Compositional Fluid Stochastic Petri Net model for operational software system performance

A. Bobbio*, S. Garg†, M. Gribaudo‡, A. Horváth‡, M. Sereno ‡, and M. Telek§
*Dipartimento di Informatica,Università del Piemonte Orientale, Alessandria, Italy
†Yahoo! Labs, Bangalore, India
‡Dipartimento di Informatica, Università di Torino, Italy
§Department of Telecommunications, Technical University of Budapest, Hungary

***Abstract*—Software systems experience gradual performance degradation due to several reasons and different preventive and corrective techniques can be applied to restore their performance level. This paper presents a unified model to describe the behavior of long running software systems with performance degrading factors such as system aging and various recovery techniques such as rejuvenation, checkpointing, rollback recovery, restart and replication.**

The proposed unified model is described as a fluid stochastic Petri net (FSPN). The FSPN formalism offers a descriptive language that allows a compact and precise description of the model behavior. Various analysis methods can be applied to obtain numerical results for the performance indices of interest. We illustrate the use of the model by means of a simple numerical example which captures rejuvenation, restart and replication.

I. Introduction

Chillarege [1] defined a software failure to be when *"The customer's expectation has not been met and/or the customer is unable to do useful work with the product"*. Under the broad umbrella of this definition, commercial software systems exhibit various malfunctions, performance degradation or outright failures. These are either due to the faults that arise in the hardware or in the software code or in error conditions which arise because of hardware / software fault combinations. Given the current hardware and software complexity, the forecasting and handling of software system malfunctions is a fundamental and challenging task. Software bugs may assume various forms, as surveyed in [2]. Even if the debugging and testing phase is aimed at removing bugs, residual faults still remain. Furthermore, the dynamic interaction of the running software with the hardware may result in degradation of software performance or even in a complete failure. This phenomenon is usually referred to as *software aging* [3], [4], [2].

Because of varied causes of faults, which induce performance degradation in software systems, appropriate preventive and corrective mechanisms are employed to counteract the effects in which the faults manifest. For example, when the time to complete a transaction or a task becomes too long with respect to user's expectation, an obvious and simple remedy is to restart the application [5], [6]. A common way for preventing the loss of work of long-running programs in the presence of failures, is to resort to checkpointing with rollback recovery. Checkpoint techniques have been studied and deployed since the pioneering work of Young [7] and a survey can be found in [8].

Software rejuvenation is a preventive maintenance action aimed at restoring the system to a "clean" state before the effect of software aging manifests as a failure. Since the source of the problem is typically unknown, bug-fix is not possible or not convenient and the only available solution is to periodically stop processing and restart the system environment and the software itself. Rejuvenation was first proposed by Huang et al. [9], and has been subsequently the object of extensive research. For an extensive list of references see: [4], [10]).

Proactive and reactive techniques to counteract software system failures are non-exclusive. In fact, they effectively complement each other to enhance system availability and correct software execution. Hence, a unified modeling framework is beneficial to capture both proactive and reactive techniques and to model inter-dependencies between these techniques. In [11], the completion time of a program is minimized by jointly using both checkpointing and rejuvenation. [10] includes rejuvenation, restoration and checkpointing and [6] considers, but separately, restart, rejuvenation and checkpointing.

The present paper is an extension of [12]. In this paper, the central idea is to provide a compositional reusable block model, in which, different components of a software system and the possible recovery techniques are presented as blocks in isolation. These can then be composed according to the actual specification of the system under study. In this sense, prior work in modeling would become specific compositions of our general modeling framework. The proposed modeling language is that of Fluid Stochastic Petri Nets (FSPN) [13], [14], since FSPNs combine discrete and continuous random variables in the same formalism.

Section II presents a narrative on the dynamics of a software system with degradation, rejuvenation, self restoration, restart and checkpointing. In Section III we describe the FSPN models of individual blocks and illustrate how to compose them according to the software system behavior. In Section IV, we present the numerical analysis of a simple system with rejuvenation, restart and replication, using a composition of FSPN modules.

II. System description

As there are multiple potential causes of degradation and failure and, accordingly, multiple potential combinations of

978-1-4244-3417-6/08/$25.00 2008 IEEE

reactive and proactive techniques for counteraction, it is important to describe the scope of the terms before we delve into the FSPN modeling framework. Below, we list and describe the modeling assumptions (system behavior) included in our compositional approach.

