Enhancing Embedded Software System Reliability using Virtualization Technology

Thandar Thein, Sung-Do Chi and Jong Sou Park

Computer Engineering Department, Korea Aerospace University, Seoul

Summary
Embedded systems, already ubiquitous, are becoming more and more part of everyday life. The complexity of modern embedded software poses formidable challenges to system reliability. The increasing use of software for implementing the functionality, has led to increasing demands for more sophisticated Embedded Software Maintenance. Preventive maintenance is applied to improve the device availability. In operational software system, their values depend on the software system structure as well as on the software component availability and reliability. These values decrease as the software age increases. With reactive maintenance becoming more and more complex and expensive, software developers are seeking more proactive approaches to maintenance. Virtualization has recently become important technology for embedded systems. In this paper, we study the virtualization technology and proactive software rejuvenation methodology to counteract the operational embedded software system aging problem. We also present the conditional based preventive maintenance model and derive the closed-form expressions of operational embedded software system availability through a Markov Process. Numerical examples are presented to illustrate the applicability of the model.
Keywords: *availability, embedded software system, modeling, software rejuvenation, virtualization.*

1. Introduction

Embedded systems, already ubiquitous, are becoming more and more part of everyday life, to the degree that it is becoming hard to imagine living without them. They are increasingly used in mission- and life-critical scenarios. Correspondingly, there are high and increasing requirements on safety, reliability and security [2].

The emergence of embedded systems in products of virtually all domains has resulted in a dramatic increase in products incorporating *Embedded Software*. The most recent generation of embedded systems relies heavily on embedded software.

In the past, embedded systems were characterized by simple functionality, a single purpose, no or very simple user interface, and no or very simple communication channels. They also were closed in the sense that all the software on them was loaded pre-scale by the manufacturer, and normally remained unchanged for the lifetime of the device. The amount of software was small. Modern embedded systems are increasingly taking on characteristics of general-purpose systems.

Many embedded systems have to operate for long time nonstop and providing high availability. These kinds of embedded system suffered from software aging. The performance characteristics of a software system are degraded over time through continuous running. The effects become manifest in reduced service performance and/or failures (system crashes or hangs). Other problems such as data inconsistency, memory leakage, unreleased file-locks, data corruption, storage space fragmentation and an accumulation of round-off errors may also occur. This constitutes a phenomenon called software aging [1], [9]. A good medicine against software aging is software rejuvenation [6].

The emergence of this wide spectrum of embedded system, and the increasing use of software for implementing the functionality, has led to increasing demands for more sophisticated *Embedded Software Maintenance*. Maintenance of embedded software is much more expensive than maintenance of non-embedded software. Preventive maintenance (such as software rejuvenation) is applied to improve the device availability.

In operational software system, their values depend on the software system structure as well as on the software component availability and reliability. These values decrease as the software age increases. With reactive maintenance becoming more and more complex and expensive, software developers are seeking more proactive approaches to maintenance. Preventive maintenance, however, incurs an overhead which should be balanced against the cost incurred due to unexpected outage caused by a failure.

Virtualization has recently become important technology for embedded systems. High-performance microkernels [4], OKL4, are a technology that provides a good match for the requirements of next-generation embedded systems.

Manuscript received November 5, 2008
Manuscript revised November 20, 2008

The goal of this paper is to describe the state-of-the-art of embedded software maintenance system using virtualization technology and proactive software rejuvenation methodology.

We analyze the condition-based preventive maintenance models with and without virtualization technology and derive the closed-form expressions. The closed-form solutions are compared with SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) tool [8] evaluation to verify the correctness of our result.

The rest of this paper is organized as follows. Section 2 gives an overview of software aging, software rejuvenation and virtualization technology in embedded systems. The system, the model description, solution methodology and some numerical results are described in Section 3, and Section 4 concludes the paper.

2. Embedded Systems

An embedded system is a part of a product with which an end user does not directly interact or control [3]. Our society has come to expect uninterrupted service from many of the systems that it employs, such as phones, automated teller machines, and credit card verification networks. Many of these are implemented as embedded systems.

