SERD: A Simulation Framework for Estimation of System Level Reliability Degradation

Saurav Kumar Ghosh
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, WB 721302
Email: saurav.kumar.ghosh@cse.iitkgp.ernet.in

Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, WB 721302
Email: soumya@cse.iitkgp.ernet.in

***Abstract*—Development of highly reliable embedded control systems is typically performed following the model driven engineering paradigm. Such systems involve software controlled interaction of mechanical subsystems. The *aging* of the overall system depends on the physical aging or reliability decay of the underlying mechanical components. The reliability of such components degrade according to their rate of usage which again is governed by the software control logic and input environment. Such dependencies of component reliabilities make the problem of deriving system level reliability degradation using exact methods combinatorially intractable.**

Given the fact that model driven system design advocates the usage of initial high level system models, methods for early stage lifetime reliability and reliability degradation estimation based on such initial models should definitely aid in robust high assurance engineering of such software controlled physical systems.

The present work proposes SERD, a lightweight, scalable simulation framework for embedded control systems. It can accommodate active as well as quiescent reliability decay rates of underlying mechanical components. It uses path based reliability modeling to estimate the reliability degradation of component based systems that are controlled by software logic. Its efficacy is further demonstrated using a thorough case study.

I. Introduction

With the increasing variety and complexity of software based control in embedded systems, the development paradigm for embedded software is moving towards model based design and validation. System reliability is regarded as a first class design criteria for the development of such systems. Hence, estimation of the lifetime reliability of such systems in the early stages of design often becomes a crucial aspect for analysis. A distinguishing feature of such control system is the activation of mechanical components guided by control logic implemented in software. Hence, failures often stem from permanent faults in such underlying components. Reliability degradation of hardware systems due to wear out and aging can be modeled with probability distributions like Weibull [1]. However, existing works on architecture based reliability analysis [2], [3] of component based softwares cannot be combined directly with Weibull decay rates of underlying hardware to estimate the overall reliability degradation for such software controlled hardware systems.

The present work proposes SERD, a usage aware simulation based approach for estimating reliability degradation of software controlled mechanical systems over finite time horizons. For brevity in presentation, we consider a single embedded software control loop running on a fully reliable electronic control unit (ECU), sensing the plant state periodically (as governed by the sampling rate) and activating a set of mechanical components as shown in Fig. 1.

This work was supported by a TCS Research Fellowship

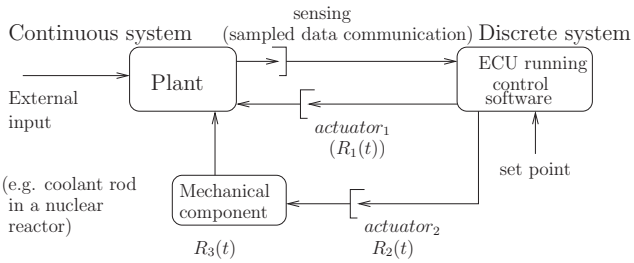


Fig. 1: Embedded control system with reliability annotation

The *control action* as computed by the software comprises a set of activation signals which dictate the functioning of a set of actuators. While some actuators are directly connected to the plant controlling rate of certain dynamic quantities (e.g. process parameters), some activate mechanical systems that effect the trajectory of evolution for the plant. As an example, consider coolant rods which can be actuated to move in and out of nuclear reactors. From a reliability perspective, we consider the set K of actuators of second type and mechanical systems.

The failure of mechanical components and actuators are assumed to be *permanent* in nature. The reliability ($R(t)$ or simply R when time is implicit) of a (mechanical) component is defined as the probability that it will perform a required function without failure at time t. Reliability of any mechanical hardware decreases with time and this decay is accelerated depending on the amount of its active usage. For example, the reliability decay rate of the brake-shoe of a braking system varies depending on how much it is used. In order to efficiently model such usage based degradation of reliability, two separate rates of decay, namely *active decay rate* and *passive decay rate* are usually considered for each individual component. SERD uses the Weibull [1] reliability model to capture the active as well as passive reliability decay of the components. Weibull distribution states that the reliability R of a component at time t is given by $R(t) = Exp\{-(t/\alpha)^{\beta}\}$ where α and β are

978-3-9815370-8-6/17/$31.00

known as the scale and shape parameters respectively. Following Weibull distribution, failure rate is a function of time, given by $\lambda(t) = \frac{\beta}{\alpha}(\frac{t}{\alpha})^{\beta-1}$. The active reliability decay rates have higher shape and lower scale parameters compared to the passive rates to imply higher rates of decay. However, this distinction among reliability decay rates make the problem of system level reliability degradation estimation more complex. Given our system specification model and assumptions, the formal problem statement is provided as follows.

