Increasing the reliability of single and multi core systems with software rejuvenation and coded processing

Juergen Braun[1], Juergen Mottok[1], Christian Miedl[2], Dirk Geyer[2], Mark Minas[3]

[1] - University of Applied Sciences Regensburg, LaS³,
Seybothstraße 2, D-93053 Regensburg,
{Juergen.Braun;Juergen.Mottok}@hs-regensburg.de
[2] - AVL Software & Functions GmbH,
Im Gewerbepark B27, D-93059 Regensburg,
{Christian.Miedl;Dirk.Geyer}@avl.com
[3] - Universität der Bundeswehr München,
D-85577 Neubiberg,
Mark.Minas@unibw.de

Abstract: The safety of electric vehicles has the highest priority because it helps contribute to customer confidence and thereby ensures further growth of the electromobility market. Therefore in series production redundant hardware concepts like dual core microcontrollers running in lock-step-mode are used to reach ASIL D safety requirements given from the ISO 26262.

Coded processing is capable of reducing redundancy in hardware by adding diverse redundancy in software, e.g. by specific coding of data and instructions. A system with two coded processing channels is considered. One channel is active and one is in cold standby. When the active channel fails, the service is switched from the active channel to the standby channel. It is imaginable that the two channels with implemented coded processing are running with time redundancy on a single core or on a multi core system where for example different ASIL levels are partitioned on different cores.

In this paper a redundant concept based on coded processing and software rejuvenation will be taken into account.

1. Introduction

Nowadays the importance of the reliability in safety-critical and life-critical systems is well recognized [Bao05]. Redundancy does not only mean the duplication of systems. By definition, all additional units and methods are included, that are implemented for error detection and error avoidance. Such methods are for example integrated test units as well as error-detecting codes and coded processing, like Safely Embedded Software (SES) [Mot12]. With embedded multicore microcontrollers software aging [Par94] is getting a more and more important point even for embedded systems. Thereby the hardware fault tolerance is well understood since many years, but the software fault tolerance is often the origin of most of the reliability problems. In a standard or low performance embedded system there are nearly no dynamic factors, like the static memory allocation, but in high performance or future embedded systems a dynamic behaviour will be necessary. At the moment these issues have less relevance for safety-critical systems. For instance, in the case of software that undergoes a safety certification process, dynamic memory management is typically avoided [Cor11]. A preventive and proactive solution to software aging is software rejuvenation involving the restoration of a system to a clean internal state [Sar09]. The proactive fault management has to be added as a complementary approach in addition to the traditional reactive recovery mechanisms with subject to aging. At a certain time with the proactive fault management a rejuvenation operation is scheduled, so that potential error conditions could be removed. There are two different approaches for software aging and rejuvenation. The measurement based are statistical analysis for predicting a time window where rejuvenation should be prefered [Cas01] [Gar98] [Li02] [She03] [Vai99]. An optimal rejuvenation time could not be determined with this statistical approach. The analytical approach is done with a Markov process or with stochastic Petri net models. With this models it is possible to compute the system availability [Doh00d] [Doh00n] [Gar95] [Hua95] [Vai01]. During my research studies for coded processing I have been inspired by the work [Kou10] for the following considerations to proof with a Continuous Time Markov Chain that software rejuvenation between different coded processing channels improves the availability of the embedded system significantly, compared with an embedded system without any rejuvenation actions and without coded processing.

The paper is structured as follows. After the presentation of the principle of Coded Processing based on the example of Safely Embedded Software in section 2, an introduction of Software Rejuvenation is given in section 3. In section 4 the redundant systems without and with Safely Embedded Software in combination with Software Rejuvenation are modelled, which are the basis for section 5 in which the Mean Time To Failure is calculated with the help of a Continuous Time Markov Chain. Section 6 contains some conclusions and an outlook for future work.

2. Coded Processing

The concept of coded processing is capable of reducing redundancy in hardware by adding diverse redundancy in software, e.g. by specific coding of data and instructions. Hardware and software coding can be combined using approaches like the Vital Coded Processor [For89]. Consequently besides the actual safety-critical control program, also the other programs can run on the same hardware. Thus it is possible to specifically protect only the safety-critical parts of the control program. Coded Processing enables the verification of safety properties and fulfills the condition of single fault detection [Mot12]. Coded Processing does not constrict capabilities but rather supplements multi-version software fault tolerance techniques like N version programming, consensus recovery block techniques, or N self-checking programming [Mot12]. In this paper the Safely Embedded Software approach is described in more detail.

2.1. Safely Embedded Software (SES)

The given SES approach generates the safety of the overall system in the application software level. SES is based on the (AN+B)-code of the Coded Monoprocessor transformation of original integer data x_f into diverse coded data x_c [Mot12].