2.1 Software Aging in Embedded Systems

Modern embedded systems feature a wealth of functionality and, as a result, are highly complex. This is particularly true for their software, which frequently measures in the millions of lines of code and is growing strongly. The complexity of modern embedded software poses formidable challenges to system reliability. Systems of that complexity are, for the foreseeable future, impossible to get correct – in fact, they can be expected to contain tens of thousands of bugs. This complexity presents a formidable challenge to the reliability of the devices.

Young programs are slight, sprightly things, but are lively and attractive. Bugs and misunderstood requirements disappoint the users, but only a few expected improvements or change for the better. With passing of time and with relentless endeavors, the young program becomes full fledged and productive in business process or product. By middle age, patches, improvements and creeping featuritus bloat the girth and hobble performance. The graying code slows and swells, though still satisfies the users. Continuing bug fixes and ever more features drive the software into old age.

2.2 Software Rejuvenation in Embedded Systems

Software rejuvenation is a technique for software fault tolerance which involves occasionally stopping the executing software, cleaning the internal state and restarting. This cleanup is done at desirable times during execution on a preventive basis so that unplanned failures, which result in higher costs compared to planned stopping, are avoided. The necessity to use this technique not only in general purpose computers but also in safety-critical and high availability systems clearly indicates the need of analysis in order to determine the optimal times to rejuvenate. A new software rejuvenation variation called Opportunistic Micro Rejuvenation (OMR) [7] is proposed where a task that “misbehaves” is identified and rejuvenated at an opportune instant, like when it is in a waiting state. OMR is natural fit with embedded software system aging.

2.3 Virtualization Technology in Embedded Systems

Virtualization, which originated on mainframes and finds increasing use on personal computers, has recently become popular in the embedded-systems space. Virtualization [5] refers to providing a software environment on which programs, including operating systems, can run as if on bare hardware (Figure 1). Such a software environment is called a virtual machine (VM). Such a VM is an efficient, isolated duplicate of the real machine. The hypervisor (or virtual-machine monitor) presents an interface that looks like hardware to the guest operating system. The software layer that provides the VM environment is called the virtual-machine monitor (VMM), or hypervisor.

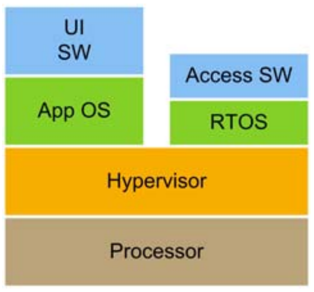

Fig. 1 Virtualization enabling the concurrent execution of an application OS and a real-time OS (RTOS)

Virtualization can provide some attractive benefits to embedded systems. One is support for heterogeneous operating-system environments. Virtualization supports architectural abstraction as the same software architecture can be migrated essentially unchanged between a multicore and a (virtualized) single core. Another motivation for virtualization is security and the wide support for virtualization enables a new model for distribution of application software. Virtualization can provide some attractive benefits to embedded systems, however there are significant limitations on the use of virtualization such as efficient sharing, scheduling, energy management, and software complexity [2].

3. System Model

Models are often an abstract representation of reality. In this section, we are interested in determining how virtualization technology can enhance the preventive maintenance action in embedded software system. We have considered the systems' availability in different stages with different actions in face of disruptions. The following scenarios are studied in this paper.

- Condition-based preventive maintenance model without virtualization
- Condition-based preventive maintenance model with virtualization

We construct the state transition models to describe the behavior of these systems as shown in figure 2 and 5.

Software aging in the target systems can be detected by monitoring the system state. We can restore system to initial or clean state using software rejuvenation. In our implementation, two kinds of preventive maintenance (software rejuvenation) will be performed on the system based on the deterioration stage: a minimal maintenance (a partial system clean up) and a major maintenance (clean restart). Assuming that, the time to repair is exponentially distributed with rate μ.

Further assuming that, an inspection is triggered after a mean duration $1/\lambda_{in}$, and takes an average time of $1/\mu_{in}$. After an inspection is completed, no action is taken if the system is found to be in healthy stage. On the other hand, if the system is found to be in the first deterioration stage, minimal maintenance action will be performed.

If the system is found to be in the second deterioration stage, major maintenance is carried out. By mapping through actions to these transition models with stochastic process, we get mathematical steady-state solution of the chain.

