On Creation and Analysis of Reliability Models by Means of Stochastic Timed Automata and Statistical Model Checking: Principle

Josef Strnadel*

Brno University of Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno, Czech Republic
strnadel@fit.vutbr.cz
http://www.fit.vutbr.cz/~strnadel

Abstract. *The paper presents a method of creation and analysis of reliability models by means of stochastic timed automata statistical model checking approach available in the UPPAAL SMC tool. The method can be seen as an alternative to classical analytic approaches based on instruments such as fault-tree or Markov reliability models of the above-specified systems. Main goal of the paper is to show that – taking the advantage of the statistical model checking – the reliability analysis of systems can be facilitated even for adverse conditions such as inconstant failure (hazard) rate of inner system components. In the paper, basic terms and principles related to modeling and analysis of fault-tolerant systems are summarized, followed by a short introduction to the UPPAAL SMC tool, its practical applicability to analysis and modeling of basic fault-tolerant systems and evaluation of the results achieved on basis of the tool.*

Keywords: reliability, model, analysis, fault tolerant, FT, stochastic timed automata, STA, statistical model checking, SMC, UPPAAL SMC, probability, distribution, bathtub, fault, transient, permanent, intermittent, rate, scenario, tripple modular redundancy, TMR

1 Introduction

Technological, parametrical and other progress related to electronic systems has resulted into the rapid expansion of such systems into many application areas, including safety, time and/or mission critical ones such as anti-lock breaking or airbag control in cars, flight-crucial avionics, medical devices like pacemaker, automated control of an industrial heavy payload robot or a nuclear plant operation.

* This work was supported by the Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science – LQ1602", by the European Regional Development Fund in the "IT4Innovations Centre of Excellence" (OP VaVpI) project No. CZ.1.05/1.1.00/02.0070 and the inner university project No. FIT-S-14-2297 (Architecture of parallel and embedded computer systems).

It is a common practice that a critical system must be designed, constructed, realized and especially analyzed so that – within a given degree of confidence interval or probability interval – its predetermined, criticality-crutial properties (such as a deadlock-free operation or high availability of provided services) are guaranteed during the system operation yet before the system starts to operate under real operating conditions. This paper limits this complex problem to the design-time modeling and analysis of selected reliability-related properties of electronic systems.

This paper is organized as follows. Sections 2.1 and 2.2 outline techniques applicable for that purpose, followed by a sum of basic terms and principles related to fault tolerance (2.3) and the UPPAAL SMC tool (2.4). Section 3 presents our method based on the tool while Section 4 summarizes achieved results and finally, Section 5 concludes the paper.

2 Preliminary

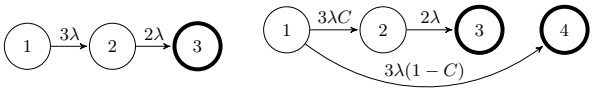
2.1 Model Checking

Various techniques can be utilized to check whether particular (typically, formally specified) properties are guaranteed under a given model of a system; in this paper it is supposed that so-called *model checking* (MC) [1] technique is utilized for that purpose. Contrary to testing, MC is able to detect all potential faults in a system and allows a designer to deal with them in early phases of a system's life cycle. MC has been implemented in several powerful tools such as SPIN [2] or SMV [3] being successfully applied in practice.

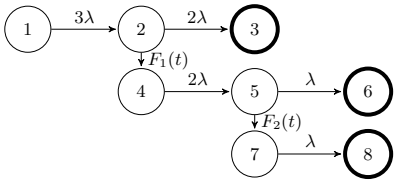
Classical MC techniques are binary (i.e. they check whether the system satisfies a property or not). Indeed, in many situations it is not enough to know whether something could or could not happen; rather, one needs to have a precise estimate of the time when some situation could arise. This motivated the creation of a number of new, so-called timed MC techniques. However, even though various optimizations and/or heuristics exist (partial order, symbolic approach, BDDs, etc.), they cannot avoid MC techniques from the state-space explosion in general, because of the complexity of problems they solve.

