Statistical Analysis of Continuous Degradation Processes in Fault Trees

David Stritzl
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
d.l.stritzl@student.utwente.nl

ABSTRACT

Over the last few decades, risk analysis of maintainable systems has played an increasingly important role in the industry. One of the most common methods used for risk analysis is fault tree analysis. In standard fault trees, a component can be either failed or not. However, in reality, the degradation of a component is often a continuous process, which currently cannot be modeled in standard fault trees. The goal of this is paper is to introduce an extension to standard fault trees that will support continuous degradation processes and strategies for analysing the extended model using UPPAAL.

Keywords

Fault Trees, Continuous Degradation, Statistical Analysis, UPPAAL

1. INTRODUCTION

Over the last few decades, the number of maintained automated systems has seen a steady growth. This has caused the reliability analysis of such systems to play an increasingly important role in the industry in order to achieve higher reliability at lower costs.

One of the most common methods used for reliability analysis is fault tree analysis (FTA)[6]. Fault trees (FTs) are used to model dependency-relations of components inside a system and can be used for identifying the possible modes of failure within a system. In FTs, the failure of individual components is represented by probability-based basic events (BEs). BEs can be combined into compound events through various gates (i.e. AND, OR, etc.) to model the failing of a larger subsystem and therefore the propagation of failures in a system.

In standard fault trees (SFTs), a basic event can only have two states: either working or failed. In reality, however, the degradation of (physical) components of a system is a continuous process[2] that can (at least partially) be reversed by repairing the component before it fails. The use of continuous degradation processes in FTs allows for more accurate models and therefore possibly smarter maintenance strategies. Since SFTs and the corresponding analysis tools have no support for continuous degradation processes by default, it is necessary to create an extension to SFTs that will allow this. There has already been work in this direction [4], which provides an DFT extension that supports multiple stages of degradation. This goal of this paper is to provide an extension to SFTs supporting completely continuous degradable components. Furthermore, the paper will answer the following subquestions:

RQ1 What is the state of the art in modelling fault trees?

RQ2 What is a suitable modeling tool for fault tree and its proposed extension?

RQ3 How can continuous degradation functions be modelled?

RQ4 How can continuous degradation functions be integrated into fault trees?

RQ5 How to validate the proposed model?

RQ6 What is a suitable method for getting reliability information from the model?

RQ7 What is a suitable method for getting information on maintenance intervals?

First, section 2 gives an overview of background information and related work. Section 3 describes the method for integrating continuous degradation function in fault trees and the validation the proposed model. Section 5 discusses the results of the paper. Lastly, section 6 contains the conclusion of the paper.

2. BACKGROUND & RELATED WORK

2.1 Fault Trees

Fault trees are a method of modelling the propagation of component failures inside systems as a tree. BEs, the leaves of the tree, represent the possible failures of components. In standard fault trees, events can have two states: active and failed. The transition between these states is often modelled with a failure rate that often grows exponentially over time.

The propagation of events within a system is described with gates. Gates can have multiple event-inputs from which new, compound events can be generated. Standard fault trees support four kinds of gates:

24th Twente Student Conference on IT January 22nd, 2016, Enschede, The Netherlands.

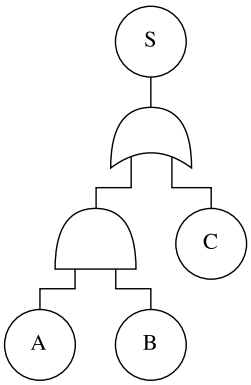

Figure 1: Example of a fault tree

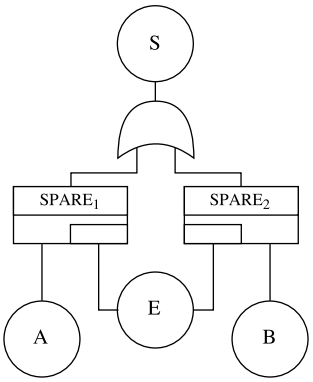

Figure 2: Example of a dynamic fault tree

AND This gate has multiple inputs and only triggers an event when all of them fail.