Consider an embedded software system that starts in a robust state and as time progresses, it can transit to deterioration state and eventually suffers a major failure. To verify the validity of our derivation formulae, we compare the results obtained from the closed-form solution and the result obtained from the numerical solution by SHARPE. For this purpose, the parameters are chosen as follows:

$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = 0.1$, $\mu in = 0.6$

$\mu=1$ $\mu_M = 2$, $\mu_m = 3$

3.1 Preventive Maintenance Model without Virtualization (PM model)

In this section, we are interested in determining how condition-based preventive maintenance approach without virtualization technology can improve availability of embedded software system. Assume that time to system failure is hypoexponential with rates λ_1, λ_2, and λ_3 as shown in Figure 2.

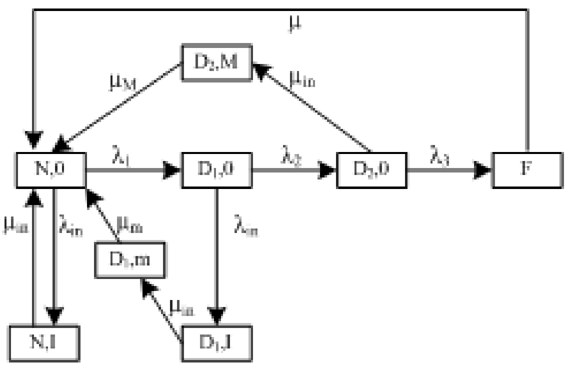

Fig. 2 State transition diagram of PM model

The state description of PM model is described in table 1.

Table 1 State Description for PM model.

State	Description
(N,O)	system is in normal operational state
(N,I)	system is under inspection state
(D_1,O)	system is operational but in the deterioration state
(D_1,I)	system is in deterioration and is under inspection state
(D_1,m)	system is in deterioration and is under minimal maintenance
(D_2,O)	system is operational but in major deterioration state
(D_2,M)	system is in deterioration and is under major maintenance state
(F)	system is in deterioration failure state

Writing down and solving the steady-state balance equations, we get the probabilities as follows.

$$P_{(N,O)} = \frac{1}{1+\frac{\lambda_{in}}{\mu_{in}}+\frac{\lambda_1}{\lambda_2+\lambda_{in}}\left(1+\frac{\lambda_{in}}{\mu_{in}}+\frac{\lambda_{in}}{\mu_m}+\frac{\lambda_2}{\mu_{in}+\lambda_3}(1+\frac{\mu_{in}}{\mu_M}+\frac{\lambda_3}{\mu})\right)} \quad (1)$$

$$P_{(N,I)} = \frac{\lambda_{in}}{\mu_{in}} P_{(N,O)} \tag{2}$$

$$P_{(D_1,O)} = \frac{\lambda_1}{\lambda_2 + \lambda_{in}} P_{(N,O)} \tag{3}$$

$$P_{(D_1,I)} = \frac{\lambda_{in}}{\mu_{in}} \frac{\lambda_1}{\lambda_2 + \lambda_{in}} P_{(N,O)} \tag{4}$$

$$P_{(D_1,m)} = \frac{\lambda_{in}}{\mu_m} \frac{\lambda_1}{\lambda_2 + \lambda_{in}} P_{(N,O)} \tag{5}$$

$$P_{(D_2,O)} = \frac{\lambda_2}{\mu_{in} + \lambda_3} \frac{\lambda_1}{\lambda_2 + \lambda_{in}} P_{(N,O)} \tag{6}$$

$$P_{(D_2,M)} = \frac{\mu_{in}}{\mu_M} \frac{\lambda_1}{\lambda_2 + \lambda_{in}} P_{(N,O)} \tag{7}$$

$$P_{(F)} = \frac{\lambda_3}{\mu} \frac{\lambda_2}{\mu_{in} + \lambda_3} \frac{\lambda_1}{\lambda_2 + \lambda_{in}} P_{(N,O)} \tag{8}$$

