Model-Based Reliability Engineering of Automotive Drivetrain Architectures With Multi-Trajectory Simulation

Sascha Schmidt[a] and Armin Zimmermann[b]
[a] Technische Universität Ilmenau, Germany, sascha.schmidt@tu-ilmenau.de
[b] Technische Universität Ilmenau, Germany, armin.zimmermann@tu-ilmenau.de

Abstract: The reliability of safety-critical systems needs to be evaluated during their design. Model-based systems engineering methods and tools for stochastic discrete-event systems support this with domain-specific models as well as simulation and analysis algorithms for complex dynamic systems. However, reliability models can become computationally expensive to evaluate, either because the state space of the systems is large or if rare events are important. The paper introduces a modular model to evaluate the reliability of fault-tolerant automotive drivetrain architectures for future electric autonomous cabs. We show how the recently proposed multi-trajectory simulation method can solve such problems efficiently, by integrating the advantages of simulation and numerical analysis.

Keywords: Automotive Drivetrain, Reliability, Rare-Event Simulation, Stochastic Petri Nets

1. INTRODUCTION

Reliability and safety are crucial design issues for numerous technical systems, especially when failures may lead to injuries or fatalities. Common examples include applications in automotive systems, reactor safety, train control, and avionics. As these are typical emergent properties, the influence of local design decisions of engineers can only be understood well with a model-based approach that evaluates the behavior at the whole system level. Mathematical models can help to describe such systems and to compute their system properties with the help of appropriate software tools, which is today understood as *model-based design* or *model-based systems engineering* [1].

This paper concentrates on reliability design of future autonomous electric cab vehicles. The common reliability design approach of traditional cars (or its negligence) is not possible for this new field any more, because any failure and subsequent crash of such a system can not be attributed to (or avoided by) a driver any more. Passengers have to completely rely on the autonomous systems' safety, which leads to stricter requirements on functional safety and its certification. Safety requirements as well as regulations of how to design and certify safety-critical systems are enforced by international standards such as the general IEC 61551 [2] and the more specific ISO 26262 [3] for automotive systems. Stochastic models for building blocks of vehicles such as the drivetrain are evaluated and predicted by the authors of [4], for instance.

The design of reliable electric cabs has to find an architecture for electric drivetrain and on-board electric distribution network as well as battery setup, which fulfills the safety requirements and leads to an efficient business case considering expenses and fees paid by passengers. There are trade-offs between high investment costs for highly reliable single modules versus using redundant solutions, and different driving mode rules have to be analyzed that decide what to do in a (partial) failure situation on the road. The set of possible modular solutions is based on existing technology, but it is not clear during the early design steps which combination and configuration is the best one for a safe, reliable and cost-efficient overall system design.

Single module faults may be masked or tolerated by static or dynamic redundancy measures. Classic models and tools for static analysis such as Fault Trees and Reliability Block Diagrams [5] are not able to cover dynamic systems in which the complex behavior influences failures. The necessary dynamic models need to consider discrete events, states, probabilistic choice and stochastic delays. Depending on the complexity of the system behavior and the corresponding size of the state space, simulation programs, Markov chains, and stochastic Petri nets (SPNs) are applied to reliability problems in the literature [5], among others. Petri nets have been suggested for reliability engineering of complex systems in an international standard [6]. The latter two are attractive as long as the underlying assumption of a Markov behavior is realistic, because a direct numerical solution is possible then [7]. The numerical analysis of models incorporating them is very restricted, only allowing the application to special cases [7] and if the number of states is not too large to be handled. An alternative evaluation technique is simulation, but the problem here is that the computational effort to generate enough failure states to achieve statistical confidence in the estimated results is too high as significant events are generated only rarely.

This problem is well-known as rare-event simulation, and there are two main approaches used: *importance sampling* [8] and *splitting* [9]. They have the common goal to increase the frequency of the rare event in order to gain more significant samples out of the same number of generated events. However, sampling is very restricted in its applicable system setups, and splitting methods require a non-trivial fine-tuning of configuration parameters, namely an importance function that guides the search for the events of interests. Multi-trajectory simulation [10, 11] is a recently proposed algorithm that integrates numerical analysis and simulation, thus allowing new trade-offs between memory space and speed. This paper demonstrates its advantages for evaluating dynamic reliability models.