OR This gate also has multiple inputs and triggers an event when at least one of them fails.

k/N This gate has N inputs and triggers an event when a least k of them fail.

INHIBIT This gate has exactly two inputs and functions like an AND-gate with two inputs. It is only used as a semantic nuance to the AND-gate.

An example of a FT is given in fig. 1. The system S consists of three fail-able components, described by the BEs A, B and C. A and B are connected to an AND-gate, which will only generate an event when both, A and B, have failed. The AND-gate and C are connected by an OR-gate, which will generate an event if either one of the subsystems fail. Since the system S is directly connected to the OR-gate, S will fail simultaneously with the OR-gate.

2.2 Degradation Processes

The degradation of a material is the change (to the worse) of its internal structure and therefore the change of its properties, i.e. strength, due to external influences over time. This can be caused by many different (external) factors, for instance, the stress [5] that is applied, the temperate that it is subjected to or corrosion. The effect of degradation on materials can appear in many different forms, most prominent being loss of strength, deformation and possibly failure, which can lead to a loss of performance of a component and higher maintenance costs.

For objects made of simple materials (i.e. pure metals) experimental data is often readily available, although for more complex components like compound materials (i.e. alloys) or objects made of multiple materials use case specific material experiments have to be carried out. Since this degradation process is influenced by too many variables to model, the overall degradation process of a component is often simplified using failure probability over time [2].

2.3 Dynamic Fault Trees

For more complex models, it can be difficult or even impossible to implement them using SFTs, since they do not allow for more complex dynamic behaviour. Dynamic fault trees[1] are an extension to the standard fault trees, integrating more dynamic behaviour like spare components or dependent components.

DFTs support a variety of additional gates:

PAND This gate has multiple inputs and only triggers an event when all input fail from left to right.

FDEP This gate has a trigger input and multiple dependency inputs. It does not trigger any event, but it will cause all dependent children to fail when the trigger input fails.

SPARE This gate has one primary input and multiple spare inputs. When a primary component fails it will claim one of the spare components. It will only trigger an CE if no other spares are available.

Furthermore, DFTs add another event state: inactive, which allows components to have different failure rates while inactive.

An example of a DFT is given in fig. 2. The system S consists of the components A, B and the spare component E, which is inactive at first. When either A or B fails, they can claim component E, which is then activated and made unavailable to other parts of the system, through the SPARE-gates $SPARE_1$ and $SPARE_2$ respectively. $SPARE_1$ or $SPARE_2$, and therefore system S, will only fail if A or B, fail and spare component E has already been claimed by one of the SPARE-gates.

2.4 Statistical Analysis of FTs

There are many different methods and tools for the statistical analysis of FTs. One of the most frequently used tools for verification and statistical analysis of time-based models is Uppaal-SMC[3]. Systems in Uppaal-SMC are described with a combination of Uppaal templates. A template describes a single process, consisting of a stochastic timed automaton and possible additional code to keep track of more complex state information and to synchronise with other templates in the system. Templates can be give instantiation parameters, ie. color switching intervals for traffic lights, in order to make templates more generic.

A stochastic timed automaton (STA) is a time-based state machine. In a STA, states can have (exponential) rates for taking transitions and invariants for staying in the state. Transitions can have guards to enable the transition based on state. Furthermore, transitions can be synchronised between STA in Uppaal using system-wide channels. Special branch points exist, that can choose the next transition based on a (linear) probability.