Availability models capture failure and repair behavior of systems and their components. States of the underlying Markov chain will be classified as up states or down states. The system is not available in the rejuvenation process in minimal maintenance state (D_1,m), major maintenance state (D_2,M) and the failure state (F). Since up states are (N,O), (N,I), (D_1,O), (D_1,I), and (D_2,O), we obtain an expression for the steady-state availability:

$$Availabilty_{(PM)} = P_{(N,O)} + P_{(N,I)} + P_{(D_1,O)} + P_{(D_1,I)} + P_{(D_2,O)} \tag{9}$$

We define the steady-state probabilities of the system are as follows:

$P_{(N,0)}$	The probability of the system is in normal operational state
$P_{(N,I)}$	The probability of the system is under inspection state
$P_{(D_1,0)}$	The probability of the system is operational but in the deterioration state
$P_{(D_1,I)}$	The probability of the system is in deterioration and is under inspection state
$P_{(D_1,m)}$	The probability of the system is in deterioration and is under minimal maintenance
$P_{(D_2,0)}$	The probability of the system is operational but in major deterioration state
$P_{(D_2,0)}$	The probability of the system is in deterioration and is under major maintenance state
$P_{(F)}$	The probability of the system is in deterioration failure state

In Figure 3, we plot the steady-state availability as a function of the mean time between inspections MTBI=1/λ_{in}. We use several different values of the time to carry out the inspection. Note that the steady-state availability reaches a maximum at MTBI=10 for μ_{in}=0.5.

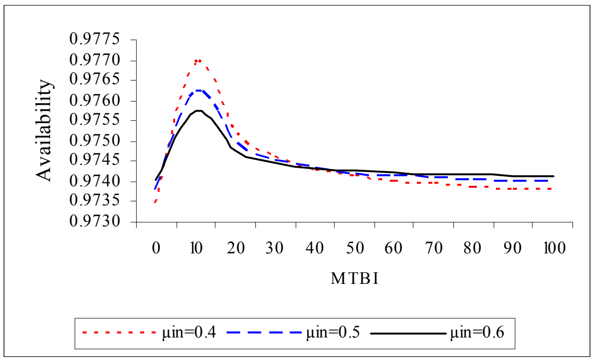

Fig. 3 Steady-state availability of PM model with different μ_{in}.

3.2 Preventive Maintenance Model with Virtualization (PMV model)

In this section, we discuss our condition-based preventive maintenance with virtualization approach. On top of the Hypervisor, we create 2 VMs as shown in Figure 4. The main application will be run on active VM. Another VM will work as standby VM. The embedded agent resides on embedded system as an application layer and it monitors system state, rejuvenating the applications or the operating system to recover from failures. Monitoring events in embedded system occur periodically with predefined interval. When major deterioration happens in active VM, first standby VM will be started and it will take active VM role. After that major deterioration VM will perform major maintenance. By using this approach the system can provide continued services even if VM needs to perform major maintenance.

