th European Safety and Reliability Conference (ESREL2020), Venice, Italy, 2020*, pages 1662–1669, 01 2020.

A Appendix

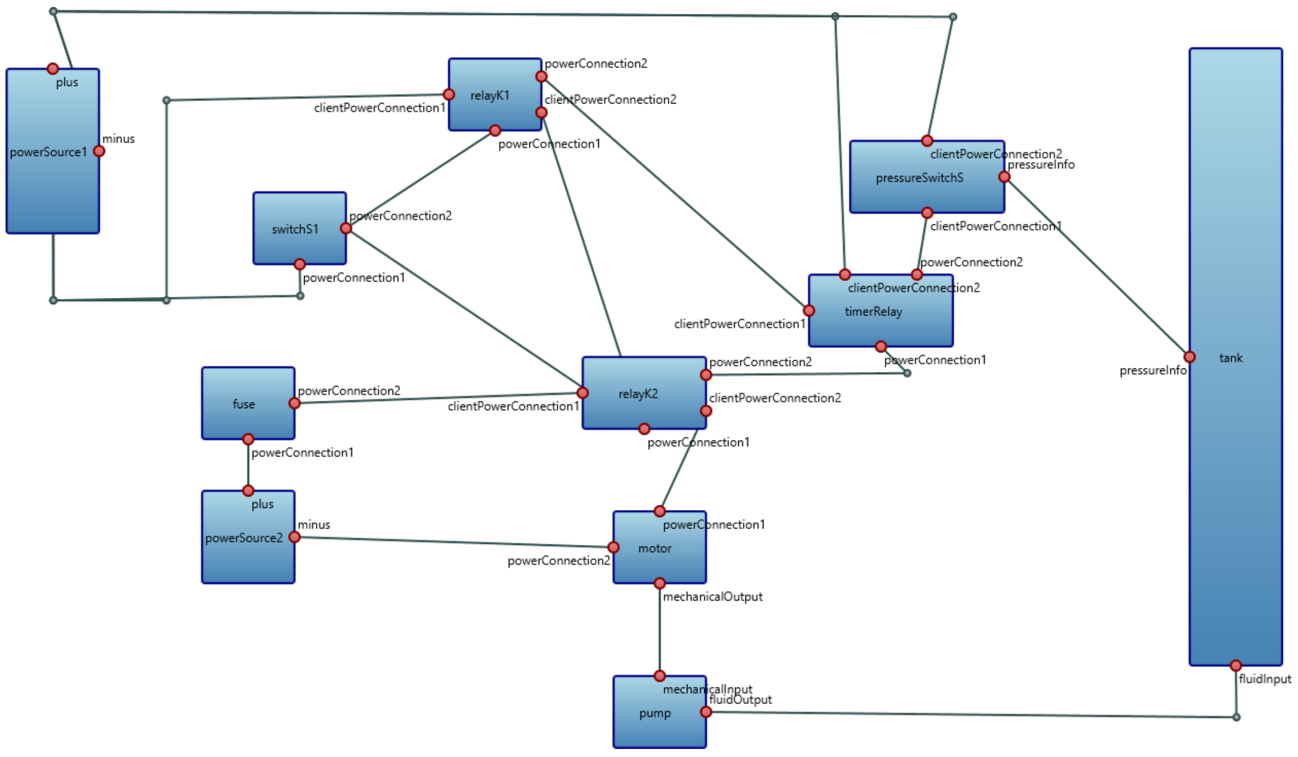


Figure 7: Screenshot of the component view of the Pressure Tank System modeled in smartIflow.