2.2 Statistical Model Checking

To avoid an exhaustive exploration of the state-space of a model, so-called *statistical model checking* (SMC) has been proposed – and implemented in several tools such as PRISM [4] or UPPAAL SMC [5] – as a compromise between testing and classical (binary, exhaustive) MC techniques. Simulation-based SMC methods are known to be far less memory/time intensive than classical ones, and are oftentimes the only option to approximate undecidable problems. Simply, SMC is based on monitoring some simulations of the system and their statistical processing (such as sequential hypothesis testing or Monte Carlo simulation) to estimate the satisfaction probability of a specified property under some degree



a) Triple modular redundancy (TMR) w.o. resp. with a single-point failure on the left (TMR_{NSF}) resp. right (TMR_{1SF}) of the figure



b) Triplex with successive degradation (TSD)

Fig. 1. Markov models of selected fault-tolerant (FT) systems based on static redundancy (a) and dynamic reconfiguration by degradation (b) – (c), spares (d) or both (e). λ is the permanent-failure rate, $F(t)$, $F_1(t)$, $F_2(t)$ are utilized to model the removal of a permanent fault and C is the ratio of faults not being the single point of failure; more details can be found in [15]

of confidence. The SMC approach has been applied to problems that are far beyond the scope of classical MCs and has been widely accepted in various areas such as biology [5], software engineering [6][7], aerospace applications [8][9] or system analysis [4][10][11][12][13].

2.3 Fault Tolerance

Fault tolerance (FT) [14] is typically based on some form of redundancy utilized to extend system reliability by extra resources; the *redundancy* may be in hardware, software, information, time, or combinations thereof. For hardware and software, the following types of redundancy can be distinguished: static (sometimes called passive), dynamic (sometimes called active), and hybrid.

Static redundancy masks faults e.g. by taking a majority of the results being produced by three replicas of the same module (*Triple Modular Redundancy*, *TMR*). For TMR, it is typical that its replicas are operational (active) and the majority is processed just by a *voter*, which is a *single point of failure* (*SPF*). For *dynamic redundancy*, it is typical that on top of a (primary) module being operational, one or more its spare (backup) modules stays in active (hot), warm (standby) or cold mode until the primary module fails (for that purpose, each module must be associated with a corresponding error detection circuitry able to signalize whether the module is faulty or not); result of just one of the operational plus spare modules is propagated to output – this is guaranteed by the *switch* component. Thus, dynamic redundancy is based on a sequence of the following steps: detection of a fault and recovery from the fault. The detection step helps to (locally) isolate the fault present in the primary module to avoid propagation of its effects. In the recovery step, the faulty module is replaced by one of its

spare modules and then, remaining system assets are reconfigured to operate with the spare module instead of the faulty one.

Hybrid redundancy combines both static and dynamic redundancy so that any disagreement among replicas – i.e., any mismatch between modules' outputs and the voting result – leads to replacement of faulty replica(s) by spare(s) from the common pool of spares, as long as the pool is not exhausted.

Typically, reliability of FT systems is modeled by means of classical well-known instruments such as *Markov models* [14] – for illustration, see Fig. 1 (find details in [15], please).

2.4 Concepts of Modeling in UPPAAL and UPPAAL SMC

UPPAAL [16] is a toolbox primarily designed for formal verification of *real-time* (RT) systems modeled by (a network of) *Timed Automata* (TA) extended with instruments such as typed variables and channel synchronization. SMC extension of UPPAAL (being denoted as UPPAAL SMC) has been proposed [17] to avoid the state-space explosion w.r.t. checking properties of an RT system model.

The modeling formalism of UPPAAL SMC is based on a stochastical extension of the original TA formalism from UPPAAL. On basis of the extension – called *Stochastic Timed Automata* (STA) –, one can validate properties of a given deterministic or stochastic system in given stochastic environment or conditions such as radiation or aging. In the next, concepts of (S)TA-based modeling are informally outlined.

First of all, it should be noted that a single TA [18] is formed of at least the start state, being represented by two concentric circles (for illustration, see state a in Fig. 2); a TA state is called a *location* too. A transition between two locations (let us say from a to b and denote it by $a \rightarrow b$) is represented by an oriented edge from a to b. Transition in Fig. 2a can be made anytime (but the concrete time is unknown), while transition in Fig. 2b – being conditioned by so-called *guard* (where x is a variable of the clock type) – can be made if x is 5 or later, but again: no upper bound is specified for x.

In Fig. 2c, time of staying in a is limited by so-called *invariant*, i.e., a condition defined for a location; the transition must be made before the invariant becomes false. In Fig. 2d, a guard/invariant combination is utilized to model a transition that can be made if $x \geq 5$, but must be made if $x \leq 7$, i.e., the transition is possible if $5 \leq x \leq 7$. Further TA-related instruments related e.g. to communication via channels, location types etc. are omitted herein because of the limited scope of this paper and no meaning for planned illustrative examples.

