Software Rejuvenation Impacts on a Phased-Mission System for Mars Exploration

Stefano Ballerini, Laura Carnevali, Marco Paolieri
Dipartimento di Ingegneria dell'Informazione
Università di Firenze, Italy
{stefano.ballerini1@stud., laura.carnevali@,
marco.paolieri@}unifi.it

Kumiko Tadano, and Fumio Machida
Knowledge Discovery Research Laboratories
NEC, Japan
{k-tadano@bq, f-machida@ab}.jp.nec.com

***Abstract*—When software contains aging-related faults and the system has a long mission period, phased-mission systems consisting of several software components can suffer from software aging, which is a progressive degradation of the software execution environment. Failures caused by software aging might impact on the mission success probability. In this paper, we present a model for a phased-mission system with software rejuvenation, and analyze the impacts of software rejuvenation on the success probability and completion time distribution of the mission. The mission of Mars exploration rover is considered as an example of phased-mission system. The analysis results show that the mission success probability is improved by software rejuvenation at the cost of the mission completion time.**

Keywords—Mars exploration rover, phased-mission system, software aging, software rejuvenation, stochastic time Petri nets.

I. INTRODUCTION

A Phased Mission System (PMS) performs a sequence of phases to accomplish a specific mission. Each phase may include several actions that have to be performed in order to proceed with the execution of the subsequent phase. System configuration, actions duration, and used resources may vary from phase to phase. Examples of PMSs are found in many industrial fields which notably include those related to nuclear, aerospace, chemical, and electronic systems [1].

An important reliability measure of PMSs is the *mission success probability* which is affected by operational errors as well as intrinsic resource failures. While operational errors directly impact on the success of a phase, resource failures can occur according to the life cycle of resources. The resulting mission success probability can be evaluated by techniques based on reliability block diagrams [2], fault trees [1], [2], [3], Markov chains [4], [5], and Petri nets [6], [7]. Such methods support quantitative evaluation of system reliability, allowing system designers to implement effective measures to improve the mission success probability. When software components are resources of the PMS, also failure and recovery behaviors of software need to be taken into account. In particular, faults in software components might generate errors during the execution and cause a system failure compromising the entire system functionality.

Software aging is the phenomenon of progressive degradation of the software execution environment caused by a special type of software faults called aging-related bugs [8]. As the development of PMSs is often completed within a limited period of time, residual aging-related bugs tend to manifest themselves during the mission period, thus reducing the mission success probability. Software rejuvenation [9] is a known and effective measure that counteracts software aging during the PMS operation by restarting the software execution environment before aging-related failures are encountered. Although software rejuvenation clears the aging state of resources, it requires additional downtime and hence might affect the mission completion time.

In this paper, we extend the approach of [7] for modeling and analysis of PMSs in order to evaluate the impact of software rejuvenation of resources affected by software aging. To this end, we consider the example of a Mars Exploration Rover (MER) [10], a robotic exploration system that performs a set of daily activities organized as a phased mission. Specifically, the mission is modeled as a stochastic Time Petri Net (sTPN) [11] and then analyzed through the approach of stochastic state classes [12] to evaluate the impact of software aging and rejuvenation on the mission success probability and duration.

The rest of the paper is organized as follows: Section II introduces the MER example as a typical instance of PMS; Section III describes the sTPN formalism and the MER model; Section IV presents the analysis results; Section V discusses related work; Section VI draws the conclusions.

II. MARS EXPLORATION ROVER

The main intent of the mission of a MER [10] is to drive the rover to specific locations to perform on-site scientific investigations. In particular, the rover is programmed to complete a sequence of daily routine activities including: *i)* waking up at a specific time, *ii)* receiving commands from the base on Earth (uplink), *iii)* navigating to a specific destination, *iv)* carrying out surface operations, *v)* summarizing data and sending it to the base (downlink), and *vi)* entering a sleep mode. Although the rover activities may include other actions to be performed, we consider the above mentioned six phases.

Since the rovers are solar-powered, they are usually designed to sleep through the night and wake up autonomously at predefined communication windows [13]. The wake-up step is performed to warm up some actuators which will be used in the subsequent phases. During the communication windows, data uplink of key activity-related information with the overhead orbiters takes place, and once such commands necessary to execute the day activities are loaded, the operational stage is started. This usually includes some exploration (a few meters per day) and a specific activity like brushing a rock through the Rock Abrasion Tool [13]. At the end of the predefined goal, the engineering and science data is transmitted to Earth to determine the status of the rover and its surroundings. Such information is processed, stored in a database, and analyzed by an engineering team planning the activities for the next day. During the sleep time, most of the rover electronics are turned off, while the mission clock and the battery charger board remain on [13].