Coded data fulfills this relation:

$$x_c = A \cdot x_f + B_x + D_t \text{ where } \begin{matrix} x_c, x_f \in \mathrm{Z}, A \in \mathrm{N}^+, B_x, \\ D_t \in \mathrm{N}_0, B_x + D_t < A \forall x, t \end{matrix} \tag{1}$$

The prime number A is important for safety characteristics like Hamming Distance and residual error probability $P_{re} = 1/A$ of the code [Mot12]. Hamming distance gives information about the maximum number of fully detectable bit errors and residual error probability P_{re} is the probability that the corruption of the data remains undetected after performing the decoding procedure [Sch06]. Number A has to be prime because in case of a sequence of i faulty operations with constant offset f, the final offset will be $i \cdot f$. If A is not a prime number then several factors of i and f may cause multiples of A. If a prime number is used as A, this offset is only a multiple if i or f is divisible by A. This is the same fact for the multiplication of one or two faulty operands. Additionally, so called deterministic criteria like the above mentioned Hamming Distance and the Arithmetic Distance must be considered for choosing an adequate prime number. Other functional characteristics like necessary bitfield size and consequential handling of overflow are also caused by the value of A. The simple transformation $x_c = A \cdot x_f$ is illustrated in Fig. 1.

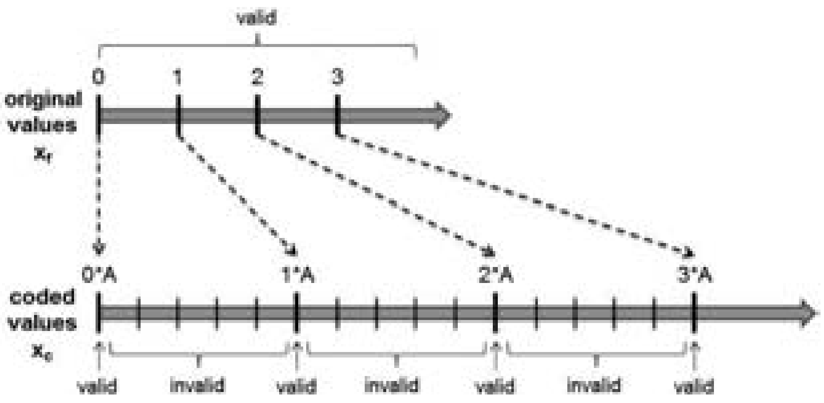

Figure 1: **Simple transformation for $x_c = A * x_f$.**

To ensure the correct memory addresses of the variables any user defined specific number or the memory address of the variable itself could be used as static signature B_x [Mot12]. The dynamic signature D_t ensures that the variable is used for example in the correct task cycle or function call.

The two software channels (original data and coded data) could be verified for example at the end of each task cycle before starting the output of the calculated values. Therefore the instructions are coded in such a way that, either a comparator could verify the diverse channel results for the condition $z_c = A \cdot z_f + B_z + D_t$, or the coded channel could be checked directly by the verification condition $(z_c - B_z - D_t) \bmod A = 0$ (cf. (1)).

3. Software Rejuvenation

Software rejuvenation is a technique which has been first proposed by Huang et. al. in 1995 [Hua95]. Since this time many papers have been published [Cor11] to characterize and mitigate the Software Aging phenomenon. In long-running operational software, due to software aging, the accumulation of errors leads for example to progressive resource depletion and finally to the crash of the system [Bao05]. Software rejuvenation is a fault reaction technique to prevent that failures in continuously running systems occur. At certain points in the program flow the currently used variables and the corresponding program counter will be stored on a stack. If an error is detected the rejuvenation could switch the service from the active coded channel to the standby coded channel. Consequently the execution of the function could be continued at this point in the program flow using the other channel. This involves stopping the task of one coded channel and switching the task to the other coded channel which was in standby. Due to this environment diversity the transient failures in software could be corrected, because the coded channel which is in standby could be restarted in an internal state with a cleaned environment like the memory. The failure triggered rejuvenation is reactive, so that the action is started after a failure. In addition the software rejuvenation process can also be started automatically due to measurements or calculations which determine the best time window for switching. This means that the action is startet preventive and proactive. The likelihood of successfully completing the current task will be increased by the following steps. The task will be switched periodically from one coded channel to the second coded channel, while restarting the task at a previous checkpoint and in the meantime the rejuvenation process will be started on the currently inactive channel. It must be stressed that the periodical switching does not replace the switching due to an error. This mechanism complements the switching due to an error, so that this case becomes less likely in the system. When errors occur in the system, it is still necessary to trigger the software rejuvenation spontaneously.