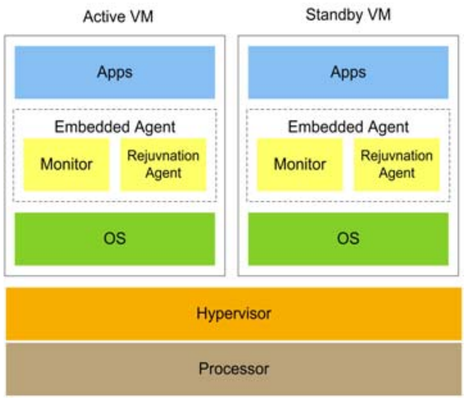

Fig 4. Virtualized system

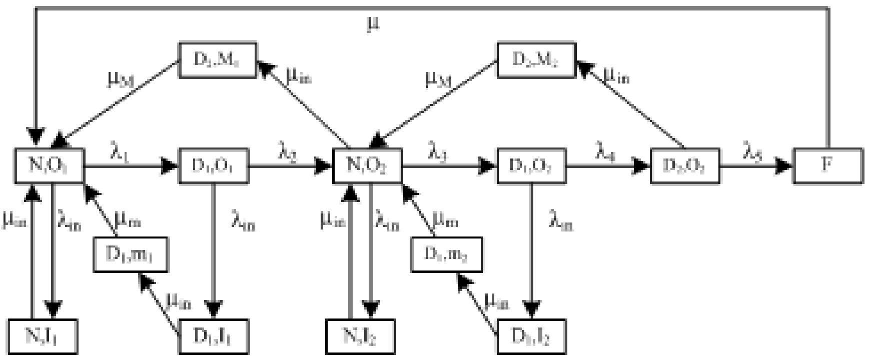

Fig. 5 State transition diagram of PMV model

Now we consider the state transition diagram of PMV model as shown in Figure 5. Assume that time to system failure is hypoexponential with rates λ_1, λ_2, λ_3, λ_4 and λ_5. The state description of PMV model is described in table 2

Table 2 State Description for PMV model.

State	Description
(N,O_i)	ith VM is in normal operational state
(N,I_i)	ith VM is under inspection state
(D_1,O_i)	ith VM is operational but in the deterioration state
(D_1, I_i)	ith VM is in deterioration and is under inspection state
(D_1, m_i)	ith VM is in deterioration and is under minimal maintenance
(D_2, O_i)	ith VM is in operation but major deterioration state
(D_2, M_i)	ith VM is in deterioration and is under major maintenance state
(F)	System is in deterioration failure state
i = (1, 2)	(number of virtual machines)

Writing down and solving the steady-state balance equations, we get the probabilities of as follows.

$$P_{(N,O_1)} = \begin{bmatrix} 1+\dfrac{\lambda_{in}}{\mu_{in}}+\dfrac{\lambda_1}{\lambda_2+\lambda_{in}}(1+\dfrac{\lambda_{in}}{\mu_{in}}+\dfrac{\lambda_{in}}{\mu_m})+ \\ Z\begin{Bmatrix} 1+\dfrac{\mu_{in}}{\mu_M}+\dfrac{\lambda_{in}}{\mu_{in}}+\dfrac{\lambda_3}{\lambda_4+\lambda_{in}}(1+\dfrac{\lambda_{in}}{\mu_{in}}+ \\ \dfrac{\lambda_{in}}{\mu_m}+\dfrac{\lambda_4}{\lambda_5+\mu_{in}}(1+\dfrac{\mu_{in}}{\mu_M}+\dfrac{\lambda_5}{\mu})) \end{Bmatrix} \end{bmatrix} \tag{10}$$

$$P_{(N,I_1)} = \frac{\lambda_{in}}{\mu_{in}} P_{(N,O_1)} \tag{11}$$

$$P_{(D_1,O_1)} = \frac{\lambda_1}{\lambda_2+\lambda_{in}} P_{(N,O_1)} \tag{12}$$

$$P_{(D_1,I_1)} = \frac{\lambda_{in}}{\mu_{in}} \frac{\lambda_1}{\lambda_2+\lambda_{in}} P_{(N,O_1)} \tag{13}$$

$$P_{(D_1,m_1)} = \frac{\lambda_{in}}{\mu_m} \frac{\lambda_1}{\lambda_2+\lambda_{in}} P_{(N,O_1)} \tag{14}$$

$$P_{(N_2,O_2)} = ZP_{(N,O_1)} \tag{15}$$

$$P_{(D_2,M_1)} = \frac{\mu_{in}}{\mu_M} ZP_{(N,O_1)} \tag{16}$$

$$P_{(N,I_2)} = \frac{\lambda_{in}}{\mu_{in}} ZP_{(N,O_1)} \tag{17}$$

$$P_{(D_1,O_2)} = \frac{\lambda_3}{\lambda_4+\lambda_{in}} ZP_{(N,O_1)} \tag{18}$$

$$P_{(D_1,I_2)} = \frac{\lambda_{in}}{\mu_{in}} \frac{\lambda_3}{\lambda_4+\lambda_{in}} ZP_{(N,O_1)} \tag{19}$$

$$P_{(D_1,m_2)} = \frac{\lambda_{in}}{\mu_m} \frac{\lambda_3}{\lambda_4+\lambda_{in}} ZP_{(N,O_1)} \tag{20}$$

$$P_{(D_2,O_2)} = \frac{\lambda_3}{\lambda_4+\lambda_{in}}\frac{\lambda_4}{\lambda_5+\mu_{in}}ZP_{(N,O_1)} \quad (21)$$