Rovers do not have traditional disk drives but rely on FLASH memory for their storage and all the previously cited activities use the FLASH. Software aging inside the FLASH memory might cause a serious impact on the mission of the Rover. On January 4th 2004, NASA's MERs *Spirit* and *Opportunity* landed successfully on Mars. An anomaly occurred on the Spirit vehicle on Sol (Martian day) 18 because of a design error in the software module that provides file system services [13]. Specifically, it suffered a debilitating FLASH anomaly precluding normal operation on the vehicle. The root cause of the problem was identified in the logical representation of deleted files in the DOS library, which affected the size of interlinked data structures in the storage of name and location of files in the private memory area. The outage was caused by a software bug (i.e., an internal human-made non-malicious permanent software development fault) which led to the accumulation of errors due to aging-related faults. This comprises an example of aging-related behaviors in space missions and points out the need to minimize their impact on the mission success probability.

III. MODELING

We model the MER mission using sTPNs [11]. An sTPN is a tuple $< \mathrm{P}, \mathrm{T}, \mathrm{A}^{-}, \mathrm{A}^{+}, \mathrm{A}^{\cdot}, \mathrm{m}_0, \mathrm{EFT}, \mathrm{LFT}, \mathcal{F}, C, \mathrm{E}, \mathrm{L} >$, where: P is a set of places, T is a set of transitions, $\mathrm{A}^{-} \subseteq P \times T$ is a set of precondition arcs, $\mathrm{A}^{+} \subseteq T \times P$ is a set of postcondition arcs, $\mathrm{A}^{\cdot} \subseteq P \times T$ is a set of inhibitor arcs, m_0 is the initial marking associating each place with a non-negative number of tokens, $\mathrm{EFT}: \mathrm{T} \rightarrow \mathbb{R}_0^+$ and $\mathrm{LFT}: \mathrm{T} \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ associate each transition with a static earliest firing time and a (possibly infinite) static latest firing time, respectively ($EFT(t) \leq LFT(t), \forall\, t \in T$), $\mathcal{F}: T \rightarrow F_t^s$ associates each transition $t \in T$ with a static Cumulative Distribution Function (CDF) F_t^s supported over $[ETF(t), LFT(t)]$. We assume that F_t^s is absolutely continuous over its support and that there exists a Probability Density Function (PDF) f_t such that $F_t^s = \int_0^x f_t(y)dy$. $C: T \rightarrow \mathbb{R}^+$ associates each transition with a weight used to resolve the random switch between concurrent transitions with the same firing time. A transition t is called *immediate* (IMM) if $[ETF(t), LFT(t)] = [0,0]$ and *timed* otherwise; a timed transition t is called *exponential* (EXP) if $F_t^s(x) = 1 - e^{\lambda x}$ over $[0,\infty]$ for some $rate\, \boldsymbol{\lambda} \in \mathbb{R}_0^+$ and *general* (GEN) otherwise; a GEN transition t is called *deterministic* (DET) if $EFT(t) = LFT(t) \geq 0$ and *distributed* otherwise. $E: T \rightarrow \{true, false\}^{\mathbb{N}^P}$ associates each transition with an *enabling function* that, in turn, associates each marking with a boolean value. $L: T \rightarrow \mathcal{P}(P)^{\mathbb{N}^P}$ associates each transition with a *flush function* that, in turn, associates each marking with a set of places. For space limitations, we refer the reader to [11], [7] for a complete discussion on syntax and semantics of sTPNs.

We model a daily mission of MER as a PMS with six phases, each made of one action, i.e., *i)* waking up, *ii)* commands uplink, *iii)* navigation, *iv)* surface operation, *v)* data downlink, and *vi)* sleep. The resource considered in this work is a FLASH memory, which is used in all the phases of the mission and might suffer from software aging as described in Section II. Although other resources such as CPUs, buses, and solar panels are also essential parts of the system and can fail during the mission as well, we primarily focus on the FLASH memory as a representative example of aging-affected resource. Activities and resources are modeled as in [7]; Figures 1 and 2 report the model of an activity (named *a1*) and a resource (named *R*), respectively.