Formal Problem Statement: *Considering an embedded control system comprising a plant P, a set of mechanical components K and a controller implemented as a program σ running on an ECU along with some input activation pattern, if each component in K is characterized by an active as well as passive reliability degradation (aging) function, what are the component level reliabilities as well as the system level reliability at the end of some finite time horizon T ?*

The input activation pattern provides a timed sequence of activation inputs for the system. For a periodic control system, such a pattern signifies external control commands, e.g., braking event in ABS controller, setting target water level in a steam boiler etc.

Existing reliability modeling approaches [4] and tools like BlockSim [5] attempt to model system level reliability degradation using Reliability Block Diagrams (RBD) and Fault Tree Diagrams (FTD) [5]. RBDs are used to model systems that can be expressed as a series and/or parallel combination of its constituent components. FTDs and RBDs have similar modeling capabilities; the only difference being that RBDs look at success combinations while FTDs look at failure combinations. Both RBDs and FTDs model systems that activate all components in every execution instance. Hence even though these tools model both active and passive reliability decay, the application of such decay functions is simultaneous over all system components. However, for systems having multiple input dependent execution paths (like ours), such models are not useful and path based reliability modeling is required. For path based models, as employed in our SERD tool, during every execution instance, only those components which are part of the execution path shall exhibit active reliability decay while others follow passive decay rates. The salient features of the SERD toolflow can be summarized as follows.

- SERD estimates component level reliability decay based on environmental inputs while factoring in active and passive reliability decay rates. As a result, it can provide system level as well as component level reliability degradation estimates. Being computationally lightweight, SERD is scalable.
- Using path based models, SERD generates component execution sequences following input probability distributions. Hence, it can accommodate Weibull and other time varying failure models.
- SERD can validate system design objectives expressed as temporal properties defined on system and component level reliability values.

Verifying such temporal reliability properties can be useful in many contexts. For example, checking whether the reliability of a component shall fall below a certain value within some designated future time instant is important for defining maintenance schedules of complex systems. SERD can check whether a maintenance plan can satisfy a reliability criteria and accordingly help in deciding component replacement schedules for complex systems. In addition to providing a simple yes/no answer, SERD is also able to identify the time point in simulation at which such a property gets violated.

II. Methodology

The novelty of SERD can be comprehended from the fact that it supports path based reliability modeling and simulation of embedded control systems. SERD computes component level as well as system level reliability while taking into account both active and passive decay rates. The overall toolflow is summarized in Fig. 2.

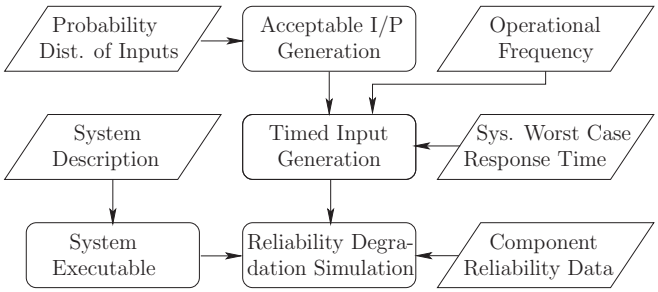


Fig. 2: Tool-Flow of SERD

Given the behavioral description of a system, its operational frequency, component reliability decay rates and the probability distribution of system inputs, SERD calculates the reliability decay of individual components based on their usage by simulating the system execution for a given time horizon. The major steps in the simulation methodology are discussed as follows.

A. Acceptable Input Generation

SERD generates system inputs by sampling their respective probability distributions. The variables in the system description may be of two types namely program variables and input variables. Program variables are used to hold internal computational results and value assignments. Input variables are used to represent entities whose values are sensed from environment. The set of such input variables for a system is denoted by $\mathcal{V}$. SERD assumes that it has been provided with probability distributions of the input variables and relational constraints (if any) among them together termed as the operational profile of the system defined as follows.