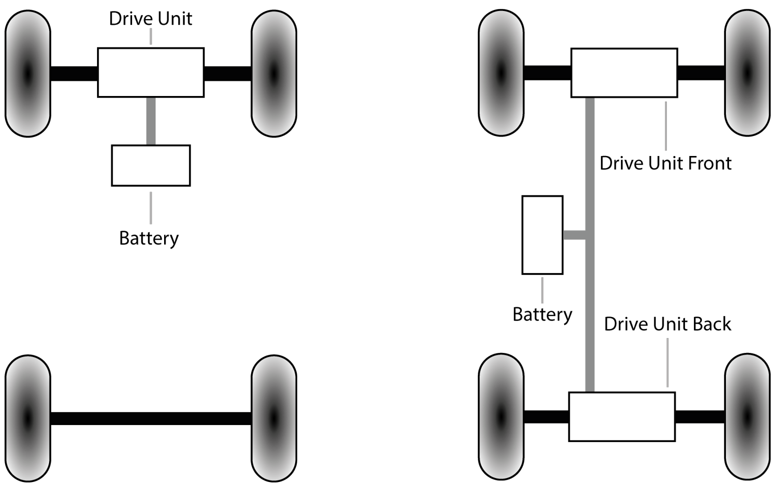
The paper is structure as follows: Section 2 introduces possible fault-tolerant drivetrain architectures of future vehicles. Stochastic Petri nets are briefly covered in Section 3, which are then used to develop a modular Petri net model for fault-tolerant drivetrains and cab operation in Section 3.1. Algorithms for the quantitative evaluation of such models is covered in Section 4. The multi-trajectory simulation algorithm is described by Section 4.1. Section 4.3 reports numerical results for systems engineering questions as well as experiments demonstrating the advantages of the multi-trajectory approach to reliability evaluation. The paper ends with Acknowledgments and Conclusion.

The contribution of the paper are stochastic Petri net models for the safety and reliability evaluation of fault-tolerant drivetrain architectures and their typical failure behaviors, as well as showing that such models can benefit from advanced simulation techniques to speed up their computation.

2. STRUCTURE AND BEHAVIOR OF FAULT-TOLERANT AUTOMOTIVE DRIVE-TRAINS

Figure 1 sketches two possible topologies of an electric vehicle. Both contain two axles, four wheels and a battery for power supply but they differ in their drivetrain design. A real-life vehicle topology comprises many more elements such as a steering system, a braking system or an electronic wiring system. Since we want to demonstrate our method on an easy to understand but relevant example, we consider a blackbox abstraction of the drive, namely the drive unit[1]. The architectures refer to common two- or four-wheel drives (2WD and 4WD, respectively). Figure 1a shows the 2WD architecture equipped by a single drive unit transmitting torque to the wheels of the front axle. The 4WD vehicle variant shown in Figure 1b is equipped with two independent drive units. One unit drives the front axle and the other unit drives the rear axle. The design of the drivetrain in both variants is assumed to operate the vehicle with only one working axle. We refer to the single drive unit variant as architecture **A** and to the variant with two drive units as architecture **B**.

Figure 1: Topologies of a centralized and axis-individual vehicle drivetrain



(a) centralized **(b)** axis-individual

Architecture **A** represents a typical central drive used both in conventional vehicles as well as in modern electric vehicles. A single drive unit sets the two wheels of an axis into a directional rotation to move the vehicle forward or backward. The international standard for functional safety (ISO 26262, [3]) imposes specific safety requirements. In the case of the vehicles braking system or the steering system, it must be ensured that they provide their functionality even in the event of a fault. A fail-safe design of these subsystems is mandatory in order to satisfy these safety requirements. Redundant architectures like the split brake system lead to a significant increase in fail-safety compared to a single circuit brake system [12]. The vehicle design of our example is assumed to fulfill these safety requirements.

Road vehicles with driver assistance systems and systems for autonomous driving are classified by the Society of Automotive Engineers (SAE) into six driving automation levels [13]. The autonomous robot cab in our example belongs to *SAE Level 4* according to the SAE International Standard (J3016, [13]). Vehicles within this level perform all driving mode-specific aspects of the dynamic driving task without the expectation that a human driver will respond when prompted to intervene [13]. In the event of a blocking failure of the drive unit in architecture **A**, the vehicle cannot safely reach a stopping point autonomously because of missing propulsion. This violates the safety requirements for highly autonomous driving. The safety requirements can be fulfilled by a highly reliable design of a single-axis drive or its redundant design. Architecture **B** represents such a redundant drivetrain design equipped with two independent drive units. An adequately dimensioned on-board power supply system is assumed in the presented architectures.

[1] Containing the electric machine, transmission, differential etc.

Identifying an optimal system design is a fundamental and challenging task in the field of systems engineering. It is necessary to specify suitable metrics for the selection and to compare different solutions from the entire design space. The model-based approach is a powerful tool for comparing solution variants. Risks can be identified in early development phases and wrong decisions in systems design can be avoided without the need of expensive prototypes.

3. STOCHASTIC PETRI NET MODELS FOR FAULT-TOLERANT AUTOMOTIVE DRIVETRAINS

Stochastic Petri nets (see [14, 15], e.g., for an overview) represent a graphical and mathematical method for the specification of processes with concurrent, synchronized and conflicting or non-deterministic activities. The graphical representation of Petri nets comprises only a few basic elements. They are therefore useful for documentation and a figurative aid for communication between systems designers. Complex systems can be described in a modular way, where only local states and state changes need to be considered. The mathematical foundation of Petri nets allows their qualitative analysis based on state equations or reachability graph, and their quantitative evaluation based on the reachability graph or by simulation.