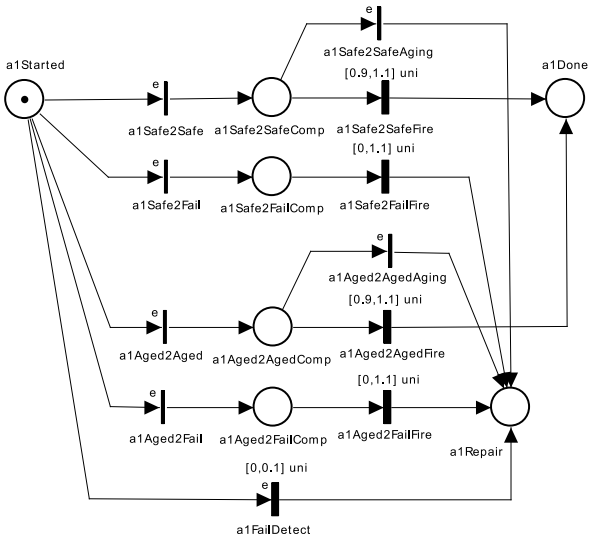

Figure 1. The model of an activity. IMM and timed transitions are drawn as thin and black rectangular boxes, respectively.

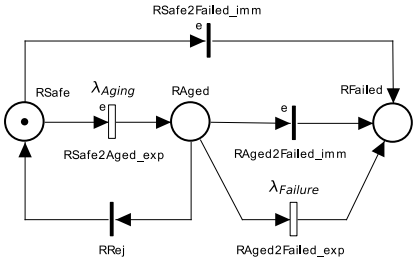

Figure 2. The model of a resource. EXP transitions are drawn as white rectangular boxes.

The action model and the resource model are mutually dependent through enabling and flush functions. The resource can be in a *safe, aged* or *failed* state, modeled by places *RSafe, RAged* and *RFailed*, respectively (Figure 2). The EXP transitions *RSafe2Aged_exp* and *RAged2Failed_exp* represent the aging behavior; the IMM transitions *RSafe2Failed_imm* and *RAged2Failed_imm* model the direct impact of the actions on the resource; the IMM transition *RRej* accounts for the successful completion of a rejuvenation action.

When a token arrives in *a1Started* (Figure 2), the action starts with a probabilistic switch modeling the concentrated probabilities of failure due the resource usage by the action. Depending on the state of the resource (Figure 2), distinct sets of transitions are enabled: *i) a1Safe2Safe* and *a1Safe2Fail* if the token is in *RSafe*; *ii) a1Aged2Aged* and *a1Aged2Fail* if the token is in *RAged*; or *iii) a1FailDetect* if the token is in *RFailed*. In cases *i* and *ii*, the choice between the enabled transitions is performed according to a discrete distribution given by transition weights, while the action duration is determined by the PDF of the subsequent GEN transition, i.e., *a1Safe2SafeFire*, *a1Safe2FailFire*, *a1Aged2AgedFire*, and *a1Aged2AgedFailFire*, respectively. Conversely, in case *iii*, the only enabled transition *a1FailDetect* also models the action duration. The successful completion of the action is modeled by a token in *a1Done*, while the failure of the action is represented by a token in *a1Repair*.

Note that, in the present model, resource repairs are not considered and, thus, the failure of an action causes a failure of the overall procedure. Also note that, if a resource does not fail due to usage, it could nevertheless fail due to software aging, thus causing a failure of the action that is using it. Such behavior is modeled by the IMM transitions *a1Safe2SafeAging* and *a1Aged2AgedAging* (Figure 1), which become enabled as soon as *RAged2Failed_exp* (Figure 2) fires.

Figure 3 shows the model of a rejuvenation action, which is similar to the model reported in Figure 1. A rejuvenation requires a preliminary step for aging detection (modeled by the GEN transition *rejAgingDet*), which is subject to potential aging failure (represented by the IMM transition *rejAgingDetAging*, which becomes enabled as soon as *RAged2Failed_exp* in Figure 2 fires). If the resource is safe, no aged state is detected and the sequence of IMM transitions *rejSafe2Safe* and *rejSafe2SafeFire* is fired. Conversely, if the resource is aged, the IMM transition rejAged2Safe fires, enabling the GEN transition *rejAged2SafeFire* modeling the rejuvenation duration.