An example of a STA is provided in fig. 3. The first transition from A to B has a constraint that is has to be taken when $c < 3$, where c is a timer. The transitions from B to C_1 and C_2 have the probabilities $\frac{1}{4}$ and $\frac{3}{4}$, respectively, to be taken. Next, the transitions from C_1 and C_2 to END

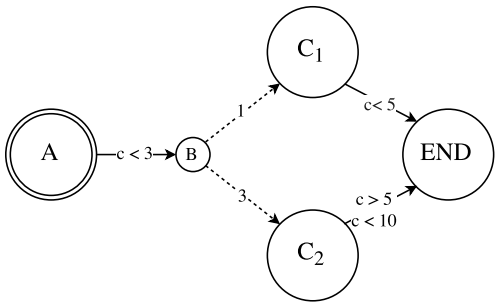

Figure 3: Example of a stochastic timed automaton

have then time-constraints $c < 5$ *and* $5 < c < 10$.

UPPAAL-SMC is a complete environment for specifying and analysing systems using STA. For the declaration of systems, it provides a visual editor for STAs and a code editor for the template code. UPPAAL can be used to compute the qualitative properties of a model using simulations. For instance, the average frequency of reaching a certain state in the system. Also, it can be used to compute the qualitative properties of a model, for example, the reachability of a state in the model.

Since FTs are not directly supported in UPPAAL-SMC, they first have to be converted to STA. This process will be elaborated later on in the paper.

2.5 Fault Maintenance Trees

Fault maintenance trees (FMTs) [4] are an extension to dynamic fault trees. They support the possibility of replacing failed components and they add maintainable BEs (MBEs) which use multiple phases to describe the current degradation state: functioning, lightly degraded and undetectable, lightly degraded and detectable, and extremely degraded and failed. MBEs have different probabilities for each transition from one degradation phase to the next.

Furthermore, FMTs allow for the addition of inspections, during which, failed or damaged components can be replaced or repaired. A visualisation of the different phases of a MBE modeled as a STA can be seen in fig. 4. Inspections can happen on a periodic interval or when specific MBEs reach their inspection threshold phase which represents the first detectable signs of the degradation of the component and will also cause an inspection to happen. Also, it the concept of repair units is added, where a group of multiple components can be repaired at the same time once a specific MBE fails, even if these components have not yet failed. Lastly, it adds an additional RDEP-gate, which can be used to describe a relation between the phase of degradation of one component and the failure-rate of another.

Although FMTs can describe degradation processes with multiple phases instead of the default two, working and failed, they do not fully support continuous degradation processes, which restricts models to phase-type failure rate distributions like the exponential distribution. Moreover, since maintenance levels can only be set on a phase transition, this limits the possible number of maintenance strategies for a model.

3. METHOD

3.1 Modelling of Degradation Processes

Since degradation processes of components depend on a large amount of external factors, they are too difficult to

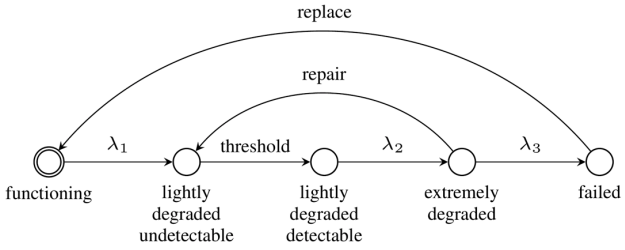

Figure 4: Maintainable BE modelled as a STA

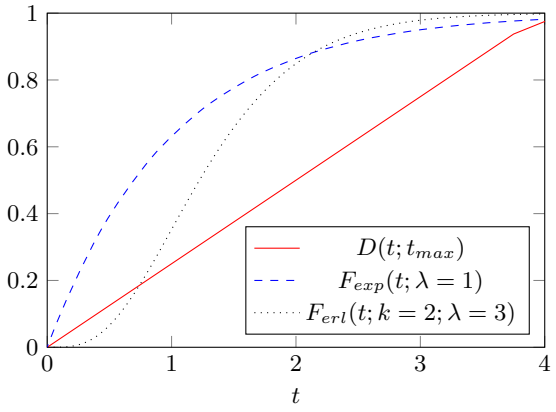

Figure 5: Example of a simplified degradation process

model accurately while avoiding complexity. Therefore, for the simplicity of this paper, all degradation processes will be modelled using linear degradations.