Petri nets contain *places* (depicted by circles), *transitions* (depicted by boxes or bars) and directed *arcs* connecting them. Places may hold *tokens*, and a certain assignment of tokens to the places of a model corresponds to its model state (called *marking* in Petri net terms). Transitions model activities (state changes, events). Just like in other discrete event system descriptions, events may be possible in a state — the transition is said to be *enabled* in the marking. If so, they may happen atomically (the transition *fires*) and change the system state.

In stochastic Petri nets [16], activities may take some time, thus allowing the description and evaluation of performance-related issues. Basic quantitative measures like the throughput, loss probabilities, utilization and others can be computed. A *firing delay* is associated to each transition, which may be stochastic (a random variable) and thus described by a probability distribution. It is interpreted as the time that needs to pass between the enabling and subsequent firing of a transition. In the net class *generalized stochastic Petri nets* (GSPN [14]) that is used here, transition delays may be zero (immediate) or exponentially distributed.

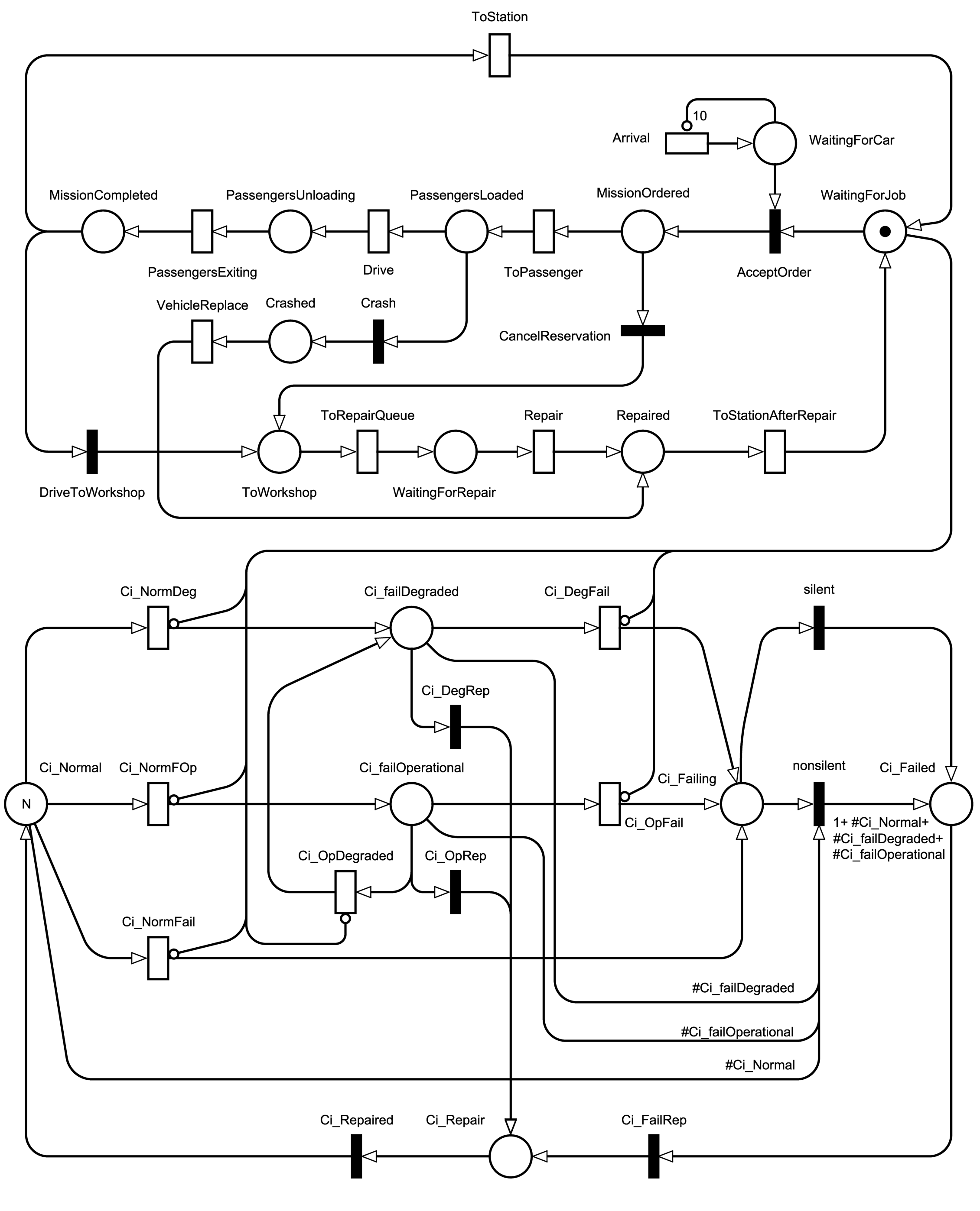
The dynamics of a Petri net are defined as follows. A transition is said to be *enabled* in a marking, if there are enough tokens available in each of its input places. Whenever a transition becomes newly enabled, a *remaining firing time* (RFT) is randomly drawn from its associated firing time distribution. The RFTs of all enabled transitions decrease with identical speed until one of them reaches zero. The fastest transition (in case of multiple ones, a probabilistic choice decides) will *fire*, and change the current marking to a new one by removing the necessary number of tokens from the input places and adding tokens to output places. The behavioral semantics of generalized stochastic Petri nets allows to map their underlying stochastic process to a continuous-time Markov chain for further analysis [14].