While we assume that rejuvenation cannot directly impact a resource leading it to a failure state or leaving it in an aged state, we take into consideration that a resource could fail because of software aging during rejuvenation. In fact, during the initial steps of rejuvenation, a system usually has to perform operations such as internal state backup and information storage, which may encounter aging and lead to the action failure. Such behavior is modeled by the IMM transition *rejAged2SafeAging*, which becomes enabled as soon as *RAged2Failed_exp* (Figure 2) fires.

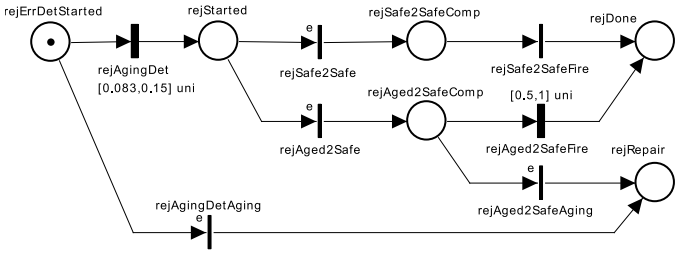

Figure 3. The model of a rejuvenation action.

IV. EXPERIMENTAL RESULTS

Figure 4 shows a sketch of the model of the MER mission, which is made of a resource model for the FLASH memory (like the one of Figure 2, not shown in Figure 4) and six chained action models (like the one of Figure 1) for the MER daily activities, here named *a1, a2, ..., a6*. Each action is outlined using a box with three circles: the upper left, upper right, and lower right circles correspond to places *a1started, a1Done,* and *a1Repair* in the action model of Figure 1, respectively. The overall mission is considered successfully completed when a token is placed in *a6Done*, which represents the successful completion of the last phase; conversely, it is considered failed if any of the actions is not correctly performed because of the failure of the resource (due to aging or the direct impact of the action itself).

Resource rejuvenation can be performed at the completion of specific actions. In the example of Figure 4, rejuvenation can be performed between actions *a5* and *a6*, and thus the mission model also includes a rejuvenation model (like the one of Figure 3) connected with the action models of *a5* and *a6*. Tables 1a, 1b, and 1c specify the enabling functions associated with transitions of the action model (the table applies to all the six actions and thus no action name is specified), the rejuvenation model, and the resource model, respectively.

Due to the absence of real data, we suppose a mean execution time of 1 h for the wakeup and sleep phases, 2 h for the communication phases (both uplink and downlink), 3 h for the exploring as well as the surface operation. Given the stressing environment the rover has to deal with, for each action it has to perform, we assume a duration equal to the mean value ± 0.1 h (6 minutes) associated with a uniform distribution, supposing that the processing performance is not changed when the FLASH resource is *safe* or *aged* (i.e., the duration of the actions is not influenced by the resource state). Table 2 shows the supports of model transitions that represent the duration of actions, as well the weights of transitions used to represent concentrated failure probabilities.

We also assume that the time needed to detect a resource failure (represented by transition *a1FailDetect* in Figure 1) is uniformly distributed between [0,0.1] h in each phase. We also assume that the aging detection phase (represented by transition *rejAgingDet* in Figure 3) has a uniformly distributed duration between 5 and 10 minutes (i.e., between 0.083 and 0.15 h) and that, in case some aging is detected, rejuvenation (modeled by transition *rejAged2SafeFire*in Figure 3) takes a time uniformly distributed between 30 minutes and 1 h.

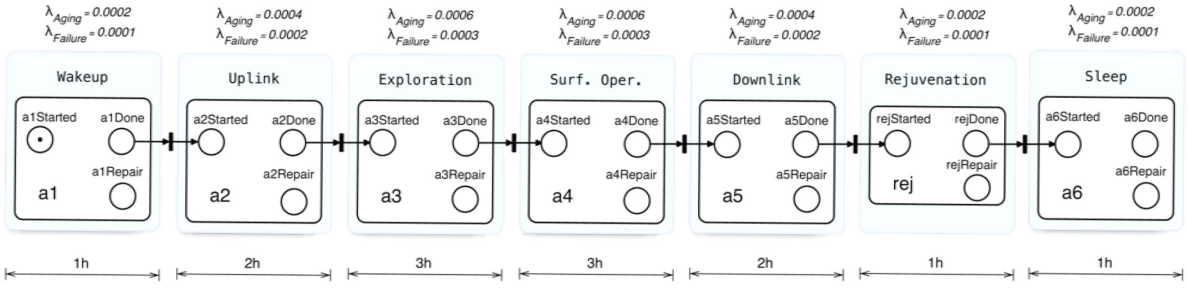

Figure 4. A sketch of the model of the MER mission referring to the case where rejuvenation is scheduled between the fifth and sixth phase.