For the failure probability of a component, various cumulative distributions can be used. This paper will deal with the exponential distribution, since this is often used for existing discrete FTs. Also, the Erlang distribution will be covered, since it essentially combines multiple consecutive exponential distributions and can be used for describing the fail rate of FTs with multiple degradation phases, as in [4]:

$$F_{exp}(t;\lambda) = 1 - e^{\lambda D(t)}$$

$$F_{erl}(t;k;\lambda) = 1 - \sum_{n=0}^{k-1} \frac{1}{n!} e^{-\lambda D(t)} (\lambda D(t)^n)$$

Other distributions are possible as well, but are not covered in this paper.

Examples of these different distributions in degradations processes can be found in fig. 5. $D(t;t_{max})$ *is the degradation of the component over time.* $F_{exp}(t;\lambda)$ *and* $F_{erl}(t;k;\lambda)$ *show possible failure probabilities of the component over time.*

3.2 Continuous Degradation Processes in FTs

To support components with continuous degradation, the degradation function has to be integrated into the BEs (RQ4).

As a first step, the most straightforward solution would be to compute a failure probability f_p for the current time since the replacement of a component t_u, using the distributions from section 3.1. Then, generate a uniform distributed random value r, where $r \in (0.0, 1.0]$, and compare it to the $r < f_p$. Since this calculation would have to be

done for every time unit of uptime of a component, it has a quite large computation overhead. Moreover, this would also lead to UPPAAL not being able to optimize the model as it could for other models, since every time unit is now a separate state.

Alternatively, this computation can be reversed by starting with a randomly generated time-to-fail t_f for each BE. For the calculation of the time-to-fail, the random functions corresponding to the components degradation distribution should be used. Provided are the random functions for the exponential and Erlang distributions.

Given a uniform random function $R_{uni}()$ that generates numbers from $U \in (0, 1]$, the exponential distributed random function can be defined as:

$$R_{exp}(\lambda) = -\frac{1}{\lambda} \ln 1 - R_{uni}()$$

And similarly the Erlang distributed random function:

$$R_{erl}(k; \lambda) = -\frac{1}{\lambda} \ln \prod_{i=1}^{k} R_{uni}()$$

Once the time-to-fail t_f has been calculated, the BE can be treated deterministically until it fails, which will lead to the model using less overall computation steps per state and therefore more efficiency. For other, more complex distributions, the use of distribution functions can prove to be difficult due to, among others, performance issues and implementation limitations in UPPAAL. For these distributions a reverse lookup table of arbitrary accuracy can be used.

Furthermore, continuous degradation has to be respected for maintenance as well. Maintainable BEs can have an inspection threshold at a certain level of degradation d_i, with $d_i \in (0.0, 1.0]$, where $d_i = 1$ means the component is completely broken. The inspection time t_i can then calculated using $t_i = d_i t_f$, assuming a linear degradation process.

3.3 Fault Trees in UPPAAL

In order to be able to analyse FTs continuous degradation processes in UPPAAL (RQ4), the FT models first have to be converted to an analogous UPPAAL model, since UPPAAL uses templates with STAs. For this purpose, a set of DFT-to-template conversion strategies for the various BEs and gates are defined for this paper similar to [4].

Each type of BE or gate is modelled in a separate UPPAAL template with a STA. In the system declaration, the templates for BEs are given an identifier and probability parameters, i.e. λ for exponential distributed BEs. Gates are given an identifier, as well as the identifiers for its sub nodes, to describe a complete tree.

Here, the way of modelling is explained using a maintainable AND-gate with two inputs in UPPAAL *can be seen in fig. 6. The* WORKING *state is the initial state where both subcomponents S_1 and S_2 are functional. From there each subcomponent can fail separately, possibly causing the failure of the gate if both fail. Failed subcomponents can be repaired leading to the gate working again.*

The other SFT gates are modelled analogously. Using this method, the different templates for the FT nodes can be composed to create a complete system specification for the FT in UPPAAL. For keeping track of the different state variables of a node, i.e. current down-time and accumulated number of failures, special monitoring STAs are used in order to keep the model simple and reduce the state space.