Definition 1. *The **operational profile** $(\mathcal{F}, \mathcal{R})$ of a system consists of the following elements.*

1) The set of probability density functions $\mathcal{F}$ considering all the input variables of a system is denoted as $\mathcal{F} = \{f_1, f_2, \cdots, f_{|\mathcal{V}|}\}$ where f_i denotes the probability density function of v_i for any input variable $v_i \in \mathcal{V}$.

2) *The set of relational constraints $\mathcal{R}$ defined over $\mathcal{V}$ that every possible set of inputs must additionally satisfy apart from the independent probability distributions.* □

The set of constraints $\mathcal{R}$ aid in the generation of correlated inputs and compensates for the lack of a joint probability distribution on $\mathcal{V}$ (such joint distributions are often very difficult to obtain). An example of such constraints can be given as follows. In order to simulate valid inputs for the Antilock Braking System of a car, wheel speed is required to be less than vehicle speed.

The operational profile attempts to mimic the environmental inputs to the system in order to simulate its operations more realistically. Using the operational profile of the system, SERD generates the acceptable system inputs for some target lifespan $\mathcal{T}$ of the system.

Definition 2. *Given a set $\mathcal{V} = \{v_1, \cdots, v_{|\mathcal{V}|}\}$ of input variables with domains $D_1, \cdots, D_{|\mathcal{V}|}$ and a set of relational constraints $\mathcal{R}$ defined over $\mathcal{V}$, a value assignment $q = \langle q_1, \cdots, q_{|\mathcal{V}|}\rangle \in D_1 \times \cdots D_k$ is said to be an **acceptable input** iff $q \vdash \mathcal{R}$, i.e. the assignment satisfies the set of relational constraints.* □

For generating acceptable inputs, SERD uses *inverse transform sampling* to independently sample values of each input variable according to its probability distribution. First the cumulative density function F_i for all the input variables $v_i \in \mathcal{V}$ is computed. Then for each input variable $v_i \in \mathcal{V}$, a random number x_i between 0 and 1 is generated which acts as the current choice of cumulative probability of v_i. A candidate value q_i of v_i is then calculated as the inverse of the cumulative density function F_i at x_i, i.e. $q_i = F_i^{-1}(x_i)$. In that way, a possible value assignment $q = \langle q_1, q_2, \cdots q_{|\mathcal{V}|}\rangle$ is generated. The value assignment is considered as an acceptable input iff $q \vdash \mathcal{R}$. Otherwise, a new value assignment for q is calculated.

B. Timed Input Generation

In general, system level reliability degradation depends on two primary factors. One of them is the system operating environment which can be modeled by operational profile of a system. Given such profile information it is possible to create input situations for the system over its lifetime. The other important factor contributing to the reliability degradation is the operational frequency of the system. Note that the frequency of execution of a system can have various possible semantics. A time triggered system may operate with the same frequency throughout its lifetime. An event triggered system may *eagerly* process inputs depending on whenever the input arrives. Moreover, there may be systems for which the execution frequency is decided by other governing factors like higher level schedulers, interactions with other subsystems etc. It is often the case that such scheduling uncertainties have computable bounds which can be inferred through static analysis methods. Assuming the recurrent occurrence of such static bounds, the load profile of a system is defined as follows.

Definition 3. *The **interval load profile** l for time t is a 3-tuple $\langle \delta, j, k\rangle$ which denotes that the number of system executions within the time-span δ will lie between a minimum value j and a maximum value k. The **load profile** of a system is a recurrent sequence of interval load profiles denoted by $\mathcal{L} = (l_1 l_2 \cdots l_m)^\omega$, $m \geq 1$, given as a ω-regular string.* □

With this representation of load profile, SERD can handle both periodic as well as sporadic input arrival patterns. Such load profiles can act as natural input specifications and they can also be derived mechanically [6] from input specification techniques like Real Time Calculus (RTC) [7] as done in existing real-time simulation tools like [8].