Table 1. Enabling Conditions of the actions (a), rejuvenation (b) and resource (c) models.

Transition	Enabling condition
Safe2Safe	RSafe==1
Safe2Fail	RSafe==1
Aged2Aged	RAged==1
Aged2Fail	RAged==1
FailDetect	RFailed==1
Safe2SafeAging	RFailed==1
Aged2AgedAging	RFailed==1

(a)

Transition	Enabling condition
rejAgingDetAging	RFailed==1
rejSafe2Safe	RSafe==1
rejAged2Safe	RAged==1
Aged2SafeAging	RFailed==1

(b)

Transition	Enabling condition
RSafe2Failed_imm	a1Safe2FailComp==1 || a2Safe2FailComp==1 || …
RAged2Failed_imm	a1Aged2FailComp==1 || a2Aged2FailComp==1 || …
RRej	rejDone==1

(c)

Table 2. Supports and weights of transitions of the model used in the experiments. All distributions are uniform over their respective supports and the weights have to be intended as in the sTPN syntax.

		a1	a2	a3	a4	a5	a6
support	*Safe2Safe Aged2Aged*	[0.9,1.1]	[1.9,2.1]	[2.9,3.1]	[2.9,3.1]	[1.9,2.1]	[0.9,1.1]
	Safe2Fail Aged2Fail	[0,1.1]	[0,2.1]	[0,3.1]	[0,3.1]	[0,2.1]	[0,1.1]
weight	*Safe2Safe*	100	200	100	100	200	300
	Safe2Fail	1	1	1	1	1	1
	Aged2Aged	10	20	10	10	20	30
	Aged2Fail	1	1	1	1	1	1

To take into account the different computational requirements of the action performed in each phase, the aging rate that leads the FLASH resource from the safe to the aged state (i.e., λ_{Aging}) and the failure rate that leads the FLASH resource from the aged to the failed state (i.e., $\lambda_{Failure}$) vary from phase to phase. We assume the aging rate λ_{Aging} that leads a resource from the safe to the aged state to be two times higher than $\lambda_{Failure}$, and, more specifically, we assume that: $\lambda_{Aging} = 0.0002$ h^{-1} and $\lambda_{Failure} = 0.0001$ h^{-1} for the wakeup and sleep phases, $\lambda_{Aging} = 0.0004$ h^{-1} and $\lambda_{Failure} = 0.0002$ h^{-1} for the uplink and downlink phases, and $\lambda_{Aging} = 0.0006$ h^{-1} and $\lambda_{Failure} = 0.0003$ h^{-1} for the exploration and surface operation phases. During the rejuvenation phase, $\lambda_{Aging} = 0.0002$ h^{-1} and $\lambda_{Failure} = 0.0001$ h^{-1}.

Stochastic transient analysis of the MER model was performed through a new release of the ORIS Tool [14] based on the Sirio framework [15], which supports the derivation of the transient probability of reachable markings within a given time bound. Since the markings reached when an action fails or the overall mission is successfully completed are absorbing, the transient probability of such markings represents the CDF of the time until absorption. According to this, the completion time distribution of any intermediate phase as well as the overall mission can be easily derived.

To evaluate different rejuvenation schedules, three preliminary experiments were performed for comparison of possible policies: in the first one, no rejuvenation is performed; in the second one, rejuvenation can be performed between the fifth and the sixth phase; in the third one, rejuvenation can be performed between the third phase and the fourth phase. For each experiment, Figures 5 and 6 show the probability that the mission is successfully completed leaving the FLASH resource in a safe state or in an aged state, respectively. In particular, in the first experiment, after 12.6 h, the probability that the mission is successfully completed and that the resource is safe is 0.9901176, while the probability that the mission is successfully completed and the resource is aged is 0.0048605 (the remaining 0.0050219 is the probability that the mission is failed). In the second experiment, after 13.8 h, the probability that the mission is successfully completed leaving the resource in a safe state is increased to 0.9949329, while the probability that the mission is successfully completed leaving the resource in an aged state is decreased to

0.000198996. In the third experiment, the probability that the mission is successfully completed and the resource is safe is slightly decreased to 0.9927939, while the probability that the mission is successfully completed and the resource is aged is increased to 0.00261764. The latter data is also summarized in Table 3, highlighting how the total reliability of the mission is influenced by the position of the rejuvenation along the sequence of phases. It is worth noting that, in the second experiment, the reliability is lower than in the third one due to a too late rejuvenation policy, leading to a performance decrease because of the accumulation of aging.