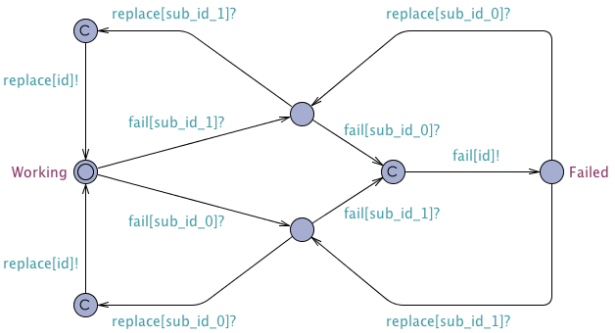

Figure 6: Maintainable AND-gate converted to an UPPAAL template

For the FT model in UPPAAL to support continuous degradation, specialized BE templates are created, as described below.

In fig. 7, a template for the continuous maintainable BE is shown. The process starts in the top left node. Since this node is marked as committed, meaning no time may pass, the time-to-fail (max_uptime_d in the model) and possibly also the inspection time (see section 3.2) are immediately calculated using the reset(t) function. The model is then in the WORKING *state or in the* WORKINGNI *state if the BE makes use of an inspection threshold. In that case, the model will trigger an inspection and transition to the regular* WORKING *state when the inspection time has been reached. From the* WORKING *state, once the computed time-to-fail has been reached, the model will transition to the* FAILING *state, and will trigger an inspection if the inspection threshold has been reached at the same time. From there the component can be repaired, which will cause the process to repeat itself and a new time-to-fail and inspection time to be calculated.*

3.4 Validation of Models

For the validation of the model (RQ5), separate smaller test cases will be used. The test cases will implemented using the proposed continuous model and the discrete model from [4]. Since both models are stochastic, they will be compared by simulating them and comparing the results statistically. If the results match, the proposed model is assumed to be valid.

The first test case uses a simple SFT with periodic replacements. The FT for this test case is shown in fig. 1. The BEs, A, B, and C, use Erlang distributed failure rates with $k = 2$ and $\lambda = 3, 2, 1$, respectively. For the replacements, all BEs use 180 as the interval.

The second text cases uses a simple DFT with periodic inspections and SPAREs. The FT for this test case is shown in fig. 2. The BEs, A, B and S again use Erlang distributed failure rates with $k = 2$ and $\lambda = 1.5, 1.0, 1.25$, respectively. While in deactivated state, the BE S only has a failure rate of $\lambda = 0.125$. For the inspections of A, B and S, the intervals 360, 360 and 180 are used respectively. The SPARE-gates used for the continuous model of this text case are a variation of the maintainable continuous BE described in section 3.3.

3.5 Case Study: Train Compressor

In order to further verify the proposed FT extension (RQ5) and retrieve reliability information from a FT (RQ6), a more real-life test case, with over 70 concurrent UPPAAL processes, will be used. The train compressor case study describes a model for an air compressor used in the pneu-