The above-mentioned principles as well as related non-deterministic behavior of TAs (such as non-deterministic choice among parallel transitions between the

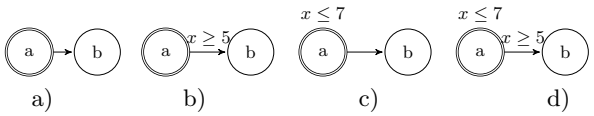


Fig. 2. Illustration to basic TA terms: place, transition, guard, invariant

same locations) are refined in STAs by stochastic ones, being briefly illustrated in the next. For example, weight annotations on locations are extended to model the staying in a location using a probability distribution; e.g., in Fig. 3a, the staying in a (i.e., entering b) is given by the exponential distribution with the rate (λ) set to $\frac{1}{2}$. In Fig. 3b, the probabilistic uniform-distribution choice between $a \to b$ (with probability $\frac{1}{5}$) and $a \to c$ (with probability $\frac{4}{5}$) is modeled. Fig. 3c illustrates the following (so-called *stopwatch*) concept able to determine the exact time that has elapsed. In Fig. 3c, the clock x is reset in parallel with setting a value (being produced by a user-defined function f) to the *delay* variable of the clock type (during $a \to b$) first; then, staying in b cannot be longer than for *delay* units of time being measured by x while *delay* is stopped ($delay' == 0$) in b. Finally, $b \to c$ is possible just if x matches *delay*.

Properties of an STA-based model can be verified (checked) using special queries a user can post in the UPPAAL SMC tool w.r.t. model; among others, queries from the following areas are supported:

- *Probability estimation* – a question in the form $Pr[bound](\phi)$ is used to get the probability that something (ϕ) – such as entering a state/place – happens under the specified *bound*,
- *Hypothesis testing* – a question $Pr[bound](\phi) \geq p$ can be posted to check whether the probability of something (ϕ) is greater or equal to a certain probability treshold (p) under the specified *bound*,
- *Probability comparison* – a question in the form $Pr[bound_1](\phi_1) \geq Pr[bound_2](\phi_2)$ can be posted to check whether the probability of ϕ_1 is greater or equal to ϕ_2 under the specified $bound_1$, $bound_2$,

where *bound*, $bound_1$, $bound_2$ define how to bound – e.g. the number of – simulation steps (runs), ϕ, ϕ_1, ϕ_2 are assertions (formulas) to check and p is a real-number value. E.g., for Fig. 3a one can post the `Pr[<=3000](<> STA.b)` query to get the probability of eventual entering the state b within 3000 units of the simulation time. A possible (probabilistic) result of the query is vizualized in Fig. 4. For further examples, please see [17].

3 Proposed Method

The method presented in this paper has been initially inspired by [19] approach giving an idea of creating a model of basic components for constructing FT

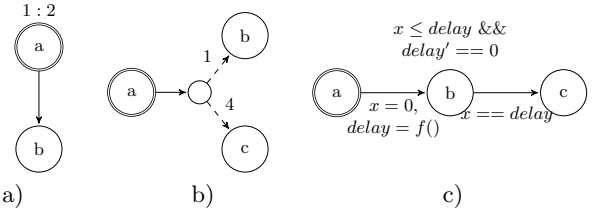


Fig. 3. Illustration to basic STA terms: place, transition, guard, invariant

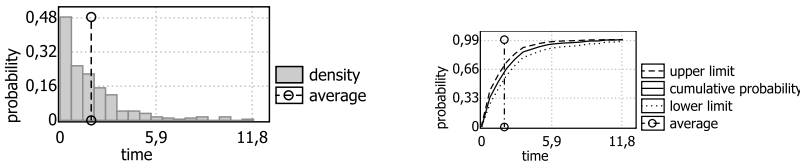


Fig. 4. Illustration to probability of entering b (left) and cumulative probability with confidence intervals (right) of that for the STA model from Fig. 3a

systems to verify properties of such systems by means of formal verification in classical UPPAAL. However, the approach was based on a timed – but deterministic – model and exhaustive verification with binary (i.e., yes/no) answers to questions about properties. We have decided to utilize a completely different approach, allowing i) creation of probabilistic models (such as those from Fig. 1) and ii) statistical model checking instruments to be applied to our models. For the purpose, several models must be created, e.g., by means of STAs supported by the UPPAAL SMC tool – details to the models follow in the sections 3.1, 3.2 and 3.3.

3.1 Probability distribution models

First, probability distribution models must be created to model various failure rates. Basically, it is not a problem to create almost any time-dependent failure rate function; however, because of limited space in this paper, we have decided to present just a simplified model of the typical bathtub curve. The curve consists of several regions – such as early (infant mortality), constant, aging, wear out, break in – each representing different progress of the failure rate. In fact, those regions seems to be enough to show that by means of STAs, it is possible to cover a wide range of very different rate functions.

The skeleton of our STA-based model of the bathtub curve is vizualized in Fig. 5, where x is the clock-type variable, EARLY, CONST and WEAR-OUT are probability weights, EXP_RATE is a rate of the exponential probability distribution and T^{FROM}_{CONST}, T^{TO}_{CONST}, $T^{FROM}_{WO_i}$, $T^{TO}_{WO_i}$, $i = 1, \ldots, n$, are constants specific to a particular bathtub-part shape.