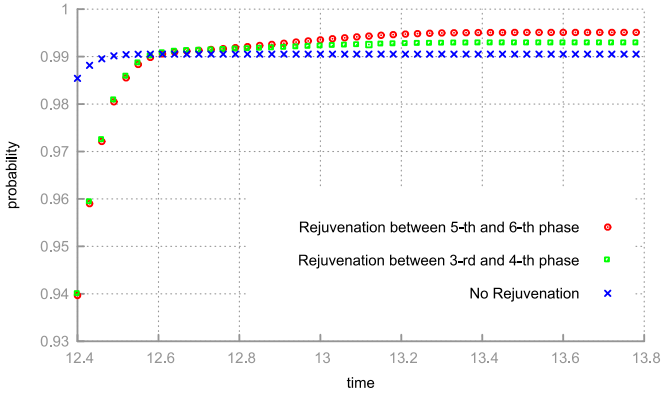

Figure 5. Probability that the overall mission is successfully completed and the resource is safe.

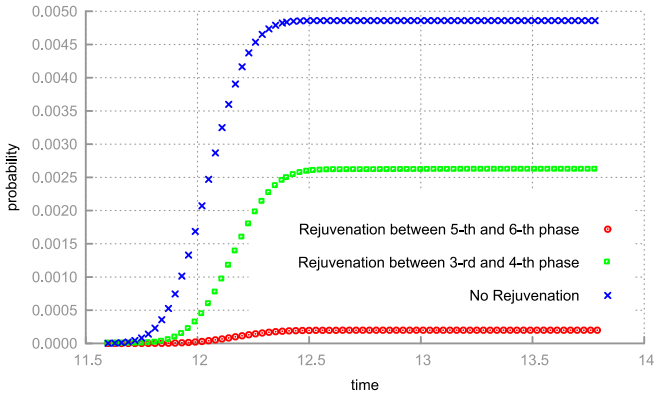

Figure 6. Probability that the overall mission is successfully completed and the resource is aged.

Rejuvenation Schedule	Probability of mission success and safe resource	Probability of mission success and aged resource	Total Mission success probability
No rejuvenation	0.9901176	0.0048605	0.9949781
Rej. between the 5th and 6th phase	0.9949329	0.000198996	0.9951319
Rej. between the 3rd and 4th phase	0.9927939	0.00261764	0.99541154

Table 3. Mission success probabilities by different rejuvenation schedules.

Figure 7 shows the completion time distribution in case of mission success. Note that, even if rejuvenation has a uniform distribution over [0.5,1] h plus a uniformly distributed duration between 0.083 and 0.1 h for aging detection, the completion time CDF of the overall mission is shifted in average by roughly 0.2 hours in the second and third scenarios. This actually depends on the fact that the completion time is prolonged only when software aging is actually detected and rejuvenation is successfully applied.

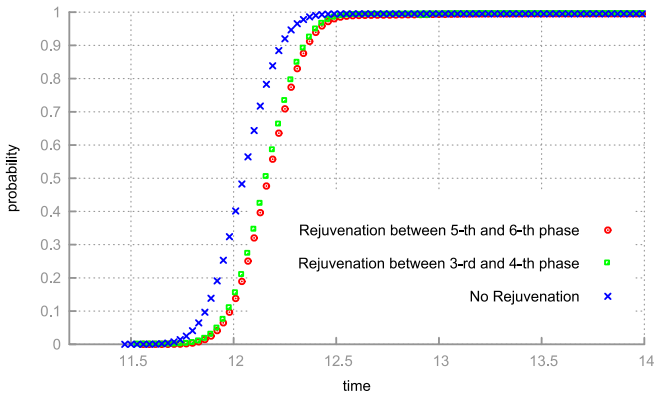

Figure 7. Completion time distribution of the overall mission.

The mission success probability is improved by ~ 0.015% and ~ 0.043% by applying software rejuvenation between the fifth and sixth phase and between the third and fourth phase, respectively. This shows the benefits achieved by incorporating a rejuvenation phase in the overall procedure and the importance of its collocation. The second experiment shows that a rejuvenation performed before the last phase of the mission can provide the highest probability of successfully ending the mission leaving the resource in a safe state. However, because of the long time elapsed between the beginning of the mission and the rejuvenation, an aging behavior can take place leading to a lower total reliability. At the same time, a rejuvenation scheduled in the middle of the mission yields an improvement in the reliability, but also a lower probability of ending up the mission by leaving the resource in a safe state.