$$P_{(D_2,M_2)} = \frac{\lambda_3}{\lambda_4+\lambda_{in}}\frac{\lambda_4}{\lambda_5+\mu_{in}}\frac{\mu_{in}}{\mu_M}ZP_{(N,O_1)} \quad (22)$$

$$P_{(F)} = \frac{\lambda_3}{\lambda_4+\lambda_{in}}\frac{\lambda_4}{\lambda_5+\mu_{in}}\frac{\lambda_5}{\mu}ZP_{(N,O_1)} \quad (23)$$

$$\text{Here,}\quad Z = \frac{\dfrac{\lambda_1\lambda_2}{\lambda_{in}+\lambda_2}}{\mu_{in}+\lambda_3-\dfrac{\lambda_3}{\lambda_4+\lambda_{in}}(\lambda_{in}+\dfrac{\mu_{in}\lambda_4}{\lambda_5+\mu_{in}})} \quad (24)$$

Since up states are ((N,O$_1$), (N,I$_1$), (D$_1$,O$_1$), (D$_1$,I$_1$),(D$_2$, M$_1$), (N,O$_2$), (N,I$_2$), (D$_1$,O$_2$), (D$_1$, I$_2$) and (D$_2$, O$_2$), we obtain an expression for the steady-state availability:

$$Availability_{(PMV)} = P_{(N,O_1)} + P_{(N,I_1)} + P_{(D_1,O_1)} + P_{(D_1,I_1)} + P_{(D_2,M_1)} + P_{(N,O_2)} + P_{(N,I_2)} + P_{(D_1,O_2)} + P_{(D_1,I_2)} + P_{(D_2,O_2)} \quad (25)$$

We define the steady-state probabilities of the PMV model are as follows:

$P_{(N,O_i)}$	The probability of ith VM is in normal operational state
$P_{(N,I_i)}$	The probability of ith VM is under inspection state
$P_{(D_1,O_i)}$	The probability of ith VM is operational but in the deterioration state
$P_{(D_1,I_i)}$	The probability of ith VM is in deterioration and is under inspection state
$P_{(D_1,m_i)}$	The probability of ith VM is in deterioration and is under minimal maintenance
$P_{(D_2,O_i)}$	The probability of ith VM is in operation but major deterioration state
$P_{(D_2,M_i)}$	The probability of ith VM is in deterioration and is under major maintenance state
$P_{(F)}$	The probability of System is in deterioration failure state
i = (1, 2)	(number of virtual machines)

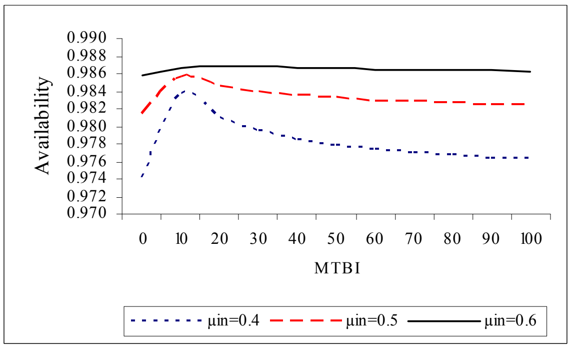

Fig. 6 Steady-state availability of PMV model with different μ_{in}.

The variation of the steady-state availability with various MTBI is shown in Figure 6. Note that the steady-state availability reaches a maximum at MTBI= 10 for μ_{in}=5.

3.3 Validation of Closed-form Results

In this section, we compare the steady-state availability of virtualized system and non virtualized system as shown in Figure 7.

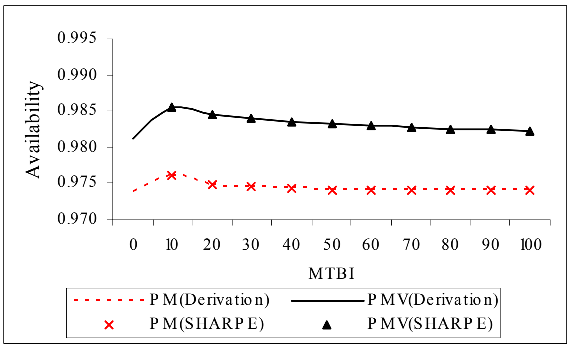

Fig. 7 Steady-state availability with different μ_{in} .