Software rejuvenation does not solve the root cause of the failures, consequently the software aging will continue so that the software rejuvenation has to be executed cyclically. Since software aging leads to transient failures in systems, environment diversity can be employed pro-actively to prevent degradation or crashes [Vai99]. Such errors are transient and they are non-deterministic or difficult to characterize. Consequently these errors could occur after a long time without disturbing the proper execution of the application before. Even now, software aging has been observed in specialized software used in high-availability and safety-critical applications [Hua95].

The main aim of SES in the case of rejuvenation is to detect errors reliable. After the detection of an error the switching to the standby unit is triggered and the rejuvenation is started. Beside this the software rejuvenation is triggered periodically, so that after every rejuvenation interval the task is restarted at a clean internal state. Due to this the reliability of the system could be increased.

4. Modelling of the redundant system

Two models are discussed in this paper. The first model is a redundant system without software rejuvenation and the second model is with coded processing and software rejuvenation.

In this paper with software rejuvenation, always the first-level rejuvenation or partial rejuvenation is meant. In the literature there are first-level and two-level rejuvenation models available. First-level rejuvenation stopps only certain tasks of the system and switches them to a redundant system. However the two-level rejuvenation, also called full rejuvenation, could stop all running tasks and restarts the system. With two-level rejuvenation depending on the condition of the software environment the different rejuvenation actions (full or partial) will be triggered [Jin05][Kou05][Xie05]. This full rejuvenation is in the automotive environment for the most systems not possible, because the demanded reaction times are shorter than the time needed for the restart of the whole system. For systems in the automotive environment a certain availability must be guaranteed for the whole driving cycle.

Consequently in this paper with software rejuvenation always the first-level rejuvenation is meant.

4.1. Markov Chain

The Markov chain is a stochastic process and named after Andrey Andreyevich Markov [Hil07]. It consists of a finite number of states and represents the possible transitions from one state to another. In the case of a Markov chain of first order, for each step it does not matter what has been the previous sequence of steps. For determining the next step it is only relevant which is the current state. Due to this fact the process is called "memoryless". This is called the Markov property. The aim is to be able to calculate the probability for the occurrence of certain states in the future.

With Markov chains it is possible to describe any redundant system. The states of the Markov chain correspond to the possible system states, which represent the functional or failure states of the components of the system. At a certain time the system is always in a certain state of the Markov chain. Consequently all failure rates as well as their appearance can be taken into account.

Each state is represented by a circle with a number in it. To improve the readability the states could be additionally named, like “no channel failed”. The transitions between the states are represented by arrows from the destination state to the target state. The arrows are labeled with the transition rate, which represents the rate that the system goes from this state to a certain state.

Electronic systems behave as if they would fail nearly at any time with the same failure rate λ. There is an exception for the early failures and for failures due to aging [Bra12]. Based on these facts the traditional Weibull distribution is unable to model the complete lifetime of electronic systems. Therefore the extended Weibull distribution has been developed which matches such systems with a bathtub-shaped failure rate function. The characteristics of the extended Weibull distribution are, that at the beginning the Weibull distribution is dominating the gradient of the curve, at the end the function is increasing rapidly according to a nominal distribution and in between the failure rate is almost constant which corresponds to exponentially distributed random failures.

For simplification an ideal model is assumed for electronic systems [Bör04]. In this ideal model the early and late failures are neglected.

4.2. System with Cold Redundancy without SES

A system with two channels is considered. One channel is active and one is in cold standby. Two channels are running on a multi core system where for example different ASIL levels are partitioned on different cores. The important point is that one channel is active and one is in standby. When the active channel fails, the service is switched from the active channel to the standby channel.

As shown in Fig. 2 we assume, that the active channel fails with constant failure rate λ_1. At the beginning no channel failed, so that the system is in state 1, where one channel is active and the other channel is in cold standby. At this point no automatic switching at a certain time is implemented. Consequently only in the case of an error the system switches from the active channel to the standby channel and enters state 2. The system enters state 3 with failure rate λ_1, if another error occures. In state 3 the system must enter the error state.

There is also the possibility that a failure occurs in both channels. In this case with rate λ_2 starting from state 1 the system must enter state 3 directly without traversing state 2.

Consequently there is no differentiation in the error state. This means that the error state will be entered for example after a common cause failure, just as well after the occurrence of transient errors due to electro migration.