V. Related Work

The effectiveness of software rejuvenation is often evaluated in terms of system availability since downtimes caused by both failure and rejuvenation affect the system availability. Huang et al. [9] and Garg et al. [16] construct models for software rejuvenation and analyze the impacts on system availability by hours of downtime. As an alternative dependability measure, the interval reliability affected by software rejuvenation is studied in [17], [18]. Reinecke et al. conduct a simulation study to characterize the effectiveness of rejuvenation on service unavailability, which is defined as the ratio of uncompleted jobs over all the arrived jobs [19]. Wang et al. considered the probability of job blocking and the throughput of jobs in addition to system availability [20]. Instead of looking into the availability measure, we study the impacts of software rejuvenation on the mission success probability and job completion time, which are important reliability measures for PMSs.

Determining the optimum software rejuvenation schedule is one of the challenging issues discussed in the research on software aging and rejuvenation. The optimal rejuvenation schedule that maximizes the availability of a system has been

analyzed using Markov Regenerative Stochastic Petri Nets (MRSPNs) [16] and semi-Markov models [21]. In terms of cost optimality, the software rejuvenation schedule should be considered together with the software test phase [22]. Okamura et al. pointed out that rejuvenation can be applied during the limited operational period in practice, and presented an optimal rejuvenation time under an opportunity-based software rejuvenation policy [23]. In general, for a PMS, opportunity-based software rejuvenation is more suitable than periodic rejuvenation schedule. We analyzed the different impacts of software rejuvenation by changing the rejuvenation timing among phases.

When a long running job in a system encounters software aging, the job completion time is affected by aging. The distribution of the job completion time on a virtualized server subject to software aging and rejuvenation is analyzed by Markov models in [24]. Software life-extension [25] can be used as an alternative countermeasure to software aging if the long running job should not be interrupted by software rejuvenation. In this paper, we also evaluate the job completion time distribution affected by simple software rejuvenation. While the presented model only considers a simple rejuvenation process, the modeling power provided by the ORIS Tool [14], [15] can be leveraged to easily extend evaluation to systems using server virtualization, software life-extension, and software rejuvenation with different policies.

VI. Conclusions

In this paper, we experiment with an approach for modeling and evaluation of phased-mission systems using an example of a robotic exploration system. Specifically, we present the stochastic time Petri net model of a Mars exploration rover with and without software rejuvenation. We evaluate the impact of software rejuvenation on the mission success probability and completion time distribution by the method of stochastic state classes implemented in the ORIS Tool. The analysis results show that the mission success probability is improved by $\sim 0.015\%$ and $\sim 0.043\%$ by applying software rejuvenation in different points of the mission, at the cost of an increase in the completion time. This shows how the modeling and solution technique can be leveraged to evaluate the benefits of different rejuvenation plans in phased mission systems.

Future work will be directed towards enriching the modeling framework so as to evaluate the impacts of software life-extension, to include software rejuvenation failures, and to relax the assumption of non-decreasing performances in the presence of software aging.

References

[1] G. R. Burdick, J. B. Fussell, D. M. Rasmuson, and J. R. Wilson. Phased mission analysis: a review of new developments and an application. *IEEE Trans. on Reliability*, vol. 26, no. 1, pp.43-49, 1977.

[2] J. D. Esary and H. Ziehms. Reliability Analysis of Phased Missions. In *Proc. of the Conf. on Reliability and Fault Tree Analysis*, SIAM, pp. 213-236, 1975.

[3] A. K. Somani, and K. S. Trivedi. Phased-mission System Analysis Using Boolean Algebraic Methods. In *Proc. of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 98-107, 1994.

[4] J. B. Dugan. Automated Analysis of Phased-Mission Reliability. *IEEE Trans. on Reliability*, vol. 40, no. 1, pp. 45-55, 1991.

[5] M. Alam, and U. M. Al-Saggaf. Quantitative Reliability Evaluation of Repairable Phased-Mission Systems Using Markov Approach. *IEEE Trans. on Reliability*, vol. R-35, no. 5, pp. 498-503, 1986.

[6] I. Mura, and A. Bondavalli. Markov Regenerative Stochastic Petri Nets to Model and Evaluate Phased Mission Systems Dependability. *IEEE Trans. on Computers*, vol. 50, no. 12, pp. 1337-1351, 2001.