To verify the validity of our formula derivations, we compare the results obtained from the closed-form solution and the results obtained from the numerical solution by SHARPE. We found that our closed-form result and SHARPE tool result are same.

4. Conclusions

In this paper, we modeled and analyzed condition-based preventive maintenance in operational embedded software systems with and without virtualization approach, in which two levels of maintenance are performed. We carried out the close-form solutions and also validated the numerical result with SHARPE tool result. We have shown that combining software rejuvenation and virtualization technology can be a partial contribution for enhancing the reliability of operational embedded software systems. Virtualization can provide some attractive benefits to embedded systems however there are some limitations. Future work will include experiments with an implementation under real world conditions to verify the practical efficacy of our approach.

Acknowledgments

This research was supported by Korea Aerospace University under contract No. 2008-01-009, “A Study for Behavior Control of Multiple Robot Systems Using the Indoor GPS”.

References

[1] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K.S. Trivedi, K. Vaidyanathan, and W.P.Zeggert. Proactive management of software aging. IBM Journal of Research and Development, vol. 45, no. 2, pp. 311–332, 2001.

[2] G. Heiser, The role of virtualization in embedded systems, In Proc. of the 1st workshop on Isolation and Integration in embedded systems, 2008, ISBN:978-1-60558-126-2, pp: 11-16.

[3] http://www.embedded.com/

[4] http://www.ok-labs.com/

[5] S. Nanda and T. Chiueh. A survey on virtualization technologies, Stony Brook University, Tech. Rep. TR-179, Feb 2005.

[6] Software Rejuvenation. Department of Electrical and Computer Engineering, Duke University Online Available: http:// www.software-rejuvenation.com/.

[7] V. Sundaram, S. HomChaudhuri, S.Garg, C. Kintala, and S. Bagchi. Improving dependability using shared supplementary memory and opportunistic micro rejuvenation in multi-tasking embedded systems. In Proc.13th Pacific Rim international Symp., on Dependable Computing (PRDC 2007), Washington, DC, December 17-19, 2007. IEEE Computer Society Press, Vol:, pp:240-247.

[8] K. S. Trivedi, SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator. DSN 2002: 544.

[9] K. Vaidyanathan, D. Selvamuthu, K. S. Trivedi. Analysis of inspection-based preventive maintenance in operational software systems. In Proc. 21st IEEE Symp., on Reliable Distributed System (SRDS’02), Suita, 13-16 October 2002. ISBN:0-7695-1659-9, pp-286-295.

Thandar Thein received M.Sc. (Computer Science) and Ph.D degrees in 1996 and 2004, respectively from University of Computer Studies, Yangon, Myanmar. Currently she is doing Post Doctorate Research in Korea Aerospace University. Her research interests include Ubiquitous Sensor Network Security, Security Engineering, and Network Security and Survivability.

Sung-Do Chi received BS and MS degrees in 1982 and 1984, respectively, in electrical engineering from Yonsei University, Seoul, Korea, and a PhD degree in electrical and computer engineering in 1991 from the University of Arizona, Tucson. From 1984 to 1986, he was a part-time instructor at the Yonsei University. He also worked as software engineer at the Digital Equipment Corporation, Seoul Branch. His research interests include traffic modeling, model-based reasoning, intelligent system design methodology, discrete-event system modeling and simulation; simulation based artificial life, computer security and Biotechnology.

Jong Sou Park received the M.S. degree in Electrical and Computer Engineering from North Carolina State University in 1986. And he received his Ph.D in Computer Engineering from The Pennsylvania State University in 1994. From 1994 - 1996, he worked as an assistant Professor at The Pennsylvania State University in Computer Engineering Department and he was president of the KSEA Central PA, Chapter. He is currently a full professor in Computer Engineering Department, Korea Aerospace University. His main research interests are information security, embedded system and hardware design. He is a member of IEEE and IEICE, and he is an executive board member of the Korea Institute of Information Security and Cryptology, and Korea Information Assurance Society.