- *Degradation*. We assume that the degradation process, which models the phenomenon of software aging, can be represented by two time-dependent continuous variables, whose variations in time measure the level of the degradation [15]. The first describes the degradation of the hardware (also called node), while the second, the degradation of the process under execution. The way these quantities change can depend on various state variables, for example, the number of jobs in the system, the actual degradation levels themselves, the fact that a self restoration process is being executed, the time elapsed since the last rejuvenation or crash, etc.
- *Rejuvenation*. We assume that the decision of performing a rejuvenation may depend on the degradation level [15] and on the time spent since the last renewal event, or, alternatively, rejuvenation is performed after an assigned number of checkpoints [10]. It is natural to assume that a rejuvenation always forces a checkpoint, otherwise the work already completed since the last checkpoint is lost.
- *Work*. A continuous quantity, which captures the amount of work done by the system. The work is occasionally saved by a checkpoint or rejuvenation. If a crash occurs the work done by the system not saved yet is lost.
- *Time*. It is also a continuous quantity that keeps track of the time elapsed since the last occurred event, i.e. checkpoint, crash or rejuvenation and is needed to model dependencies as well as to calculate measures of interest.
- *Checkpoint*. When a checkpoint occurs the work done by the system not saved yet is saved.
- *Crash*. We distinguish between the crash of the system and crash of the process. When the system crashes, the work done by the system not saved yet by a checkpoint is lost. A crash initiates a recovery action that may or may not be successful. When it is successful, a renewal event occurs, i.e., the degradation level of the system gets reset. Following [10] a crash may be *active* when it is detected immediately or *passive* when the detection is deferred to the next checkpoint.
- *Self Restoration*. By self-restoration, we mean the actions do not cause down-time in the system but result only in performance overhead. When in progress, self-restoration continually decreases the degradation level. This mechanism is intended to model, for example, a garbage collector.
- *Workload*. It is used to represent the arrivals and departures of jobs. The service time may depend on the degradation level and on the number of customers in the system. We assume that the number of customers that can be accepted by the system is limited by a buffer of finite size. When the buffer is full or during a crash or a rejuvenation the arrival process is stopped. On the other hand, the service stops during checkpoints, rejuvenations, restarts, and crashes.
- *Restart*. For a given task, when the system response time has a decreasing hazard rate, preempting the task and retrying might improve the perceived system performance. The restart event influences only the task under service. It does not affect the degradation level of the system.
- *Replication*. Since we distinguish between two kinds of crashes, system crash and process crash, we assume that the system is able to survive a given number of process crashes by replicating the process. After the failure of a given number of replicas a system crash occurs.

III. FSPN MODELS

We apply the FSPN formalism to model the above system behavior. We refer to [13], [12], [14] for details of this formalism. For what concerns the notation, we use symbol p_j to indicate a standard PN place (with a discrete number of tokens) and c_i to indicate a fluid (continuous place) whose fluid level is denoted by the real positive variable x_i with the same index. Capital T indicates stochastically timed transitions with failure rate $F(\bullet)$ that can be function of the time, of the number of tokens in some discrete place as well as the value of some fluid level x. Small t indicates immediate transitions. Fluid places are drawn as double circles and fluid arcs by pipes. The instantaneous flow rate for fluid arcs is denoted with $R(\bullet)$ and may depend on discrete as well as continuous elements of the FSPN.

For each one of the features listed in the previous section we propose a FSPN block model that can be combined with other blocks to compose the overall system model. Blocks can be combined by sharing transitions (either immediate or timed). Each block indicates which are the output transitions (only) than can be shared with other blocks. Furthermore, triangular icons with the same label are used to indicate inhibiting effects among blocks without replicating the whole block structure.

The overall block structure and the block dependency relations are depicted in Figure 1.

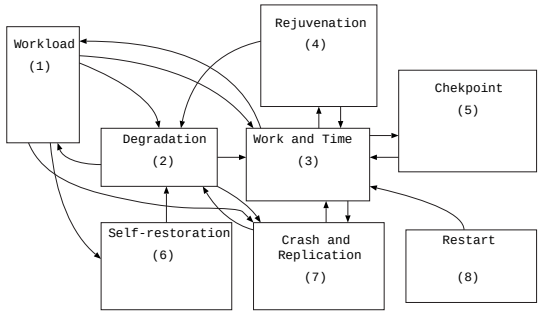

Fig. 1. Block model of software system with rejuvenation, checkpointing, self restoration and replication.