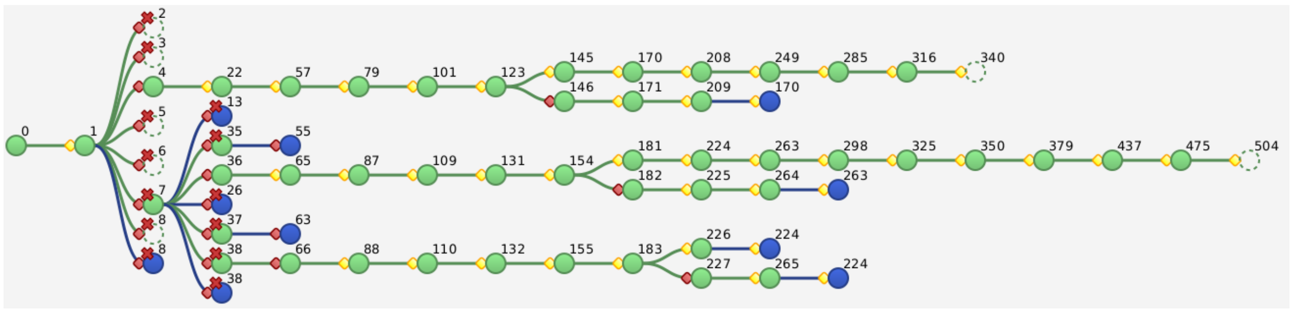
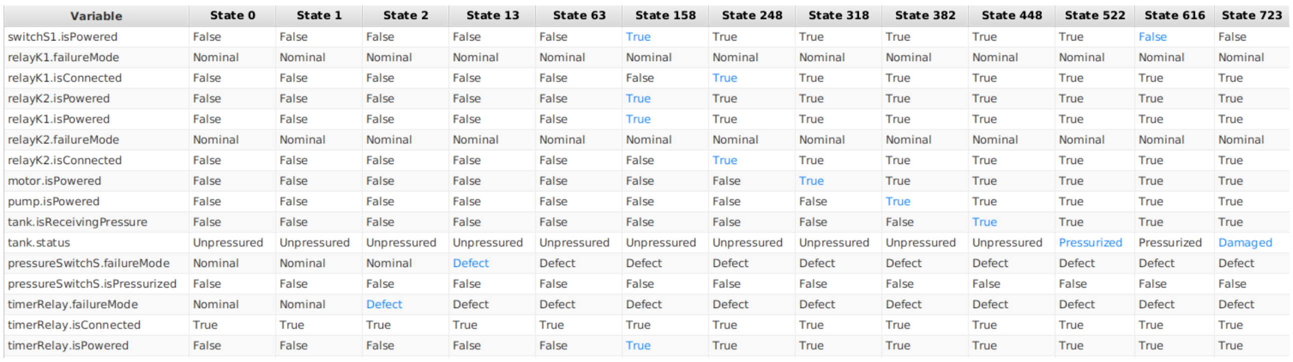
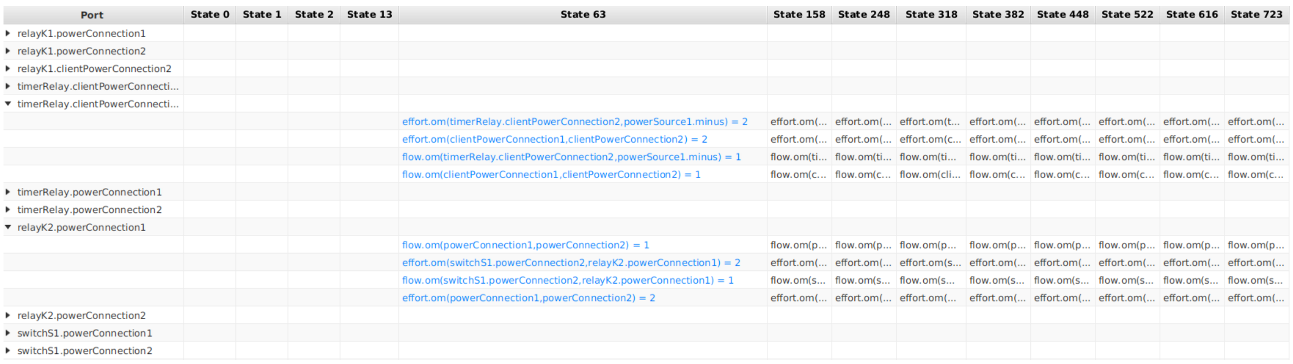


Figure 8: Screenshot of the state view displaying the different system state. By clickin on different nodes, the user can inspect the variables and ports of the select state and prior states.