Given a sequence of acceptable inputs generated as per the operational profile of the system, the activation sequence of the system as per the load profile needs to be generated. Assuming that the system processes inputs sequentially, even though the inputs may arrive following the operational profile, the processing of inputs by the system is actually further constrained by the execution time of the system and the load profile. In case of an input arrival, if the system is busy, the input gets buffered. (Buffer overflow issues are not part of the present work). Buffered inputs are processed as per the load profile and after the system finishes processing the previous input. Let us consider a load profile given as $\mathcal{L} = (l_1 l_2 \cdots l_m)^\omega = (\langle \delta_1, j_1, k_1\rangle \langle \delta_2, j_2, k_2\rangle \cdots \langle \delta_m, j_m, k_m\rangle)^\omega$. As defined earlier, the number of system executions inside the time interval $[t_i, t_{i+1}]$ where $t_i = \delta_1 + \cdots + \delta_{i-1}$ & $t_{i+1} = \delta_1 + \cdots + \delta_i$ shall be in the interval $[j_i, k_i]$. If the least possible reliability degradation is required to be estimated, SERD schedules j_i executions of the system; otherwise it schedules k_i executions in order to estimate the maximum possible degradation. Methods for balanced selection for intermediate values in the range $[j_i, k_i]$ are not addressed in the present work.

Given that most commonly used component decay functions (like Weibull) are parameterized with time, the time stamps at which the system gets activated by a given input sequence needs to be generated. Hence, SERD computes the time instants at which the system consumes each acceptable input for reliability degradation analysis. The process of *timed input generation* is discussed next.

Definition 4. *A **timed input** $\langle q, t\rangle$ is a pair comprising an acceptable input q and the time instant t at which the input is actually processed by the system.* □

For each timed input, an execution time-stamp is paired with an acceptable input. Given a load profile $\mathcal{L} = (l_1 l_2 \cdots l_m)^\omega = (\langle \delta_1, j_1, k_1\rangle \langle \delta_2, j_2, k_2\rangle \cdots \langle \delta_m, j_m, k_m\rangle)^\omega$, inside the time interval $[t_i, t_{i+1}]$ where t_i & t_{i+1} are as defined earlier, s time stamps are generated where $s = j_i$ or k_i. Let the Worst Case Response Time (WCRT) of the system be given by τ. This information is considered as an input to the framework along with the system description. For the first time stamp after t_i, a random value in the range $[t_i, (t_{i+1} - s \times \tau)]$ is generated accounting for the pathological scenario when every s number of consecutive instances of system execution suffer

from worst case timing behavior. In case, $(t_{i+1} - s \times \tau) \leq t_i$, the corresponding interval load profile is reported as inadmissible and s is reduced by the minimum number of events required to preserve a causal chain of system activation. Let the time instant thus generated be t. Thus an admissible time stamp for the next acceptable input shall be in the range $[t, (t_{i+1} - (s-1) \times \tau)]$. Continuing in this manner the required number of time stamps are generated inside the interval $[t_i, t_{i+1}]$.

C. Reliability Degradation Simulation

Reliability degradation simulation of an embedded control system consists in 3 steps, namely, extracting a executable σ from the behavioral description of the system $\mathcal{D}$, simulating the system by running the executable σ with timed inputs, and lastly calculating the reliability degradation of the individual components as well as the entire system using the execution path followed by the system for the corresponding timed input. An **execution path** $\pi = C_1 C_2 \cdots C_{|\pi|}$ ($C_i \in \mathcal{C}$ where $\mathcal{C}$ is the set of components of the system) is a finite sequence of components that are activated when the system executable σ is run with the timed input $\langle q, t \rangle$. Given a controller implemented as a software program σ comprising a set $\mathcal{P} = \pi_1, \pi_2, \cdots, \pi_n$ of execution paths[1], the corresponding reliability model can be perceived as a collection of n such possible execution paths.