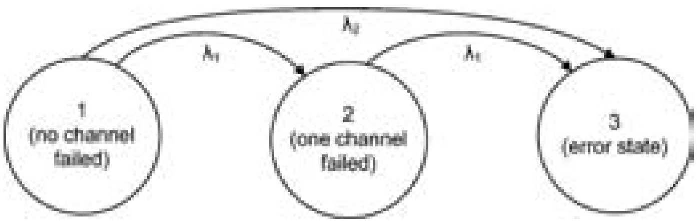

Figure 2: **system with cold redundancy (no rejuvenation)**

4.3. Software Rejuvenation on a System with Cold Redundancy with SES

Again a system with two channels is considered. In this model each channel is coded with SES wherupon one channel is active and one is in cold standby. The system switches from the active channel to the standby channel in the case of an error as well as cyclically at the time when the cyclic rejuvenation is started. Consequently there is an automatic switching implemented at a certain time. As shown in Fig. 3, at the beginning no channel failed, so that the system is in state 1, where one channel is active and the other channel is in cold standby.

In the case of an error the system switches from the active channel to the standby channel and enters state 2. The system will enter state 4 where the rejuvenation will be started or the system will enter state 3, if another error occures during the checks of the failed channel. In state 3 the system must enter the error state.

Additionally the rejuvenation process will be triggered cyclic. For the cyclic rejuvenation process the system switches from the active channel to the standby channel, so that the rejuvenation process could be started for the channel, which was the active channel before and enters state 5. After the rejuvenation process, the coded channel is available in cold standby again.

Compared to the previous model without sofware rejuvenation, also the case where software rejuvenation can fail has to be considered. If the rejuvenation process (triggered due to an error or cyclically) is not completed properly or is performed improperly, the state 3 will be entered to bring the system into the error state. This is the result, because the failed rejuvenation is a failure state, which indicates an abnormal function. The consequence of this is, that the system fails in the same manner as before, if a failure occurs on the active channel during the software rejuvenation process. Because of the ongoing software rejuvenation process, no switching to the standby channel is possible.

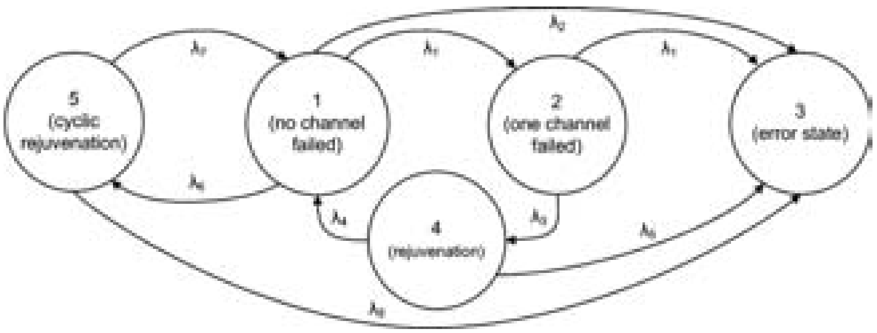

Figure 3: system with coded channel in cold redundancy and rejuvenation

5. Continuous Time Markov Chain (CTMC)

With the help of Markov-models it is possible to calculate reliability values and safety parameters like point-availability, availability, MTTF and reliability. The rates for the state transitions can be specified for each transition from one state to another as well as the probability that the system remains in the state. Constant failure rates are assumed and an exponential distribution. Thus the transition rates are always $\lambda \geq 0$.

5.1. Introduction

Normally the reliability problems are concerned with systems which consist of a number of discrete and identifiable states and continuous in time. A stochastic chain with a continuous parameter space is called markov chain iff. for all $n \geq 0$, each sequence $t_0 < t_1 < ... < t_n < t$:

$$\begin{aligned} &P(X(t) = x \mid X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, ..., X(t_0) = x_0) \\ &= P(X(t) = x \mid X(t_n) = x_n) \end{aligned} \tag{2}$$

The transition matrix Q represents the transition rates for all n states:

$$Q = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1n} \\ q_{21} & q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ q_{n1} & q_{n2} & \cdots & q_{nn} \end{bmatrix} = \begin{bmatrix} 1-A_1 & \lambda_{12} & \cdots & \lambda_{1n} \\ \lambda_{21} & 1-A_2 & \cdots & \lambda_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \cdots & 1-A_n \end{bmatrix} \tag{3}$$

The row sum has to be 1. If there is no transition from one state to another the transition rate is 0. The elements of the matrix described with q_{ij} represent the transition rate from state *i* to state *j* where $i, j \in [1...n]$. Therefore each row represents all transitions from state i to another state and each column represents all transitions from another state to state *j*.