3.1. Petri Net Models for Typical Fault-Tolerant Drivetrain Topologies

One of the advantages of Petri net models over, e.g., Markov chain, is their compressed and modular way of describing complex systems. A fault model of a generic vehicle component embedded in an assumed drive scenario and operating strategy of a autonomous robot cab is proposed in the following text.

The described drivetrain topologies are modeled with a GSPN in the following, to evaluate performance

Figure 2: Stochastic Petri net model of robot cab driving scenario and component failure behavior



measures afterwards. Our model demonstrates the modular composition capability of different system aspects like driving dynamics, operating strategy, component failures and repair cycles as well as the effortless variation of central design decisions. The model has been constructed with the tool TimeNET [17] and is presented in Figure 2. The different aspects of the system are visually grouped to increase understandability: Driving scenario / driving behavior including the passenger demand and vehicle repair procedure is depicted in the upper part. The lower part contains the component failure module for the drive unit(s).

The normal driving behavior without component faults is modeled in the top part. The vehicle waits for a new passenger transport mission with the initial marking of the model. This is modeled by a Petri net token in place `WaitingForJob`. Passenger requests are generated by firing the transition `Arrival`. Its interarrival time thus specifies the expected customer load. Up to ten passenger requests are stored in the place `WaitingForCar`. An imaginary control center organizes the assignment of passenger to vehicles and initiates the mission. This task is modeled by the transition `AcceptOrder`. The vehicle sets off for the passenger after the accepted booking. The duration of an average 3 minutes for driving to the passengers' pick-up destination and boarding is modeled by the transition `ToPas-senger`. During the journey to the destination there is a token in the place `PassengersLoaded`. The transition `Drive` has a delay of 40 minutes and represents a usual inner city cab ride. After reaching the planned destination, the token moves to the `PassengerUnloading` location. Passenger deboarding and baggage unloading take approximately three minutes and are assigned to the `PassengerExiting` transition. Our driving scenario considers the mission to be completed after the passengers have left the vehicle at their planned destination (place `MissionCompleted`). The starting point of the fault-free driving cycle is reached again by firing the transaction `ToStation`. It represents the time required for driving to the next parking or charging station. We assume that this process takes about five minutes.

The repair routine is modeled similar to the driving cycle and starts in the place `ToWorkshop`. Together with the transition `ToRepairQueue`, the mean driving time or the transport to the workshop is modeled by a 45 minute delay. The vehicle waits in a queue `WaitingForRepair` as the workshop cannot always start the repair directly. The waiting time together with the repair time is specified by a delay of the transition `Repair` of seven hours (420 min). The vehicle returns to the operational fleet after being successfully repaired. It requires another 45 minutes and is modeled by transition `ToStationAfterRepair`. We do not consider irreparable damage inside the repair procedure in order to keep the model complexity low. There are four immediate transitions in the upper part of the model which implement the operating strategy of the fleet operator. Each of them fires according to the marking of the model during the evaluation by the definition of *guard (or enabling) functions*. As the chosen operating strategy of our application example is based on the component conditions, we will cover the component failure behavior located in the bottom part of Figure 2 first.

Our two architecture variants differ in the design and number of their drive units. While architecture **A** is equipped with only a single drive unit, architecture **B** contains two. One of the advantages of Petri nets is the ability to express such symmetric redundancies by the number of tokens. The place `Ci_Normal` represents `N` components `C` of type `i` which provide their defined target functionality. Corresponding to the topologies, $N = 1$ corresponds to architecture **A** and $N = 2$ to architecture **B**. We distinguish four detailed degradation states for drive units as described in Table 1.

Table 1: Levels of component degradation

Degradation Level	**Description**
normal	The target function is fully operable.
fail-operational	The target function remains operable for a certain time, but will eventually degrade further or fail.
fail-degraded	The target function is retained for a certain time, but its quality of service (i.e., torque output) is reduced.
failed	The target function is not provided.

The transition rates in which a degradation from the normal state to a failure state occurs are modeled by transitions `Ci_NormDeg`, `Ci_NormFOp`, `Ci_NormFail` and their corresponding places `Ci_failDegraded`, `Ci_failOperational`, `Ci_failed`. As the vehicle is operable in both *fail-operational* and *fail-degraded* levels, further degradation might occur. We assume that the condition of the component can only degrade during operation. The component can get back to the error-free status *normal* with a repair procedure which is explained later. If the component is in the *fail-operational* state for a certain period of time, it can either degrade further to *fail-degraded* or fail completely (*failed* state). If the component is already *fail-degraded*, it can only fail completely. This failure behavior is described by the corresponding transitions `Ci_OpDegradedl`, `Ci_OpFail` and `Ci_DegFail`. Failures of one component may affect other components or not. This distinction is implemented in the model with a probabilistic choice between transitions `silent` and `non-silent`. In our application example, 95% of faults are assumed to fail silently and 5% non-silently. All `N` components switch to the failed state as a result of non-silent faults. The exponentially distributed delays of all other transitions are listed in Table 2.