Input: System Description $\mathcal{D}$, Operational Profile $(\mathcal{F}, \mathcal{C})$, WCRT τ, Load Profile $\mathcal{L}$, Simulation Time Horizon $\mathcal{T}$
Output: Component Level Reliability Decay Estimation

1: Generate executable σ from description $\mathcal{D}$
2: $i = 1, \delta = time = 0$ ▷ Initialization
3: **while** $time < \mathcal{T}$ **do**
4: Extract interval load profile $l_i = \langle \delta_i, j_i, k_i \rangle$
5: Generate acceptable i/p seq. $q_1 \cdots q_s$; $s = j_i$ or k_i
6: $\delta = \delta + \delta_i$
7: **for all** x = 1 to s **do**
8: $t_x^i = random(time, (\delta - (s - x + 1) \times \tau))$
9: Passive_Decay_of_all_Components($time$, t_x^i)
10: $time = t_x^i$
11: Execution path π = Run σ with input q_x
12: For π, calculate Active and Passive Decay $\forall C \in \mathcal{C}$
13: Update $time$
14: Update frequency of π
15: **end for**
16: Passive_Decay_of_all_Components($time$, δ)
17: $time = \delta$
18: **end while**

Algorithm 1: Reliability Degradation Simulation

Given a system description $\mathcal{D}$, operational profile $(\mathcal{F}, \mathcal{C})$, load profile $\mathcal{L} = (l_1 l_2 \cdots l_m)^\omega$ and target time horizon $\mathcal{T}$, the reliability degradation simulation implemented in SERD in presented in Algorithm 1. The first step of SERD is to manually extract or build a system executable σ from the system description $\mathcal{D}$ (Line 1). We consider σ to a gcc-compatible C program (hence σ can be derived from the actual control software source also). Line 2 initializes the interval load profile counter (denoted by i), interval simulation time (δ) and system simulation time (denoted by $time$) to 1, 0, and 0 respectively. The interval simulation time (δ) keeps track of the end time of the current interval load profile. SERD continues the system simulation until $time \geq \mathcal{T}$. Then for each interval load profile $l_i = \langle \delta_i, j_i, k_i \rangle$ in $\mathcal{L}$, a sequence of acceptable input $\langle q_1^i, \cdots, q_s^i, \rangle (s = j_i \text{ or } k_i)$ is generated for the i^{th} interval load profile (Lines 4 and 5). The interval simulation time (δ) is updated to mark the end if the i^{th} interval load profile (Line 6). Let the processing of the i^{th} interval load profile start from $time = t$. We now generate the time stamp (t_1^i) of the first timed input of the i^{th} interval load profile (Line 8). From $time = t$ up to $time = t + t_1^i$, the reliability of every component $C \in \mathcal{C}$ is decayed following the passive decay rate as the system has not yet started execution (Line 9). Then the system simulation time is updated to $time = t_1^i$ (Line 10) and the system executable σ is run with the input q_1^i in order to synthesize an execution path π (Line 11). When the system is actively executing, exactly one component is active at any instant. For the execution time of the active component, its reliability gets decayed according to its active decay rate while all other component reliabilities get decayed according to their passive decay rates. In this way the reliability decay (both active and passive) for all the components of the system is calculated for the execution path π (Line 12). After every instance of decay calculation (both active and passive) for each component in execution path π, the simulation timing is advanced accordingly. We symbolically represent the total update of system simulation time for execution path π in Line 13. Subsequently the frequency of execution path π is updated (Line 14). At the end of the execution of the 1^{st} timed input of the i^{th} interval load profile, the next time stamp t_2^i is calculated using methods discussed earlier. The simulation time is now advanced up to the start of the next execution instance at t_2^i and the simulation loop (Lines 7-15) repeats until all the x timed inputs for the i^{th} interval load profile is exhausted. After this, all the components of the system are allowed to decay with passive decay rates (Line 16) until the end of the i^{th} interval load profile marked by the interval simulation time (δ). Then the system simulation time is updated to $time = \delta$ to mark the end of the i^{th} interval load profile (Line 17). Similarly, subsequent interval load profiles are extracted and the reliability decay of the underlying components is computed until the system simulation time reaches the time horizon $\mathcal{T}$.

As shown in Algorithm 1, after processing each timed input, the frequency of the corresponding execution path is updated (Line 14). For a system operating over time horizon $\mathcal{T}$, the **weight** w of an execution path π is calculated as the ratio $\eta/\mathcal{N}$ where η is the total number of times the system followed path π and $\mathcal{N}$ is the total number of times σ is executed in $\mathcal{T}$. The **reliability** $R_i(t)$ of an execution path π_i is the product of the reliabilities of its constituent components given

[1]As control software typically contains bounded loops, we always have an integer bounded number of finite depth execution paths.