For example the matrix *Q* could be subtracted from the identity matrix *I* to get the matrix *M*:

$$M = I - Q = \begin{bmatrix} A_1 & -\lambda_{12} & \cdots & -\lambda_{1n} \\ -\lambda_{21} & A_2 & \ddots & -\lambda_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ -\lambda_{n1} & -\lambda_{n2} & \cdots & A_n \end{bmatrix} \tag{4}$$

Where the exit rate A_i of state *i* is defined as:

$$A_i = \sum_{k=1, k\neq i}^{n} \lambda_{ik} \tag{5}$$

5.2. System with Cold Redundancy without SES

Based on the Markov chain Fig. 2 defined in chapter 4.2 in combination with (3) and (5) the matrix *Q* could be defined for a system with cold redundancy:

$$Q = \begin{bmatrix} 1-(\lambda_1+\lambda_2) & \lambda_1 & \lambda_2 \\ 0 & 1-\lambda_1 & \lambda_1 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The matrix *G* contains only of state 1 and 2, because the system is fully operational in these states.

$$G = \begin{bmatrix} 1-(\lambda_1+\lambda_2) & \lambda_1 \\ 0 & 1-\lambda_1 \end{bmatrix} \tag{7}$$

For the mean time to failure (MTTF) calculation, which measures the average time to failures, the matrix *M* is calculated:

$$M = I - G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1-(\lambda_1+\lambda_2) & \lambda_1 \\ 0 & 1-\lambda_1 \end{bmatrix} = \begin{bmatrix} \lambda_1+\lambda_2 & -\lambda_1 \\ 0 & \lambda_1 \end{bmatrix} \tag{8}$$

With this matrix *M* the matrix *N* could be calculated:

$$N = [M]^{-1} = \begin{bmatrix} \lambda_1+\lambda_2 & -\lambda_1 \\ 0 & \lambda_1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{\lambda_1+\lambda_2} & \frac{1}{\lambda_1+\lambda_2} \\ 0 & \frac{1}{\lambda_1} \end{bmatrix} \tag{9}$$

For the calculation of MTTF value, the first row of the matrix *N* has to be summed up:

$$MTTF = \frac{1}{\lambda_1+\lambda_2} + \frac{1}{\lambda_1+\lambda_2} \tag{10}$$

The following values are assumed:

$$\lambda_1 = 10^{-7}\frac{1}{h} \tag{11}$$

$$\lambda_2 = \beta \cdot \lambda_1 = 0.05 \cdot 10^{-7}\frac{1}{h} \tag{12}$$

For the determination of the transition rate λ_1 the ISO 26262 [ISO11] was used. In the ISO 26262 the random hardware failure rate of ASIL B and ASIL C systems is defined as $10^{-7}\ h^{-1}$. According to the IEC 61508 part 6 [IEC00] a β-factor is used, to calculate the rate λ_2. This is the rate of a failure of the safety-related system due to a common cause failure. For the system without SES the given β-factor for redundant systems with poor diagnostic tests has been choosen from IEC 61508.

Consequently with equation 10 to 12 the MTTF could be calculated:

$$MTTF \approx 2{,}170y \tag{13}$$

5.3. Software Rejuvenation on a System with Cold Redundancy with SES

Again based on the Markov chain Fig. 3 defined in chapter 4.3 in combination with (3) and (5) the matrix Q could be defined for a system with cold redundancy and software rejuvenation:

$$Q=\begin{bmatrix} 1-(\lambda_1+\lambda_2+\lambda_6) & \lambda_1 & \lambda_2 & 0 & \lambda_6 \\ 0 & 1-(\lambda_1+\lambda_3) & \lambda_1 & \lambda_3 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \lambda_4 & 0 & \lambda_5 & 1-(\lambda_4+\lambda_5) & 0 \\ \lambda_7 & 0 & \lambda_8 & 0 & 1-(\lambda_7+\lambda_8) \end{bmatrix} \tag{14}$$

The matrix G contains only of state 1, 2, 4 and 5, because in these states the system is fully operational.

$$G=\begin{bmatrix} 1-(\lambda_1+\lambda_2+\lambda_6) & \lambda_1 & 0 & \lambda_6 \\ 0 & 1-(\lambda_1+\lambda_3) & \lambda_3 & 0 \\ \lambda_4 & 0 & 1-(\lambda_4+\lambda_5) & 0 \\ \lambda_7 & 0 & 0 & 1-(\lambda_7+\lambda_8) \end{bmatrix} \tag{15}$$