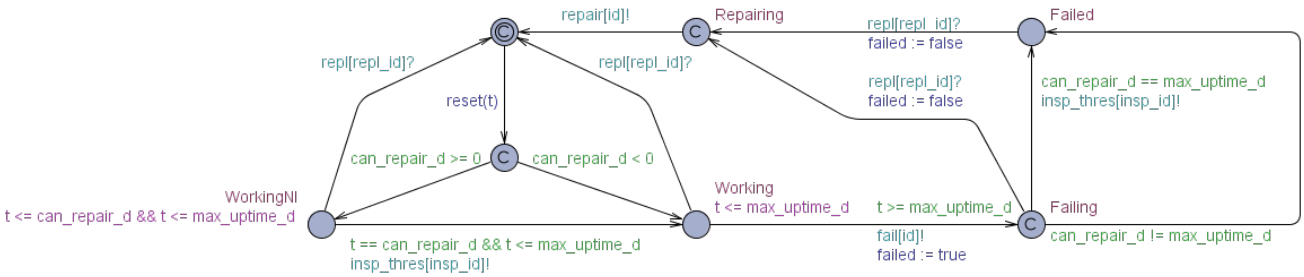

Figure 7: Continuous maintainable BE as an UPPAAL template

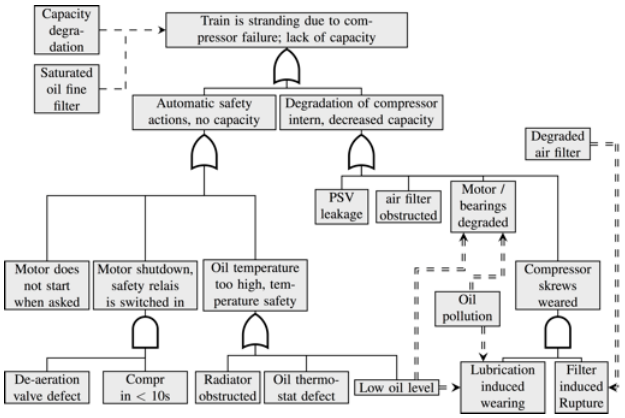

Figure 8: Overview of the train compressor model from [4]

matic system of a passenger train. It is provided by NedTrain and is also used in [4]. The pneumatic system is, among other things, responsible for braking and opening/closing doors. Failure of the compressor can lead to downtimes or possibly even causalities and therefore is to be avoided.

The compressor model makes use of discrete DFTs including periodic maintenance and specialised BEs for specific cases. In the discrete version, the BEs have different stages of degradation, each of which uses a cumulative exponential distribution for transition to next stage. For the conversion to a continuous model, these BEs can be modeled using cumulative Erlang distributions to ensure equivalent behaviour as described in section 3.1. An overview of the model can be seen in fig. 8.

As for the smaller test cases in section 3.4, the model will be simulated in UPPAAL and the results will be compared to the corresponding discrete FMT from the case study in [4].

3.6 Results

Validation Test Cases.

In Figure 9 and fig. 10, the accumulated amount of failures for each test case from section 3.4 can be seen. It shows that there is sufficient similarity between the continuous and discrete models to consider their behaviour equivalent. Therefore the proposed FT extension is assumed to be valid for these models.

Train Compressor.

In fig. 11, the accumulated amount of failures for the train compressor model can be seen. Although, the two simulation do not overlap perfectly, it shows sufficient similarity