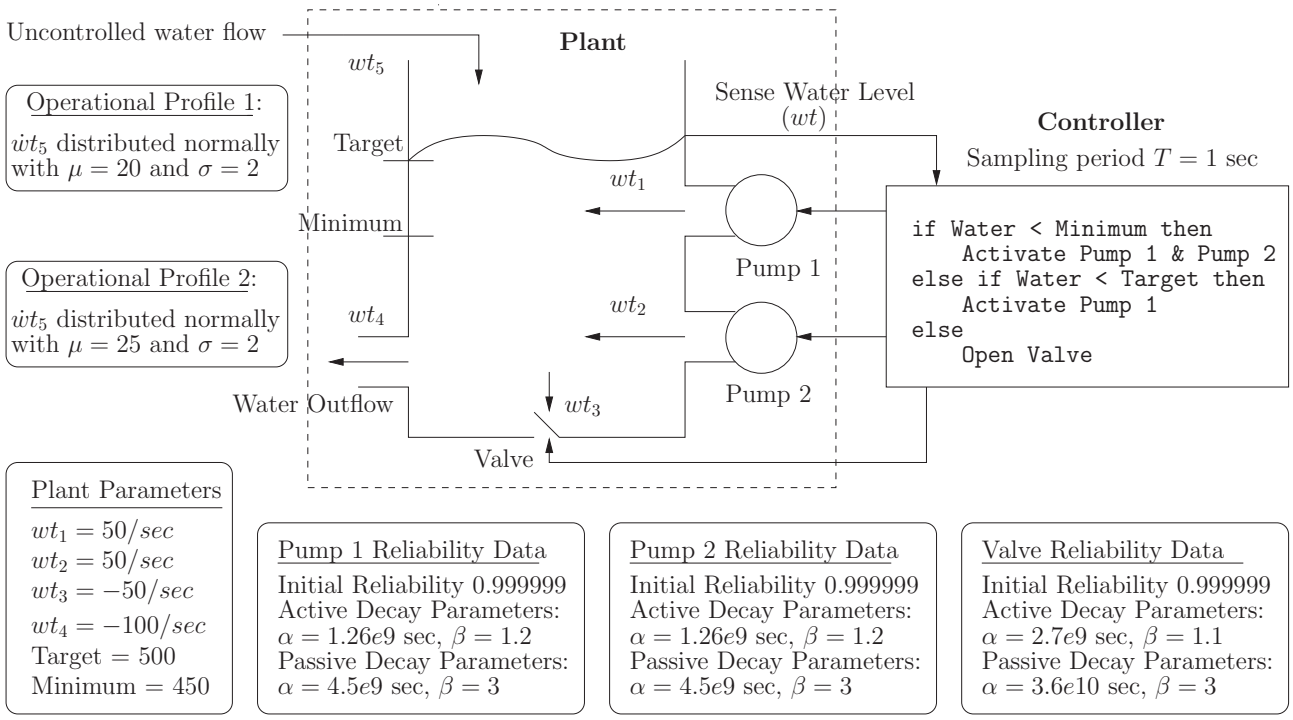


Fig. 3: Water Level Controller System

by $R_i(t) = \prod_{\forall C \in \pi_i} r_C(t)$, where reliability of component C is denoted as $r_C(t)$ at time t. Following general methods of path based reliability estimation [9], the **system level reliability** $R(t)$ is calculated as the weighted average of the reliability of the individual execution paths of the system given by $R(t) = \sum_{i=1}^{n} w_i \times R_i(t)$ where $n = |\mathcal{P}|$.