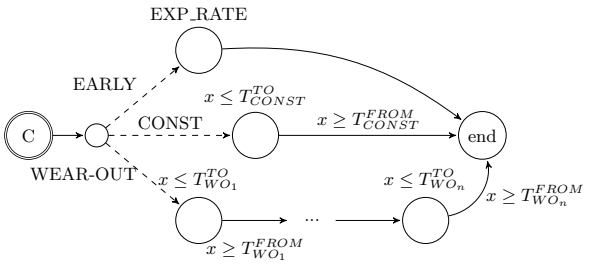


Fig. 5. Idea of composing a probability distribution model by means of STAs in the UPPAAL SMC tool (bathtub failure rate example)

Basically, a separete branch exists in the model for each region in the curve. From the initial state, the STA can transit into one of the three consecutive states (starting a particular branch), each being enabled with different probability – typically, very small values of probability are utilized for the early and wear out regions of the curve, while high value is utilized for its constant-rate region. Interval in which a fault of a given rate is allowed to appear can be limited later in the particular branch.

For example, a failure from the early-rate region can occur rarely – this is guaranteed by the low ($EARLY$) probability weight – and with the exponential probability distribution given by the EXP_RATE parameter; a failure from the constant-rate region can occur more often – that is defined by the high ($CONST$) probability weight –, but with its rate uniformly distributed in the $< T^{FROM}_{CONST}, T^{TO}_{CONST} >$ interval with a lower resp. upper bound defined by a guard resp. invariant (see Fig. 2d). Alike, the probability distribution for the wear-out region can be formed e.g. using a chain of consequent, properly-shaped uniform distributions of a failure occurence.

3.2 Fault generation models

In the next step, it is necessary to model fault generators. A fault generator is required to produce a fault with times of its occurence being defined by a probability distribution model (such as from 3.1) of the rate corresponding to the fault. When the time comes for the occurence of a fault in a system, the fault is introduced so that an instance of the STA (representing the corresponding behavioral model of the fault – for such a model, see 3.3) is dynamically created. To create a fault dynamically, the *spawn* keyword must be utilized in STA.

This kind of modeling is very close to the reality – after its introduction to a system, a fault can remain there for a predetermined time and then disappear and never show again or occur/disappear repeatedly (timing may vary accross faults of the same type) or, a fault may last until it is removed.

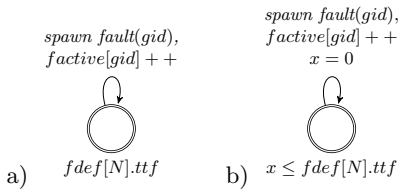


Fig. 6. Illustration to dynamic creation of faults with rates described using a) exponential resp. b) constant failure rates. $factive[gid]$ is utilized as a counter of the number of active faults dynamically created by the generator identified by *gid*. For better readability, $fid[gid]$ is substituted by N

In Fig. 6, an example of dynamic creation of faults with rates described using a) exponential resp. b) constant, i.e. uniformly distributed, failure rates is

illustrated; for the bathtub case, the *spawn* construction will be applied to the *end* state of the STA from Fig. 5.

Listing 1.1. Basic components of a fault in our model

```
1  typedef struct {   // fault:
2     t_ftype ft;     // - type: 0-perm., 1-tran., 2-int.
3     t_ttf ttf;      // - time to (occurence of a) fault: 0, 1, ...
4     t_pdist pttf;   // - probab. distr.: 0-uni., 1-exp.,
5                     //                   2-norm., 3-bathtub, 4-early, ...
6     t_ttd ttd;      // - time to disappear: 0, 1, ...
7     t_pdist pttd;   // - probab. distr.: 0-uni., 1-exp., ...
8  } t_sFault;
```

Let it be noted there that a fault is of a given type (see Listing 1.1) allowing to create multiple fault definitions in $fdef[]$ (see Listing 1.2).

Listing 1.2. Illustration to declaration of an array with fault definitions

```
1  const t_sFault fdef[t_nfault] = {
2  //  ft    ttf    pttf    ttd   pttd   array-index
3      {1,   100,     0,    100,     0},     // 0
4      {0,   500,     0,    500,     0},     // 1
5      {1,     1,     1,      1,     1},     // 2
6      {1,     5,     1,      5,     1}      // 3
7  };
```

The mapping $fid[]$ (see Listing 1.3) is needed to make a relation between the definition of a fault and the generator.