For the mean time to failure (MTTF) calculation, which measures the average time to failures, the matrix M is calculated:

$$M=I-Q= \tag{16}$$

$$=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}-\begin{bmatrix} 1-(\lambda_1+\lambda_2+\lambda_6) & \lambda_1 & 0 & \lambda_6 \\ 0 & 1-(\lambda_1+\lambda_3) & \lambda_3 & 0 \\ \lambda_4 & 0 & 1-(\lambda_4+\lambda_5) & 0 \\ \lambda_7 & 0 & 0 & 1-(\lambda_7+\lambda_8) \end{bmatrix}=$$

$$=\begin{bmatrix} (\lambda_1+\lambda_2+\lambda_6) & -\lambda_1 & 0 & -\lambda_6 \\ 0 & (\lambda_1+\lambda_3) & -\lambda_3 & 0 \\ -\lambda_4 & 0 & (\lambda_4+\lambda_5) & 0 \\ -\lambda_7 & 0 & 0 & (\lambda_7+\lambda_8) \end{bmatrix}$$

With this matrix *M* the matrix *N* could be calculated:

$$N=[M]^{-1}=\begin{bmatrix}(\lambda_1+\lambda_2+\lambda_6) & -\lambda_1 & 0 & -\lambda_6\\ 0 & (\lambda_1+\lambda_3) & -\lambda_3 & 0\\ -\lambda_4 & 0 & (\lambda_4+\lambda_5) & 0\\ -\lambda_7 & 0 & 0 & (\lambda_7+\lambda_8)\end{bmatrix}^{-1}= \tag{17}$$

$$=\det(M)\cdot\begin{bmatrix}(\lambda_1+\lambda_3)(\lambda_4+\lambda_5)(\lambda_7+\lambda_8) & \lambda_1(\lambda_4+\lambda_5)(\lambda_7+\lambda_8)\\ \lambda_3\lambda_4(\lambda_7+\lambda_8) & (\lambda_1+\lambda_2+\lambda_6)(\lambda_4+\lambda_5)(\lambda_7+\lambda_8)-\lambda_6(\lambda_4+\lambda_5)\lambda_7\\ (\lambda_1+\lambda_3)\lambda_4(\lambda_7+\lambda_8) & \lambda_1\lambda_4(\lambda_7+\lambda_8)\\ (\lambda_1+\lambda_3)(\lambda_4+\lambda_5)\lambda_7 & \lambda_1(\lambda_4+\lambda_5)\lambda_7\end{bmatrix}\cdots$$

$$\cdots\begin{matrix}\lambda_1\lambda_3(\lambda_7+\lambda_8) & \lambda_6(\lambda_1+\lambda_3)(\lambda_4+\lambda_5)\\ -\lambda_6\lambda_3\lambda_7+(\lambda_1+\lambda_2+\lambda_6)\lambda_3(\lambda_7+\lambda_8) & \lambda_6\lambda_3\lambda_4\\ (\lambda_1+\lambda_2+\lambda_6)(\lambda_1+\lambda_3)(\lambda_7+\lambda_8)-\lambda_6(\lambda_1+\lambda_3)\lambda_7 & \lambda_6(\lambda_1+\lambda_3)\lambda_4\\ \lambda_1\lambda_3\lambda_7 & (\lambda_1+\lambda_2+\lambda_6)(\lambda_1+\lambda_3)(\lambda_4+\lambda_5)-\lambda_1\lambda_3\lambda_4\end{matrix}\Bigg]$$

$$\det(M)=\frac{1}{(\lambda_1+\lambda_2+\lambda_6)(\lambda_1+\lambda_3)(\lambda_4+\lambda_5)(\lambda_7+\lambda_8)-\lambda_1\lambda_3\lambda_4(\lambda_7+\lambda_8)-\lambda_6(\lambda_1+\lambda_3)(\lambda_4+\lambda_5)\lambda_7}$$

For the calculation of MTTF value, the first row of the matrix *N* has to be summed up:

$$MTTF=\frac{1}{\det(M)}\cdot((\lambda_1+\lambda_3)(\lambda_4+\lambda_5)(\lambda_7+\lambda_8)+\lambda_1(\lambda_4+\lambda_5)(\lambda_7+\lambda_8)+ \\ +\lambda_1\lambda_3(\lambda_7+\lambda_8)+\lambda_6(\lambda_1+\lambda_3)(\lambda_4+\lambda_5)) \tag{18}$$

The following values are assumed:

$$\lambda_1=\frac{1}{31541}\cdot 10^{-7}\frac{1}{h} \tag{19}$$

$$\lambda_2=\beta\cdot\lambda_1=\left(\frac{0.005}{2}+\frac{0.02}{2}\right)\cdot 10^{-7}\frac{1}{h} \tag{20}$$

$$\lambda_3=\frac{1}{\tau_{\mathrm{Re}\,pair}}=\frac{1}{\tau_{Diagnosis}}=\frac{1}{100}\cdot\frac{1}{3600000}\frac{1}{h} \tag{21}$$

$$\lambda_4 = \frac{1}{\tau_{\mathrm{Re}\,juvenation}} = \frac{1}{300} \cdot \frac{1}{3600000} \frac{1}{h} \tag{22}$$