Table 2: Transition delays of component failure model

Transition	**Delay**	
Ci_NormDeg	259200m	180d
Ci_NormFOp	129600m	90d
Ci_NormFail	604800m	420d
Ci_DegFail	1440m	24h
Ci_OpDegraded	1440m	24h
Ci_OpFail	1440m	24h

As mentioned above, our fault-tolerant operating strategy of the vehicle is based on the condition of the drive units. The operating strategy is represented by a set of rules when and under which conditions reservations are accepted or canceled, missions are stopped and repairs are initiated. Decisions based on the system state can be implemented in the Petri net model type supported by the TimeNET tool by transition guards of immediate transitions. They represent Boolean functions on the current state of the model and only allow their transition to be enabled if they evaluate to *True*.

The upper part of our model contains the four immediate transitions `AcceptOrder`, `CancleReservation`, `Crash` and `DriveToWorkshop` implementing the operating strategy. The vehicle performs missions as long as the components do not show any degradation. A deviation from the normal driving scenario is made in the case of a degradation. If a drive unit fails on the way to a passenger after having an order already accepted we decide to cancel the reservation and redirect the vehicle to the workshop. The guard of the `CancleReservation` transition checks the amount of tokens in the place `Ci_failed`. If it equals the total number of drive units `N`, the repair process is initiated. Similar behavior with different test conditions are chosen for the other immediate transitions in the upper part of the model. A particularly severe failure is described by the sequence `Crash`, `Crashed`, `VehicleReplace`. The failure of all components in the vehicle is assumed to be a total loss which leads to an expensive new acquisition.

Like the implementation of the operation strategy depends on the state of the components, the component repair depends on the repair procedure. With the transitions `Ci_DegRep`, `Ci_OpRep` and `Ci_FailRep` the repair status is checked by the enablingFunction $\#Repaired = 1$. The transitions fire and reset the components to the normal state if this enablingFunction evaluates to TRUE. The place `Ci_Repair` is used for the calculation of performance measures and just represents a temporary state transition without a specific meaning.

The presented Petri net model integrates passenger demand, a driving scenario, operating strategy, and a generic component failure model allowing the representation of various fault-tolerant vehicle topologies in a compact way. It requires only 16 places and 25 transitions for describing the dynamic behavior and interaction of these subsystems and is constructed modularly to allow simple future extensions.

4. PERFORMABILITY EVALUATION

Performance, dependability and other quantitative measures of interest can be derived from stochastic discrete-event models such as Petri nets in two main ways [5]: either 1) by numerical analysis, which constructs the full reachable state space and solves equations that cover the stochastic process as a whole; or 2) simulation, which generates one of the many possible state-event evolutions (trajectory) and measures the values from it until the required accuracy is achieved. Both methods have their advantages and disadvantages [11]: numerical analysis gets hard or impossible when the state space becomes larger (or infinite) or when the amount of non-Markovian activity delays increases. Simulation does not have these problems and can easily cover systems with any distribution type and infinite state space, but its run time increases into impractical regions when the measures of interest converge only slowly, usually because of a so-called *rare-event problem*. These are characterized by system models in which some events happen often, while others (or some states of interest) are only visited rarely. This is a typical problem for reliability models in which the dynamics of the regular operation also has to be taken into account. Rare-event simulation methods [18] can be applied in such settings. *Multi-trajectory simulation* [10] is a recently proposed extension, which integrates both methods in a new way. Subsection 4.1 explains the algorithm.

Subsection 4.3 shows example results for the model introduced in Section 3. When more sub modules of a vehicle will be added towards a more realistic overall systems design evaluation later on, the number of states will grow significantly and numerical analysis will not be applicable any more. Moreover, the model might have to be changed such that there is no boundary on the number of reachable states, for instance as the passenger load model can change to an open load. As failures of a fault-tolerant robotic cab will happen several magnitudes less frequently than normal driving operations, this is a typical example of a rare-event problem. Subsection 4.3 shows the advantages of the multi-trajectory approach for rare-event simulation problems with the application example of this paper.

4.1. Efficient Reliability Evaluation With Multi-Trajectory Simulation

The multi-trajectory algorithm [10, 11] evaluates generalized stochastic Petri nets [14] which may include transitions with zero firing time (immediate) or exponentially distributed ones. The basic idea is to treat a model as a discrete-time Markov chain by on-the-fly automatic embedding [5], and compute transient state probabilities until convergence with a randomly evolving subset of states only.