Listing 1.3. Example to mapping of a fault generator onto an index to `fdef[]`

```
1  // generator indexes:    0  1  2 ...
2  t_nfault fid[t_ngen] = { 0, 1, 1 };  // fdef[] indexes
```

3.3 Fault behavior models

After a fault is introduced into a system, it can behave in a way that can be defined by a special STA. In Fig. 7, a skeleton of the behavioral model for intermittent, permanent and transient types of faults is illustrated, focusing to the branch for transient faults. A transition from the initilal state is enabled for a particular fault type (F_{INTER}, F_{PERM} or F_{TRAN}). In the next state, a branch for modeling duration of the fault is selected (P_{UNI} resp. P_{EXP} for uniform/-constant resp. exponential probability distribution of the durations, based on the STA design patterns from Fig. 2d resp. Fig. 3a).

Before reliability models (such as those from Fig. 1) can be created on basis of the above-mentioned modeling techniques, an instrument able to signalize the occurence of a fault is needed to make the construction of reliability models as straightforward as possible. In our approach, we have based the signalization mechanism on the STA from Fig. 8. The mechanism relies on sending a message via the broadcast channel named $fail$ while at least one active fault exists that has been produced by the corresponding fault generator (identified by *gid*). Such an STA is created for each fault generator in a system. If ! resp. ? follows

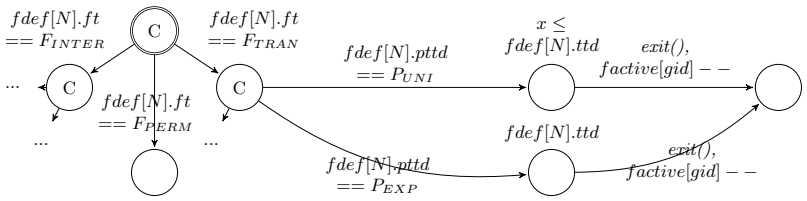


Fig. 7. Illustration to the behavioral model of a fault. After its duration is over, a dynamically created transient fault removes itself from a system by calling $exit()$ and decrementing the number ($factive[gid]$) of active faults introduced by the same generator. For better readability, $fid[gid]$ is substituted by N

the channel name (i.e., $fail!$ resp. $fail?$ is associated with a transition) then a message is sent resp. expected via the $fail$ channel reserved for a fault generator.

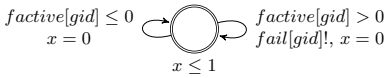


Fig. 8. Illustration to a fault signalization mechanism

3.4 Reliability models

In the next, a method for construction of reliability models by means of the above-mentioned modeling techniques is described. To create a reliability model, definitions of considered faults must be prepared (such as in Listing 1.2) first in $fdef[]$. Then, a decision about the number of fault generators – each able to produce a fault of a given definition – must be made and stored into $fid[]$. Let it be noted that it is possible to create multiple generators for the same fault definition; for example, in Listing 1.3 three generators are utilized, where generator identified by $gid = 0$ is associated with the fault definition $fdef[gid] = (1, 0, 100)$, i.e. a transient fault with probability of its ocurence uniformly distributed over 100 units of time while generators identified by $gid = 1$ and $gid = 2$ are associated with $fdef[gid] = (0, 0, 500)$, i.e. a permanent fault with uniform probability distribution over 500 units of time.

Once STA-based models for the fault generator, fault behavior and fault signalization are created, process of creation of a reliability model can be started. In the next, an idea of such a process is discussed in the form of a straightforward transformation from the classical models from Fig. 1. Because of the limited space in this paper and simplicity of the transformation, the resulting STA-based models are omitted herein. Key principle w.r.t. our model relies on replacing λ – or similar probabilistic quantity such as $F(t)$, $F_1(t)$, $F_2(t)$, $F_z(t)$, $W_1(t)$, $W_2(t)$ from Fig. 1 – in a Markov model by waiting for a message on the $fail$ channel (Fig. 9).

Fig. 9. Principle of converting a Markov model (a) to an STA model (b). N identifies a generator producing a fault the edge is sensitive to

With no impact to the generality, all important design patterns w.r.t reliability models can be presented over a simple TMR model from Fig. 1a. Basically, a separate fault generator is needed for each of independent faults (TMR_{NSF} from Fig. 1a, Fig. 10a) – in such a case, separate fault signalization mechanisms are available. On the contrary, signalization of the same fault can be utilized e.g. to model a SPF (TMR_{1SF} from Fig. 1a, Fig. 10b).