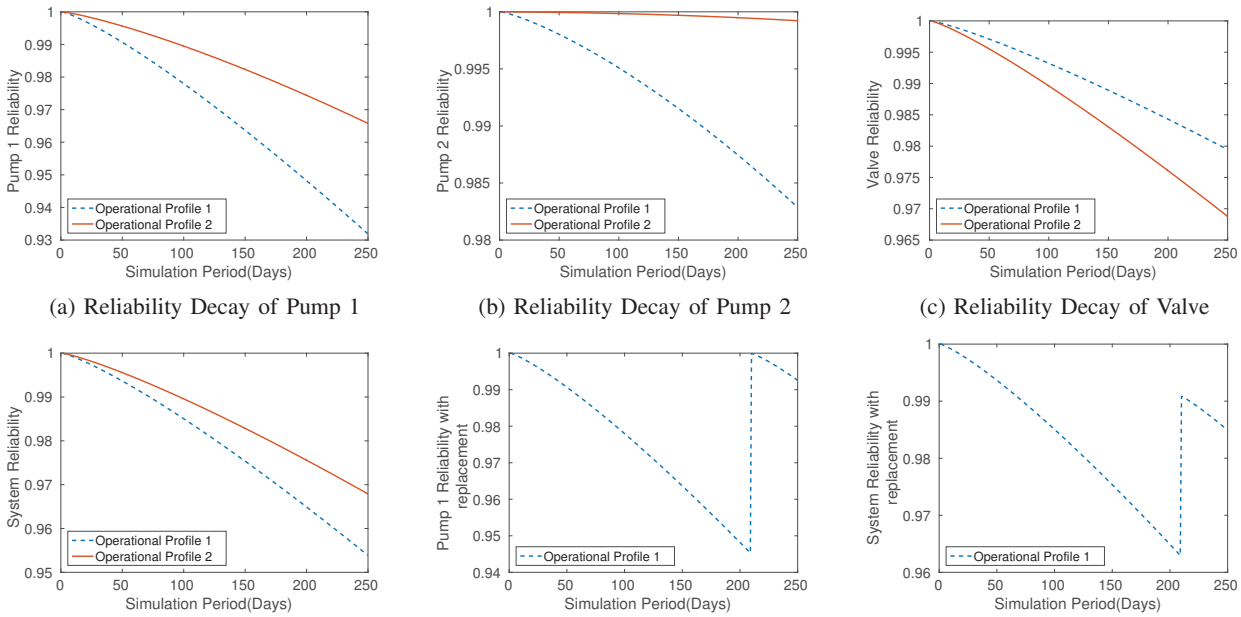
III. Case Study and Observations

We consider an example of Water Level Controller (WLC) [10] for reliability simulation and degradation analysis with SERD. WLC contains two pumps (supporting water inflow rates wt_1 and wt_2), one valve (supporting water outflow rate wt_3), a constant outflow wt_4, and an external uncontrolled inflow wt_5 into the system. As shown in Fig. 3, the control objective of WLC is to keep the water level around a set point 'Target'. It operates by sampling the water level periodically and decides whether to turn the pumps and valve ON or OFF, depending on the current water level wt. Such logic based on-off control can be designed using the Stateflow toolbox of MATLAB [11].

For this case study, the WLC is simulated under two different operational profiles of the rate of the uncontrolled water inflow wt_5 as shown in Fig. 3. The Weibull reliability rates of decay of the components are taken from the Weibull Reliability Database [12].

The plots in Fig. 4 show the overall as well as individual component level reliability decay for simulating the continuous operation of WLC for 6000 hours (250 days) under two operational profiles. The speed and scalability of SERD is demonstrated by the fact that it takes less than 1 minute (on Intel Core-i3-4130 with 4 GB main memory) to complete the WLC system simulation. This is made possible by 1) efficient generation of acceptable inputs based on operational profile using the Boost C++ library, 2) use of C executables for quickly extracting the set of execution paths for a given load profile. In the experimental results (Fig. 4(d)) it is found that for Operational Profile 1, the overall system reliability drops below 0.96. If the WLC needs to maintain its reliability over 0.96 for 6000 hours as a system design criterion, we need to replace some component(s) at some point of time. It is observed that that Pump 1 is the most heavily degraded component under Operational Profile 1(Fig. 4(a)). So in order to reach the reliability target, Pump 1 is replaced after 5000 hours. SERD runs the simulation again and verifies that with this replacement policy, the reliability of WLC stays above 0.96 throughout 6000 hours (Fig. 4(f)). The time for the replacement of Pump 1 is also predicted from the reliability plot outputs of the SERD simulation. SERD is run several times with Operational Profile 1 under this replacement policy to verify that the system meets its design goal.