$$\lambda_5 = 10^{-7} \frac{1}{h} \tag{23}$$

$$\lambda_6 = \frac{1}{\tau_{\mathrm{Re}\,juvenationCyclic}} = \frac{1}{0.25} \frac{1}{h} \tag{24}$$

$$\lambda_7 = \frac{1}{\tau_{\mathrm{Re}\,juvenation}} = \frac{1}{300} \cdot \frac{1}{3600000} \frac{1}{h} \tag{25}$$

$$\lambda_8 = 10^{-7} \frac{1}{h} \tag{26}$$

According to the IEC 61508 part 6 [IEC00] a β-factor is used, to calculate the rate λ_2. This is the rate of a failure of the safety-related system due to a common cause failure. For the system with SES the given β-factor for diverse systems with good diagnostic tests has been choosen from IEC 61508 with the ratio of dangerous undetectable and dangerous detectable from [Bör04].

Consequently with equation 18 to 26 the MTTF could be calculated:

$$MTTF \approx 3{,}420{,}000 \text{ y} \tag{27}$$

6. Conclusion

In this paper a software rejuvenation model is proposed, in which SES is supplemented with the partial rejuvenation. Due to the SES more failures could be detected, so that it gets more likely, that the system is working in state 1. Nevertheless the software rejuvenation may not be successful under some circumstances. This leads to a failure of the system, but in total the reliability gets higher due to the fact that former undetected failures will be eliminated during the rejuvenation process.

As shown before with the help of the Markov modell of the two examplary systems it was possible to point out a great improvement of the system reliability by the calculation of a CTMC. Without coded processing and software rejuvenation the assumed system will have a MTTF of 2,170 years (cf. (13)). This means, that one system will fail every 2,170 years, which sounds quite good. But if you assume, that this system is running in one million cars, then about 460 cars will fail every year and this sounds no longer as good as before. With coded processing and with software rejuvenation the assumed system will have a MTTF of 3,420,000 years (cf. (27)). In the point of view like before, this also means that only one car will fail every three to four years, if one million cars are running with the system safeguarded by SES in combination with software rejuvenation.

Of course the calculation of the MTTF could be done also for ASIL A and ASIL D systems. In Tab. 1 again the hardware failure rates defined in the ISO 26262 [ISO11] are used. It could be seen that the failure rate of the choosen hardware has a direct influence on the MTTF of the system regardless of whether SES is used or not.

	ASIL A	**ASIL B/C**	**ASIL D**
failure rate	10^{-6} h^{-1}	10^{-7} h^{-1}	10^{-8} h^{-1}
MTTF without SES	217 y	2,170 y	21,700 y
MTTF with SES	342,000 y	3,420,000 y	34,200,000 y

Table 1: **Comparisson of MTTF of ASIL A to D systems according ISO 26262.**

A further step would be to do a Semi-Markov calculation using the extended Weibull-function instead of the constant lambda values to be able to take the whole lifetime of an electronic system into account. With a Semi-Markov modell it would be even possible to calculate the optimal rejuvenation interval, which will improve the system's availability further.

Bibliography

[Bao05] Bao, Y.; Sun, X.; Trivedi, K.S.: A workload-based analysis of software aging and rejuvenation. IEEE Transactions on Reliability, Vol. 54, No. 3, pages 541-548, 2005

[Bör04] Börcsök, J.: Electronic Safety Systems: Hardware Concepts, Models, and Calculations. Hüthig, ISBN 978-3778529447, 2004

[Bra12] Braun, J.; Mottok, J.; Miedl, C.; Geyer, D.; Minas M.: Capability of single hardware channel for automotive safety applications according to ISO 26262. In Proceedings of the International Conference on Applied Electronics 2012 (AE2012) in Pilsen, September 2012.

[Cas01] Castelli, V.; Harper, R. E.; Heidelberger, P.; Hunter, S.W.; Trivedi, K.S.; Vaidyanathan, K.; Zeggert, W.P.: Proactive management of software aging. IBM Journal of Research & Development, Volume 45 Issue 2, pages 311-332, March 2001

[Cor11] Cotroneo, D.; Natella, R.; Pietrantuono, R.; Russo, S.: Software Aging and Rejuvenation: Where we are and where we are going. The 3rd International Workshop on Software Aging and Rejuvenation (WoSAR 2011), 2011

[Doh00n] Dohi, T.; Goseva-Popstojanova, K.; Trivedi, K.S.: Analysis of software cost models with rejuvenation. In Proceedings of the 5th IEEE International Symposium on High Assurance Systems Engineering, pages 25-34, Albuquerque, New Mexico, November 2000