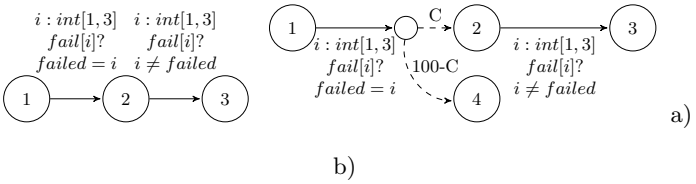


Fig. 10. STA-based realization of the TMR models from Fig. 1: a) TMR_{NSF}, b) TMR_{1SF}. C is probability that a fault is not SPF

STA-based versions of the TMR are depicted in Fig. 10. In a), the $1 \rightarrow 2$ transition is utilized to wait for a reception of a signal over three channels, i.e., $fail[i]$, $i = 1, 2, 3$, each belonging to one of the three replicas in the TMR system. If a fail signal is received on a particular channel then the corresponding value of i is stored into $failed$ to identify the failed replica. The consecutive $2 \rightarrow 3$ is sensitive just to a failure in replicas identified by $i = 1, 2, 3$, $i \neq failed$, i.e., it is sensitive just to a failure of the remaining two replicas. The same principle is applied in the case b), extended to model a SPF by means of a probabilistic choice made at the end of the transition outgoing from 1 ,i.e., after one of the three replicas fails; then, TMR can either operate in a two-replica mode (if it transits to 2; here, it operates in the same way as in the a) case) or it can be a subject to a SPF and fail (if it goes to 4).

4 Evaluation

To show practical applicability of our above-mentioned modeling techniques and its benefits, we have decided to present few results produced on basis of our models (see Fig. 11, Fig. 12).

4.1 Selected Results

The results from Fig. 11 were produced on basis of the model-checking query $Pr[<= 100](<> sta.end)$, being applied consequently to special cases of the Fig.

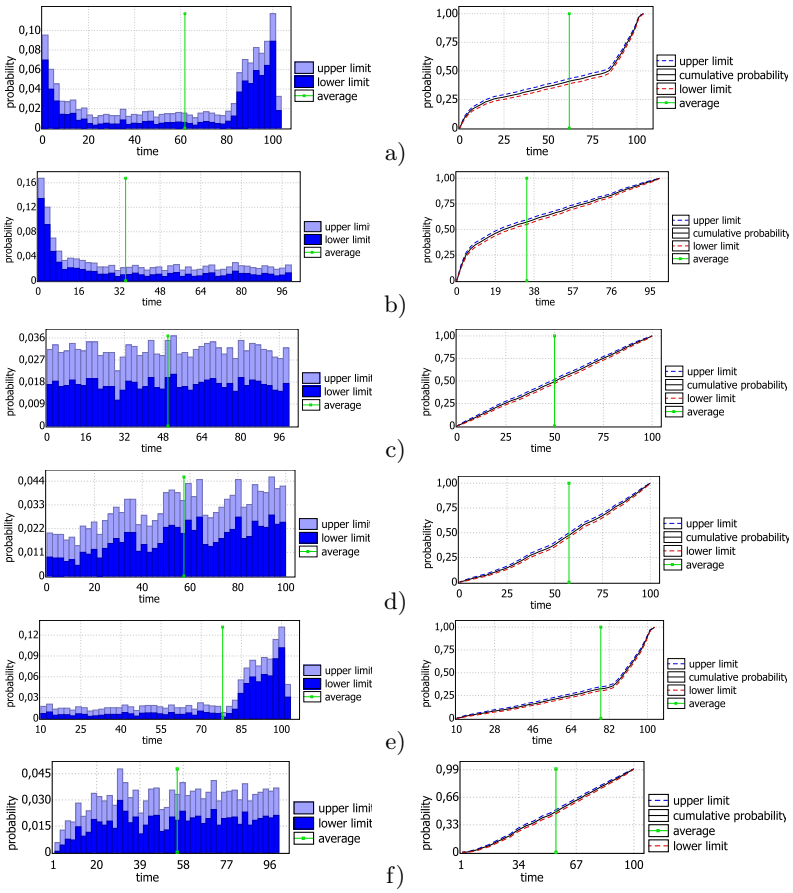


Fig. 11. Illustration to probability (left column) and cumulative probability (right column) with confidence intervals for selected failure rates models realized by means of UPPAAL SMC: a) bathtub, b) early (infant mortality), c) constant, d) aging, e) wear out, f) break-in

5; *sta* is the name of an STA representing the bathtub model. It can be seen that STA-based models are able to cover all regions of the bathtub, allowing us to analyze reliability under different fault rate scenarios.