These results validate the principal claim of this work, that for a software controlled hardware system, the overall reliability degradation of the system as well as the reliability decay of the individual underlying components depend on the inputs to the system. SERD is able to keep track of the component level reliability values for a system throughout the simulation period which enables it to verify overall system level reliability as well as properties defined on component



(a) Reliability Decay of Pump 1

(b) Reliability Decay of Pump 2

(c) Reliability Decay of Valve

(d) Reliability Degradation of Overall WLC System

(e) Reliability Decay of Pump 1 with replacement

(f) Reliability Degradation of Overall WLC System with replacement

Fig. 4: Water Level Controller Case Study

reliability values at any time in the lifetime of the system. As a result SERD proves very useful in ascertaining whether to replace a component at some future time point or design a system with a different reliability option of a component in order to meet some lifetime reliability design criterion.

As one may observe, our approach works for any system with components for which Weibull decay parameters are provided. Manufactures provide such parameters for significant number of mechanical systems used in embedded control. Such parameter values are typically extracted using methods like Reliability Life Data Analysis, Accelerated Life Testing [13]. However, our method is also generic enough to account for other methods of reliability aging function specification like exponential and lognormal distribution.

IV. Conclusion

This paper presents a novel simulation framework for reliability degradation estimation of complex component based systems. SERD is able to calculate the reliable life of a system as well as verify suitable component replacement policy to meet some overall lifetime reliability given as a system design criterion. The efficacy of SERD in simulating complex embedded control systems is illustrated with the case study. Being lightweight, scalable and highly functional, SERD should certainly prove itself useful for the design and analysis of reliable systems. In safety critical control systems, mechanical components like actuators are often implemented with redundancy. Given a target reliability, decisions like the amount of redundancy required in such systems can be influenced by model driven lifetime reliability degradation estimates generated by our approach [14].

References

[1] W. R. Wessels, "Use of the weibull versus exponential to model part reliability," in *Annual Reliability and Maintainability Symposium*. IEEE, 2007, pp. 131–135.

[2] B. Littlewood, "Software reliability model for modular program structure," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 241–246, 1979.

[3] R. C. Cheung, "A user-oriented software reliability model," *IEEE Transactions on Software Engineering*, vol. 6, no. 2, pp. 118–125, 1980.

[4] L. Ding, H. Wang, K. Kang, and K. Wang, "A novel method for sil verification based on system degradation using reliability block diagram," *Reliability Engineering & System Safety*, vol. 132, pp. 36–45, 2014.

[5] R. Corporation, *BlockSim: System Reliability and Maintainability Analysis Software Tool*, ReliaSoft Corporation. [Online]. Available: http://www.reliasoft.com/BlockSim/index.html

[6] M. Moy and K. Altisen, "Arrival curves for real-time calculus: the causality problem and its solutions," in *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2010, pp. 358–372.

[7] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *International Symposium on Circuits and Systems (ISCAS)*, vol. 4. IEEE, 2000, pp. 101–104.

[8] S. Anssi, K. Albers, M. Dörfel, and S. Gérard, "chronval/chronsim: A tool suite for timing verification of auto-motive applications," *Proc. Embedded Real-Time Software and Systems, ERTS*, 2012.

[9] S. M. Yacoub, B. Cukic, and H. H. Ammar, "Scenario-based reliability analysis of component-based software," in *Proceedings of the 10th International Symposium on Software Reliability Engineering*. IEEE, 1999, pp. 22–31.

[10] K. Ogata and Y. Yang, *Modern control engineering*. Prentice-Hall Englewood Cliffs, NJ, 1970.

[11] *MATLAB and Stateflow Toolbox*, The MathWorks Inc., Natick, Massachusetts, United States, Release R2015b.

[12] P. Barringer, *Weibull Database*, Barringer & Associates, Inc. [Online]. Available: http://www.barringer1.com/wdbase.htm

[13] H. Rinne, *The Weibull distribution: a handbook*. CRC Press, 2008.

[14] A. K. Naskar, S. Patra, and S. Sen, "Reconfigurable direct allocation for multiple actuator failures," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 397–405, 2015.