[Doh00d] Dohi, T.; Goseva-Popstojanova, K.; Trivedi, K.S.: Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. In Proceedings of the 2000 Pacific Rim International Symposium on Dependable Computing (PRDC 2000), pages 77-84, Los Angeles, CA, December 2000

[For89] Forin, P.: Vital Coded Microprocessor Principles and Application for Various Transit Systems. IFAC Control, Computers, Communications, pages 79–84, Paris, 1989

[Gar95] Garg, S.; Puliafito, A.; Telek, M.; Trivedi, K.S.: Analysis of software rejuvenation using Markov regenerative stochastic Petri Nets. In Proceedings of the Sixth International Symposium on Software Reliability Engineering, pages 180-187, Toulouse, France, October 1995

[Gar98] Garg, S.; van Moorsel, A.; Vaidyanathan, K.; Trivedi, K.S.: A methodology for detection and estimation of software aging. In Proceeding of the International Symposium on Software Reliability Engineering (ISSRE 1998), pages 283-292, November 1998

[Hil07] von Hilgers, P.; Velminski, W.: Andrej A. Markov: Berechenbare Künste. Mathematik, Poesie, Moderne. Diaphanes, ISBN 978-3935300698, 2007

[Hua95] Huang, Y.; Kintala, C.; Kolettis, N.; Fulton, N.D.: Software Rejuvenation: Analysis, Module and Applications. In Proceedings of the 25th International Symposium on Fault Tolerant Computer Systems, pages 381-390, Pasadena, California, June 1995

[IEC00] CEI/IEC 61508-6:2000: Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 6: Guidelines on the application of lEG 61508-2 and lEC 61508-3

[ISO11] ISO 26262: Road vehicles - Functional safety, International Organization for Standardization, 2011

[Jin05] Jing, Y.; Jian, X.; Xue-lLong, Z.; Feng-Yu, L.; Modeling and availability analysis of nested software rejuvenation policy. In Proceedings of IEEE Intl. Conf. on Systems, Man and Cybernetics, Vol. 1, pages 34-38, 2005

[Kou05] Koutras, V.P.; Platis, A.N.: Optimizing the amount of the resources on a computer system using software rejuvenation. In Proceedings of European Safety and Reliability Conference, pages 1187-1192, 2005

[Kou10] Koutras, V.P.; Platis, A.N.: Semi-Markov performance modelling of a redundant system with partial, full and failed rejuvenation. International Journal of Critical Computer-Based Systems 2010, Vol.1 No.1/2/3, pages 59-85, 2010

[Li02] Li, L.; Vaidyanathan, K.; Trivedi, K.S.: An approach for estimation of software aging in a web server. In Proceedings of the 2002 International Symposium on Empirical Software Engineering, pages 91-102, Nara, Japan, October 2002

[Mot12] Mottok, J.; Schiller, F.; Zeitler, T.: Embedded Systems – Theory and Design Methodology. Chapter 2: Safely Embedded Software for State Machines in Automotive Applications, InTech, March 2012

[Par94] Parnas, D.L.: Software aging. In Proceedings of 16th International Conference on Software Engineering, pages 279-287, ACM Press, New York, 1994

[Sar09] Saravakos, P.K.; Gravvanis, G.A.; Koutras, V.P.; Platis, A.N.: A Comprehensive Approach to Software Aging and Rejuvenation on a Single Node Software System. Proceedings of The 9th Hellenic European Research on Computer Mathematics & its Applications Conference (HERCMA 2009), 2009

[Sch06] Schiller, F.; Mattes, T.: An Efficient Method to Evaluate CRC-Polynomials for Safety-Critical Industrial Communication. Journal Of Applied Computer Science, Vol. 14, No. 1, 2006

[She03] Shereshevsky, M.; Crowell, J.; Cukic, B.; Gandikota, V.; Liu, Y.: Software aging and multifractality of memory resources. In Proceedings of the 2003 International Conference on Dependable Systems and Networks, pages 721-730, San Francisco, June 2003

[Vai99] Vaidyanathan, K.; Trivedi, K.S.: A measurement-based model for estimation of resource exhaustion in operational software systems. In Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE 1999), pages 84-93, Boca Raton, Florida, November 1999

[Vai01] Vaidyanathan, K.; Harper, R.E.; Hunter, S.W.; Trivedi, K.S.: Analysis and implementation of software rejuvenation in cluster systems. In Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer System, pages 62-71, Cambridge, Massachusetts, USA, June 2001

[Xie05] Xie, W.; Yiguang, H.; Trivedi, K.S.: Analysis of a two-level software rejuvenation policy. Reliability Engineering and System Safety, Vol. 87, No. 1, pages 13-22, 2005