The simulation is initialized with one *particle* for the initial state $\vec{m}_0$ at simulation time $t = 0$ with weight 1. This equals the initial probability distribution vector π_0 being 1 for the initial Petri net marking $\vec{m}_0$. We use the term particle here to denote one simulation state as part of a set.

The algorithm maintains two sets (current *Particles* and next *Particles*$'$) with elements of the form $p = (\text{marking } \vec{m}, \text{particle weight } w)$. Both are used similarly to sparse vectors in numerical algorithms. In each step of the main simulation loop, the algorithm iterates over the current set of particles *Particles*. For each particle $p \in$ *Particles* stored, two treatments are possible:

- *Propagate*: the particle is simulated until the next state-transition, like in a standard simulation by probabilistically choosing one of the subsequent states. The weight is not changed.
- *Split*: all possible subsequent states are computed, similar to an iterative step in a numerical transient solution of a discrete-time Markov chain. The weight of the particle is distributed over all descendant particles resulting from a split by multiplying it with the enabled transition's firing probabilities.

All created particles are stored in the next particle set $Particles'$. If a particle with an identical state (marking) already exists, their weights are simply added. For both cases, the simulation time is updated according to the average sojourn time in each particle, which can be computed from the delays of the enabled transitions.

For practical implementation reasons, the size of the particle sets needs to be bounded by a number of N particles to be stored. If the state space size of the model is larger than N, not every possible split will be executable. The question of whether to use propagation or splitting influences the algorithm's performance. We are working on heuristics to control the number of particles and how to spread them over the state space towards further improvements of the algorithm efficiency.

The behavior of the algorithm depends significantly on the number of particles and size of state space: For $N = 1$, only one particle will be considered, for which the weight will stay at 1. There will never be a split and the algorithm behaves like a normal simulation with one single trajectory. Numerically exact firing probabilities of all enabled transitions are computed instead and used to randomly select the next state.

If the algorithm is started with $|RS| < N$, particles for all markings $\vec{m} \in RS$ of the Petri net can be stored, and the algorithm works similar to a numerical algorithm that computes the stationary solution by converging the transient probabilities. Settings of practical interest are thus $1 \ll |RS| < N$ and $|RS| > N$.

The described multi-trajectory algorithm has been implemented as a prototype in our tool TimeNET [17, 15], which supports several model classes of stochastic Petri nets and their analysis and simulation. The detailed algorithm can be found in [10, 11].

4.2. Efficiency of the Algorithm for the Application Example

In the following, results of experiments are presented to underline the efficiency of the multi-trajectory simulation approach. We have carried out two experiments to analyze 1) the effect of the reachability graph size and 2) the "rareness" of reliability results on the run times of the different available evaluation algorithms. As this part is not about the actual results, the model from Figure 2 has been slightly reconfigured for these experiments: the firing delays of the three main failure transitions `Ci_NormDeg`, `Ci_NormFOp` and `Ci_NormFail` has been set to a variable which is adjustable to change the hardness for the simulation, and the cardinality of the inhibitor arc at transition `Arrival` is set to a second variable.

Table 3: Overall run times of evaluation algorithms (in seconds) for varying state space size

Inhibitor arc multiplicity	10	32	100	316	1.000	3.162
Number of tangible states	544	1573	4954	15538	49054	154992
numerical analysis	1	1,5	2,1	4,1	23	188
simulation	8	9	8	8	9	9
multi-trajectory simulation	1	1	1	1	1	1

Different sizes of the state space size can easily be inserted into the model by varying the second variable (inhibitor arc). This is an artificial change that does not affect the results significantly. The results of the corresponding experiments are shown in Table 3 for a logarithmic change of the arc cardinality. It becomes clear that only the numerical analysis method suffers from an increasing state space size, while simulation and multi-trajectory simulation do not become slower for larger reachability graphs (i.e., more complex system behavior). Simulation and numerical analysis use the standard TimeNET tool settings, and the multi-trajectory simulation was started with a small number of particles.

Table 4: Overall run times of evaluation algorithms (in seconds) for varying failure delays

Failure delays (in thousand minutes)	50	150	250	350	450	550	650	750	850
numerical analysis	2	2	2	2	2	2	2	2	2
simulation	7	23	30	47	79	73	88	102	116
multi-trajectory simulation	1	1	1	1	1	1	1	1	1

The second experiment analyses the impact of longer failure times (i.e., MTTF values for the low-level failures), which leads to a more reliable system and longer simulation run times as the evaluation becomes a harder rare-event problem. The multi-trajectory simulation worked best here with a higher maximum number of particles. Table 4 contains the results of this experiment, which show that only the regular simulation becomes significantly slower when the event of interest happens less frequently.

The results show that the multi-trajectory algorithm has the potential to speed up model evaluations that cannot be analyzed by numerical evaluation any more. We are working on automatic configuration settings for the algorithm and better heuristics for managing particle numbers.

4.3. Reliability Evaluation of the Application Example

The presented Petri net model can now be used to analyze the influence of the system's design parameters (i.e., redundancy or not) on performance and reliability measures of the covered vehicle topologies. The results can be used to choose balanced tradeoffs for example between investment costs and reliability or availability (influencing the earnings) of the vehicle already during early phases of the system's design process. How such values of interest can be derived and calculated is explained in this section.