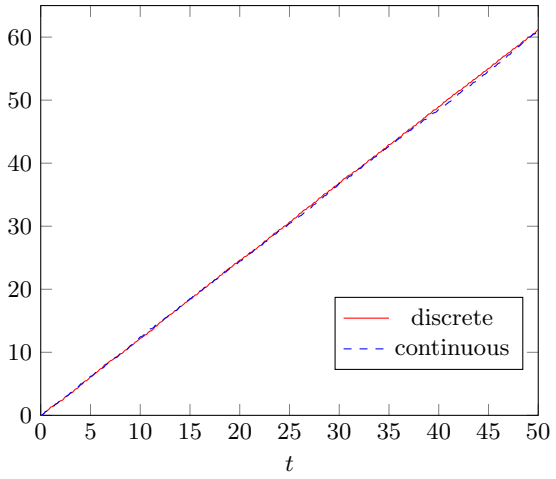

Figure 9: Case 1: accumulated amount of failures over time

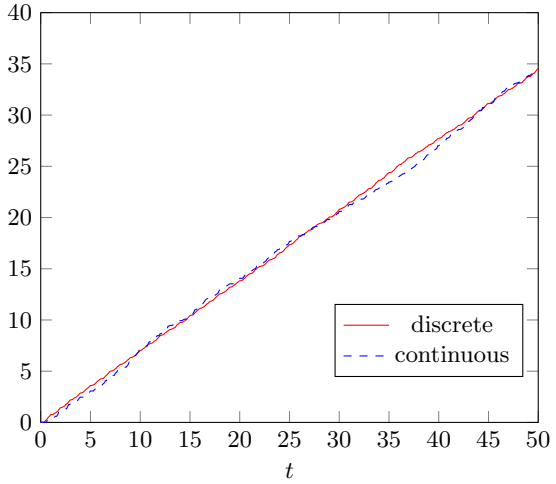

Figure 10: Case 2: accumulated amount of failures over time

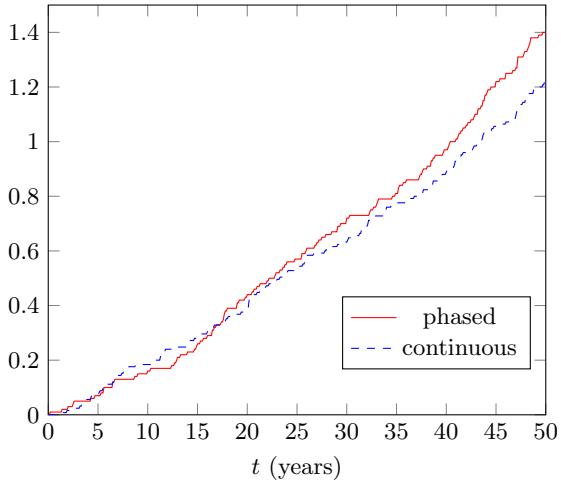

Figure 11: Train compressor: accumulated amount of failures over time

between the proposed continuous model and the model with multiple degradation phases from [4]. This is can probably be lead back to subtle differences in the calculation of failure rates in respect to reparations. Therefore, the extension is assumed to be correct for this model as well.

Furthermore, this extended model can be used to derive information about the failure behaviour of the air compressor. Although, only the average amount of system failures over time can be seen in fig. 11, more information can be retrieved from the simulation. For example, the average time-to-fail or uptime a component or the cost of repairs and inspections in the system.

4. DISCUSSION

The suggested implementation of the FT extension in UPPAAL is a proof of concept that continuous degradation can be integrated in fault trees, allowing for more precise models, that provide more accurate reliability information and therefore allow for smarter maintenance strategies.

Since UPPAAL is a multi-purpose tool for modelling and analysing state machines, it is not the most efficient solution for some more advanced calculations, ie the Erlang distribution. More efficient solutions maybe either found in dedicated extensions of UPPAAL, other existing tools, or dedicated tailored implementations to be done.

5. CONCLUSIONS

The work shown provides an extension to fault trees integrating continuous degradation processes of components, as well as strategies for analysing such fault trees in UPPAAL. The implementation of the model in UPPAAL shows that the model generates the expected behaviour, ie. the model is valid. Furthermore, it is shown that it is possible to convert existing discrete fault trees to the extended model without altering its behaviour.

Future work could include supporting more types of distributions for the degradation functions. Increasing the computational performance of degradation functions using more complex distributions, ie. the cumulative Erlang distribution, while keeping accurate models, could be looked into as well.

6. REFERENCES

[1] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 708–717, 2007.

[2] V. Crk. Reliability assessment from degradation data. *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 155–161, 2000.

[3] A. David, K. G. Larsen, A. Legay, M. Mikuionis, and D. B. Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 2015. Article in Press.

[4] E. Ruijters, D. Guck, P. Drolenga, and M. Stoelinga. Fault maintenance trees: reliability centered maintenance via statistical model checking. *Proceedings of the 62nd Annual Reliability & Maintainability Symposium (RAMS 2016), Tucson, Arizona, USA, January 25-28*, 2016.

[5] J. C. Simo and J. W. Ju. Strain- and stress-based continuum damage models-i. formulation. *International Journal of Solids and Structures*, 23(7):821–840, 1987.

[6] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick III, and J. Railsback. Fault tree handbook with aerospace applications. 2002.