Variable	State 0	State 1	State 2	State 13	State 63	State 158	State 248	State 318	State 382	State 448	State 522	State 616	State 723
switchS1.isPowered	False	False	False	False	False	True	True	True	True	True	True	False	False
relayK1.failureMode	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal
relayK1.isConnected	False	False	False	False	False	False	True	True	True	True	True	True	True
relayK2.isPowered	False	False	False	False	False	True	True	True	True	True	True	True	True
relayK1.isPowered	False	False	False	False	False	True	True	True	True	True	True	True	True
relayK2.failureMode	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal
relayK2.isConnected	False	False	False	False	False	False	True	True	True	True	True	True	True
motor.isPowered	False	False	False	False	False	False	False	True	True	True	True	True	True
pump.isPowered	False	False	False	False	False	False	False	False	True	True	True	True	True
tank.isReceivingPressure	False	False	False	False	False	False	False	False	False	True	True	True	True
tank.status	Unpressured	Unpressured	Unpressured	Unpressured	Unpressured	Unpressured	Unpressured	Unpressured	Unpressured	Unpressured	Pressurized	Pressurized	Damaged
pressureSwitchS.failureMode	Nominal	Nominal	Nominal	Defect	Defect	Defect	Defect	Defect	Defect	Defect	Defect	Defect	Defect
pressureSwitchS.isPressurized	False	False	False	False	False	False	False	False	False	False	False	False	False
timerRelay.failureMode	Nominal	Nominal	Defect	Defect	Defect	Defect	Defect	Defect	Defect	Defect	Defect	Defect	Defect
timerRelay.isConnected	True	True	True	True	True	True	True	True	True	True	True	True	True
timerRelay.isPowered	False	False	False	False	False	True	True	True	True	True	True	True	True

Figure 9: The variable watchlist in smartIflow for a path in which both, the pressure switch and the timer relay are defect.



Port	State 0	State 1	State 2	State 13	State 63	State 158	State 248	State 318	State 382	State 448	State 522	State 616	State 723
▸ relayK1.powerConnection1													
▸ relayK1.powerConnection2													
▸ relayK1.clientPowerConnection2													
▸ timerRelay.clientPowerConnecti...													
▾ timerRelay.clientPowerConnecti...													
					effort.om(timerRelay.clientPowerConnection2,powerSource1.minus) = 2	effort.om(...	effort.om(...	effort.om(t...	effort.om(...	effort.om(...	effort.om(...	effort.om(...	effort.om(...
					effort.om(clientPowerConnection1,clientPowerConnection2) = 2	effort.om(...	effort.om(...	effort.om(c...	effort.om(...	effort.om(...	effort.om(...	effort.om(...	effort.om(...
					flow.om(timerRelay.clientPowerConnection2,powerSource1.minus) = 1	flow.om(ti...	flow.om(ti...	flow.om(ti...	flow.om(ti...	flow.om(ti...	flow.om(ti...	flow.om(ti...	flow.om(ti...
					flow.om(clientPowerConnection1,clientPowerConnection2) = 1	flow.om(c...	flow.om(c...	flow.om(cli...	flow.om(c...	flow.om(c...	flow.om(c...	flow.om(c...	flow.om(c...
▸ timerRelay.powerConnection1													
▸ timerRelay.powerConnection2													
▾ relayK2.powerConnection1													
					flow.om(powerConnection1,powerConnection2) = 1	flow.om(p...	flow.om(p...	flow.om(p...	flow.om(p...	flow.om(p...	flow.om(p...	flow.om(p...	flow.om(p...
					effort.om(switchS1.powerConnection2,relayK2.powerConnection1) = 2	effort.om(...	effort.om(...	effort.om(s...	effort.om(...	effort.om(...	effort.om(...	effort.om(...	effort.om(...
					flow.om(switchS1.powerConnection2,relayK2.powerConnection1) = 1	flow.om(s...	flow.om(s...	flow.om(s...	flow.om(s...	flow.om(s...	flow.om(s...	flow.om(s...	flow.om(s...
					effort.om(powerConnection1,powerConnection2) = 2	effort.om(...	effort.om(...	effort.om(...	effort.om(...	effort.om(...	effort.om(...	effort.om(...	effort.om(...
▸ relayK2.powerConnection2													
▸ switchS1.powerConnection1													
▸ switchS1.powerConnection2													

Figure 10: The ports watchlist in smartIflow for a path in which both, the pressure switch and the timer relay are defect.