All model creation tasks and computations have been carried out with the software TimeNET on a server configured with 8GB main memory and a dual core Intel Xeon E5645 CPU running at 2.4 GHz. For the *Stationary Analysis* algorithm we used the default preferences of TimeNET.

We are interested in how the two topologies perform with regard to the design of their drive units in the modeled operator strategy and driving / repair scenario. Vehicle architecture A, without a redundant drive unit, is assumed to have lower investment cost than Architecture B with a redundant drive unit design. We use assumed but realistic values for the calculation of economic values.

The relative additional costs for the redundant design of the vehicle amount to €9000 in our application example. This are 1.7 cent in cost per minute, assuming a payback period of three years and 100 percent utilization (meaning the transport of passengers) of the vehicle during a daily eight-hour operating period. This amount might still be calculated without considering the system dynamics. Our method has the advantage of determining the actual vehicle utilization depending on the travel scenario, passenger demand and operating strategy. The performance measure representing the actual vehicle utilization is the probability of a Petri net token in the place `PassengersLoaded`. Dividing the relative acquisition cost by the actual utilization results in the actual cost per minute of 2.3 cent. The used constant definitions and performance measures in TimeNET syntax are given below.

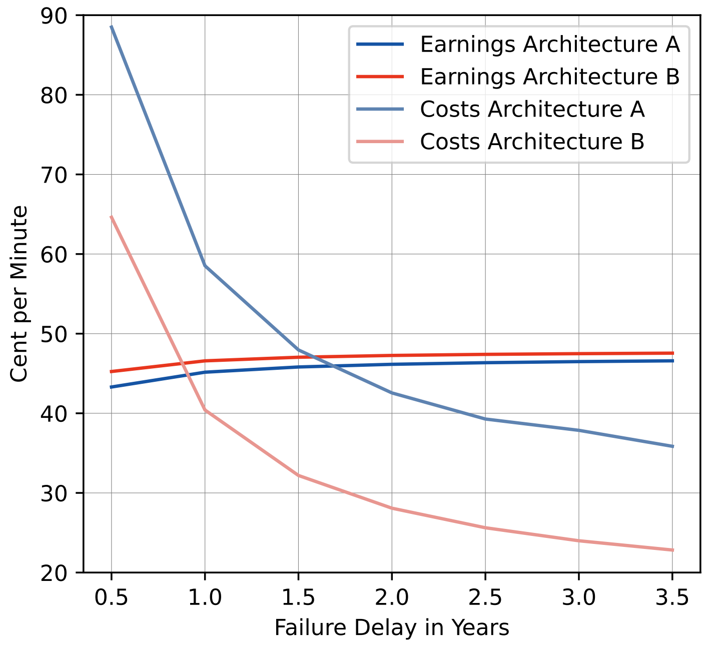
$$\begin{aligned}
Price &:= 9000\\
Amortisation &:= 3\\
CostPerMinuteNet &= (Price * N - Price)/Amortisation/360/8/60\\
ProbOnMission &= (\#PassengerLoaded > 0)\\
CostPerMinuteGross &= (Price * N - Price)/Amortisation/360/8/60/(\#PassengerLoaded > 0)
\end{aligned}$$

In addition to the initial investment, further expenses are incurred due to energy consumption during operation, repairs / replacements and imputed costs in the event of canceled missions. The average energy consumption of the vehicle is estimated to be 2.3 cents per minute of driving. This value results from an assumed energy consumption of 15 kWh per 100 km at 30 km/h average speed in an urban area and a price of 30 cent per kWh. Energy consumption in our calculations is limited to consumption by the drive units. Comfort functions such as multi-media services or air conditioning were not taken into account but could be integrated. The drive units consume energy when the vehicle is moving. The probability of being in such model states is evaluated to 91.4%. Multiplied by the daily travel time oh eight hours, the energy costs amount to 0.27 cent per minute of travel.

The costs to be considered for the complete replacement of a vehicle as well as the repair costs can be derived from the throughput of the transitions `VehicleReplace` and `Repair`. For the replacement of the vehicle we set € 80,000 for architecture **A** (Respectively € 89,000 for architecture **B**). Repairs are valued at a flat rate of € 600 regardless of the topology. Multiplied by the transition throughput, the normalized results amount to 19 cent for replacements and 0.7 cent for repairs per minute.

The last type of considered expenses are imputed costs. A negative customer experience is caused whenever an already started mission has to be aborted due to an unacceptable degradation (according to the operation strategy). A high value of € 1,000 is intended to reflect not only the lost fare but also

Figure 3: Earnings and costs of vehicle topologies for varying failure delays



compensation payable to the passenger and potential damage to the operators image. Aborted trips are derived by the difference of the throughputs of transitions `ToPassenger` and `Drive`. This results in imputed costs of 0.2 cent per minute as part of the total cost of ownership.