[7] L. Carnevali, M. Paolieri, K. Tadano, and E. Vicario. Towards the quantitative evaluation of phased maintenance procedures using non-Markovian regenerative analysis. In *Proc. of 10th European Workshop on Performance Engineering (EPEW13)*, 2013.

[8] M. Grottke, R. Matias Jr., and K. S. Trivedi. The fundamentals of software aging. In *Proc. of the 1st Int'l Workshop on Software Aging and Rejuvenation (WoSAR2008)*, pp.1-6, 2008.

[9] Y. Huang, C. Kintala, N. Kolettis, N.D. Fulton. Software rejuvenation: analysis, module and applications. In *Proc. of 25th Symp. on Fault Tolerant Computing (FTCS-25)*, pp.381-390, 1995.

[10] J. K. Erickson. Living the dream: an overview of the Mars exploration project. *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 12–18, 2006.

[11] E. Vicario, L. Sassoli, and L. Carnevali. Using stochastic state classes in quantitative evaluation of dense-time reactive systems. *IEEE Trans. on Software Engineering*, vol. 35, no. 5, pp. 703–719, 2009.

[12] A. Horváth, M. Paolieri, L. Ridi, and E. Vicario. Transient analysis of non-Markovian models using stochastic state classes. *Performance Evaluation*, 2012.

[13] G. Reeves and T. Neilson. The Mars Rover Spirit FLASH anomaly. In *Proc. of IEEE Aerospace Conference*, pp. 4186–4199, 2005.

[14] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario. Oris: a tool for modeling, and evaluation of real-time systems. *Int. Journal of Software Tools for Technology Transfer,* 12(5):391-403, 2010.

[15] L. Carnevali, L. Ridi, and E. Vicario. A Framework for Simulation and Symbolic State Space Analysis of Non-Markovian Models. In *Proc. of SAFECOMP*, pp. 409–422, 2011.

[16] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of software rejuvenation using Markov regenerative stochastic Petri nets. In *Proc. of Int'l Symp. on Software Reliability Engineering (ISSRE 1995)*, pp. 180-187, 1995.

[17] H. Suzuki, T. Dohi, N. Kato, and K. S. Trivedi. Maximizing interval reliability in operational software system with rejuvenation. In *Proc. of Int'l Symp. on Software Reliability Engineering (ISSRE 2003)*, pp. 479-490, 2003.

[18] T. Dohi, H. Okamura and K. S. Trivedi. Optimizing Software Rejuvenation Policies under Interval Reliability Criteria. In *Proc. of Int'l Conf. on Ubiquitous Intelligence & Computing and Autonomic & Trusted Computing (UIC/ATC)*, pp. 478-485, 2012.

[19] P. Reinecke, and K. Wolter. A simulation study on the effectiveness of restart and rejuvenation to mitigate the effects of software aging. In *Proc. of the 2nd Int'l Workshop on Software Aging and Rejuvenation (WoSAR 2010)*, 2010.

[20] D. Wang, W. Xie, and K. S. Trivedi. Performability analysis of clustered systems with rejuvenation under varying workload. *Performance Evaluation*, vol. 64, no. 3, pp. 247-265, 2007.

[21] T. Dohi, K. Goševa-Popstojanova, and K. Trivedi. Estimating Software Rejuvenation Schedules in High-Assurance Systems. *The Computer Journal*, vol. 44, no. 6, pp. 473–485, Jan. 2001.

[22] M. Grottke and B. Schleich. Cost Optimality in Testing and Rejuvenation. In *Proc. of 4th Int'l Workshop on Software Aging and Rejuvenation (WoSAR 2012)*, pp. 259-264, 2012.

[23] H. Okamura and T. Dohi. Optimization of Opportunity-Based Software Rejuvenation Policy. In *Proc. of 4th Int'l Workshop on Software Aging and Rejuvenation (WoSAR 2012)*, pp. 283-286, 2012.

[24] F. Machida, V. F. Nicola, and K. S. Trivedi. Job Completion Time on a Virtualized Server with Software Rejuvenation. *ACM Journal of Emerging Technologies*, to appear.

[25] F. Machida, J. Xiang, K. Tadano and Y. Maeno. Software life-extension: a new countermeasure to software aging. In *Proc. of Int'l Symp. on Software Reliability Engineering (ISSRE 2012)*, pp. 131-140, 2012