The results from Fig. 12 have been achieved on basis of several queries, details of which follows; in all cases, one simulation run has been performed, which is denoted by *simulate* 1 at the beginning of the queries. For a) – d), the query *simulate* 1 $[<= N]\{numOf(fault), n\_fin/K\}$ has been utilized with $N = 200$ resp. $N = 2000$ for a), b) resp. c), d) and $K = 1, 50, 100, 200$ for a), b), c), d). For 12e) resp. f), *simulate* 1 $[<= 1000]\{numOf(fault), 2 * n\_fperm, n\_ftrans\}$ resp. *simulate* 1 $[<= 2000]\{factive[0], 10 + factive[1], 20 + factive[2], 30 + factive[3], 40 + factive[4], 50 + factive[5]\}$ queries have been utilized.

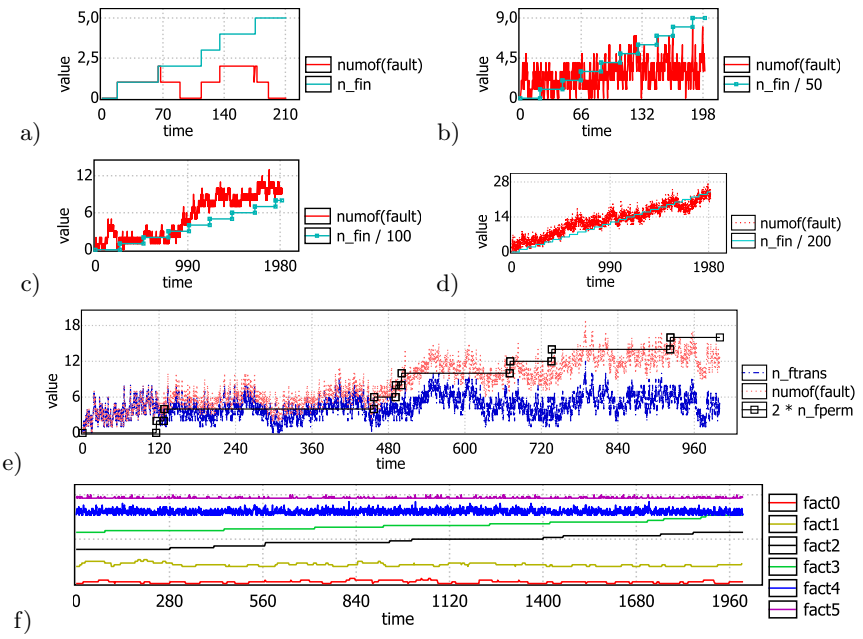


Fig. 12. Results produced on basis of various declarations of `fid[]`: a) { 0 }, b) { 0, 2, 3 }, c) { 0, 1, 3 }, d–f) { 0, 0, 1, 1, 2, 3 }. In a)–d), an evolution of the number of all incomming faults (n_fin) and of faults in the system ($numOf(fault)$) are visualized. In e), a relation among the number of permanent ($2 * n\_fperm$), transient ($2 * n\_ftran$) and all faults is depicted. In f), an evolution of the number of active faults ($fact_i$) produced by a particular fault generator (indexed by $i = 0, 2, \ldots, 5$) is visualized

4.2 Benefits

Benefits of utilizing modeling instruments proposed in this paper can be summarized as follows. First, our models are scalable and our solution is open to additions – proposed models can be easily extended to further types of fault rates, behavior types etc. Moreover, our approach can be utilized to analyze properties of systems

- with variable failure rates described e.g. by means of the complete bathtub curve(s) rather than just by their isolated portions,
- formed of dependent components or suspensible to dependent faults,
- dynamic, evolvable/reconfigurable systems able to add, remove their components and/or change their parameters at run-time,
- in the context of further parameters such as liveness, safety and/or timing, power and other constraints.

Second, model checking engine in the UPPAAL SMC tool can be utilized to simply check key properties (such as probability that something, like a failure, may happen) w.r.t. a system being modeled.

Third, transformation of existing reliability models (such as widely-utilized Markov models) is straightforward and there is no need to solve any system of equation by your own.

It can be concluded that the benefits represent a very good prerequisite for rapid prototyping as well as reliability analysis of FT systems under various fault scenarios and applied FT techniques.

5 Conclusion

In the paper, a method of creation and analysis of reliability models by means of the STAs and SMC approach supported by the UPPAAL SMC tool has been presented. The method allows more precise and close-to-reality modeling comparing to classical approaches such as Markov reliability models. Further activity w.r.t. topic of the paper can be seen especially in

- applying the proposed method to complex, practical FT systems, systems with dynamic redundancy and hybrid (i.e., discrete/analog) systems,
- reliability analysis throught multiple bathtub regions and of particular system classes such as memories, CPU cores or operating system kernels,
- analyzing an impact of multiple faults of same/different type to reliability of an FT system equipped by particular FT techniques.