Operating costs are compensated by earnings from passenger-paid missions. A fee of € 25 was charged for the implemented driving scenario, corresponding to a man-driven cab ride. The modeled passenger demand together with the availability of the vehicle generates a return of 48 cent per minute.

How the two topologies perform against each other at different component failure rates was determined experimentally by varying the model parameters. As mentioned in Section 3.1, the transition `Ci_NormFail` models the degradation from a normally operating component to a failed component. The values presented at the beginning were calculated with a mean failure frequency of 420 days (or 14 months). Since the vehicle only runs one third of a day, this delay corresponds to an economic operating period of 3.5 years. We increase the frequency of normal to failed component degradation for the experiment by decrementing the delay by half per economic year in each analysis run. The results are plotted in Figure 3.

Both topologies equal in their graph shape, meaning the earnings slightly rise and costs reduce considerably with higher failure delays. The intersection of the cost and earnings curves represents the break-even point of each topology. Despite more frequent component failures, this point is achieved much earlier by the redundant topology (red) than by the topology without redundancy (blue). At the same time, the difference between revenues and costs, the profit, grows faster in the redundant topology. With our model-based approach, the two comparison topologies could be evaluated with respect to a cost-benefit tradeoff.

5. ACKNOWLEDGMENTS

The paper presents intermediate results of the AnRox project, which develops a fault-tolerant and resource-efficient electric drivetrain architecture for future robotic cabs. The consortium of 14 partners is headed by Bosch Germany, and the project is funded by the German Federal Ministry for Economic Affairs and Energy.

6. CONCLUSION

The paper covers model-based evaluation of reliability and safety of drivetrain architectures for future autonomous electric cab designs. A modular stochastic Petri net is proposed for this application example. The paper shows how design trade-offs with regard to redundancy impact costs and revenues under a certain drive cycle with maintenance operations can be evaluated based on such a model. Moreover, the computational efficiency of different evaluation algorithms for complex reliability models is compared, demonstrating the advantages of the multi-trajectory simulation approach.

REFERENCES

[1] A. Ramos, J. Ferreira, and J. Barcelo, “Model-based systems engineering: An emerging approach for modern systems,” *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 1, pp. 101 –111, January 2012.

[2] *IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems*, International Electrotechnical Commission Std., 2010.

[3] *ISO 26262 Road Vehicles : Functional Safety*, International Organization for Standardization Std., 2018.

[4] I. Bolvashenkov and H.-G. Herzog, "Approach to predictive evaluation of the reliability of electric drive train based on a stochastic model," in *2015 International Conference on Clean Electrical Power (ICCEP)*, 2015, pp. 486–492.

[5] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, 2nd ed. Wiley, 2002.

[6] "Analysis techniques for dependability — Petri net techniques," IEC 62551:2012, Sep. 2013.

[7] R. German, *Performance Analysis of Communication Systems, Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley and Sons, 2000.

[8] P. W. Glynn and D. L. Iglehart, "Importance sampling for stochastic simulations," *Management Science*, vol. 35, no. 11, pp. 1367–1392, Nov. 1989.

[9] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic, "Multilevel splitting for estimating rare event probabilities," *Operations Research*, vol. 47, pp. 585–600, 1999.

[10] A. Zimmermann, T. Hotz, and A. Canabal Lavista, "A hybrid multi-trajectory simulation algorithm for the performance evaluation of stochastic Petri nets," in *Quantitative Evaluation of Systems (QEST 2017)*, ser. Lecture Notes in Computer Science, N. Bertrand and L. Bortolussi, Eds., vol. 10503. Berlin, Germany: Springer, sep 2017, pp. 107–122.

[11] A. Zimmermann and T. Hotz, "Integrating simulation and numerical analysis in the evaluation of generalized stochastic Petri nets," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 29, no. 4, 2019.

[12] B. Breuer and K. Bill, *Bremsenhandbuch: Grundlagen, Komponenten, Systeme, Fahrdynamik*, ser. ATZ/MTZ-Fachbuch. Vieweg+Teubner Verlag, 2013.

[13] *SAE-J3216 Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, Society of Automotive Engineers Std., 2018.

[14] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, ser. Series in parallel computing. John Wiley and Sons, 1995.

[15] A. Zimmermann, *Stochastic Discrete Event Systems*. Springer, Berlin Heidelberg New York, 2007.

[16] P. J. Haas, *Stochastic Petri Nets: Modelling, Stability, Simulation*, ser. Springer Series in Operations Research. Springer Verlag, 2002.

[17] A. Zimmermann, "Modeling and evaluation of stochastic Petri nets with TimeNET 4.1," in *Proc. 6th Int. Conf on Performance Evaluation Methodologies and Tools (VALUETOOLS)*. IEEE, 2012, pp. 54–63.

[18] S. Asmussen and P. Glynn, *Stochastic Simulation: Algorithms and Analysis*, ser. Stochastic Modelling and Applied Probability. New York: Springer, 2007, vol. 57.