References

1. C. Baier and J.-P. Katoen, *Principles of Model Checking*, ser. Representation and Mind. MIT Press, 2008. [Online]. Available: `https://mitpress.mit.edu/books/principles-model-checking`
2. G. J. Holzmann, "The model checker spin," *IEEE Transactions on Software Engineering*, vol. 23, pp. 279–295, 1997.
3. K. L. McMillan, "Symbolic model checking: An approach to the state explosion problem," Ph.D. dissertation, Pittsburgh, PA, USA, 1992, uMI Order No. GAX92-24209. [Online]. Available: `http://www.kenmcmil.com/pubs/thesis.pdf`
4. M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic model checking for performance and reliability analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 40–45, Mar. 2009. [Online]. Available: `http://doi.acm.org/10.1145/1530873.1530882`
5. A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, and S. Sedwards, "Statistical model checking for biological systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 17, no. 3, pp. 351–367, Jun. 2015. [Online]. Available: `http://dx.doi.org/10.1007/s10009-014-0323-4`
6. C. Dubslaff, S. Klüppelholz, and C. Baier, "Probabilistic model checking for energy analysis in software product lines," in *Proceedings of the 13th International Conference on Modularity*, ser. MODULARITY '14. New York, NY, USA: ACM, 2014, pp. 169–180. [Online]. Available: `http://doi.acm.org/10.1145/2577080.2577095`
7. R. Calinescu, C. Ghezzi, K. Johnson, M. Pezze, Y. Rafiq, and G. Tamburrelli, "Formal verification with confidence intervals to establish quality of service properties of software systems," *IEEE Transactions on Reliability*, vol. PP, no. 99, pp. 1–19, 2015.

8. K. Hoque, O. Ait Mohamed, Y. Savaria, and C. Thibeault, "Early analysis of soft error effects for aerospace applications using probabilistic model checking," in *Formal Techniques for Safety-Critical Systems*, ser. Communications in Computer and Information Science, C. Artho and P. C. lveczky, Eds. Springer International Publishing, 2014, vol. 419, pp. 54–70. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-05416-2_5
9. Y. Lu, Z. Peng, A. A. Miller, T. Zhao, and C. W. Johnson, "How reliable is satellite navigation for aviation? checking availability properties with probabilistic verification," *Reliability Engineering & System Safety*, vol. 144, pp. 95 – 116, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832015002252
10. N. Benes, B. Buhnova, I. Cerna, and R. Oslejsek, "Reliability analysis in component-based development via probabilistic model checking," in *Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering*, ser. CBSE '12. New York, NY, USA: ACM, 2012, pp. 83–92. [Online]. Available: http://doi.acm.org/10.1145/2304736.2304752
11. A. Basu, S. Bensalem, M. Bozga, B. Delahaye, and A. Legay, "Statistical abstraction and model-checking of large heterogeneous systems," *International Journal on Software Tools for Technology Transfer*, vol. 14, no. 1, pp. 53–72, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10009-011-0201-2
12. Z. Peng, Y. Lu, A. Miller, C. Johnson, and T. Zhao, "A probabilistic model checking approach to analysing reliability, availability, and maintainability of a single satellite system," in *Modelling Symposium (EMS), 2013 European*, Nov 2013, pp. 611–616.
13. P. Swain, P. Bhaduri, and S. Nandi, "Probabilistic model checking of ieee 802.11 ibss power save mode," *Int. J. Wire. Mob. Comput.*, vol. 7, no. 5, pp. 465–474, Sep. 2014. [Online]. Available: http://dx.doi.org/10.1504/IJWMC.2014.064818
14. I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.
15. B. Ricky W. and J. Sally C., "Techniques for modeling the reliability of fault-tolerant systems with the markov state-space approach," Tech. Rep., 1995. [Online]. Available: http://shemesh.larc.nasa.gov/fm/papers/Butler-RP-1348-Techniques-Model_Rel-FT.pdf
16. G. Behrmann, A. David, and K. Larsen, "A tutorial on uppaal," in *Formal Methods for the Design of Real-Time Systems*, ser. Lecture Notes in Computer Science, M. Bernardo and F. Corradini, Eds. Springer Berlin Heidelberg, 2004, vol. 3185, pp. 200–236. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30080-9_7
17. A. David, K. Larsen, A. Legay, M. Mikuionis, and D. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10009-014-0361-y
18. R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, Apr. 1994. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(94)90010-8
19. M. Zhang, Z. Liu, C. Morisset, and A. Ravn, "Design and verification of fault-tolerant components," in *Methods, Models and Tools for Fault Tolerance*, ser. Lecture Notes in Computer Science, M. Butler, C. Jones, A. Romanovsky, and E. 0.5em minus 0.4emSpringer Berlin Heidelberg, 2009, vol. 5454, pp. 57–84. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00867-2_4