Workload subnet (Figure 2): Arrival of jobs to a buffer of size k is represented by transition T_{11}. This transition is blocked when the buffer is full by means of an inhibitor arc with multiplicity k. Moreover, T_{11} is blocked when the system itself is down or it is under rejuvenation. This fact

is represented by the two triangular labels, n and r, which refer to places of other submodels (r, for example, refers to place p_{42} in Figure 5). Service of jobs is represented by transition T_{12} which is blocked if the node or the process is crashed (triangular labels n and p), or if restart, rejuvenation or checkpointing is under execution (triangular labels s, r and c). The firing rate $F(m, x_{21}, x_{22})$ of T_{12} depends on the number of jobs in the system m, and on the degradation levels x_{21} and x_{22} (described in the next submodels). Further dependencies on other state variables could also be included.

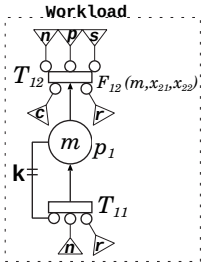

Fig. 2. FSPN Model of the workload.

Degradation subnet (Figure 3): The degradation subnet is divided in two parts representing the system degradation and the process degradation, respectively. We assume that the system degradation level can be measured by a single continuous index x_{21} representing the level of the continuous place c_{21}. Transition T_{21} pumps fluid into place c_{21} at a rate $R_{21}(m)$, i.e., the change of the degradation level depends on the number of jobs m present in the system. Degradation of the process under execution is measured by the fluid variable x_{22} representing the level of fluid place c_{22}. The change of its level, modeled by T_{22} with flow rate $R_{22}(m, x_{21})$, depends not only on m but on the actual node degradation level, x_{21}, as well. More complex dependencies could also be modeled; for example, the intensity of the degradation could depend on the time elapsed since the last crash by making the flow rate function also dependent on the fluid variable x_{32} (see *Work & Time* block) $R_{22}(m, x_{21}, x_{32})$. System degradation is stopped in case of system crash and process degradation is stopped in case of both system and process crash (triangular labels n and p).

The *Degradation* block can be connected to other blocks as depicted in Figure 3. The presence of transitions of other subnets in Figure 3 is due to the fact that the level of places c_{21} and c_{22} are changed by events of other submodels. For example, firing of transition T_{41}, which models the end of a rejuvenation action (see Figure 5), empties both c_{21} and c_{22}, i.e., sets back to 0 the degradation levels. Transition T_{61} instead models the fact that self restoration decreases the process degradation level by gradually depleting place c_{22} with a flow rate $R_{62}(m, x_{22})$.

Work and Time subnet (Figure 4): Transition T_{31} becomes enabled each time there is a renewal action and is disabled when the system is performing restart, rejuvenation or checkpointing (labels s, r and c), or when a crash occurs (labels n and p). Thus T_{31} remains enabled until the system works correctly. This subnet describes the time elapsed and the work

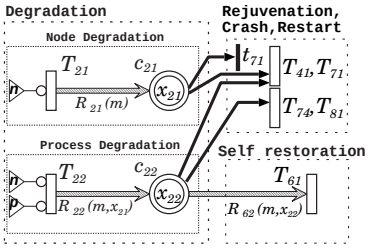

Fig. 3. FSPN Model of the system degradation.

performed between two renewals. Transition T_{31} pumps fluid in place c_{32} with rate $R_{32} = 1$, hence the fluid level x_{32} counts the elapsed time since last renewal event (i.e., crash or rejuvenation), whereas c_{32} is filled with rate $R_{31}(m, x_{21})$ and the level x_{31} represents the work of the system not saved yet by a checkpoint. The flow rate $R_{31}(m, x_{21})$ which determines the work accumulation, depends on the number of jobs in the system and the actual system degradation level (further dependencies could also be included). Place c_{31} is flushed out by the firing of transitions T_{41} (rejuvenation forces checkpoint), T_{51} (checkpoint without rejuvenation), T_{71} and t_{71} (occurrence of a system crash). Place c_{32} is flushed out by firing of transitions that represents renewal events (rejuvenation or crash).

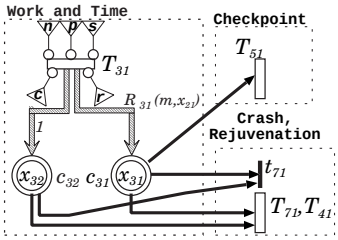

Fig. 4. FSPN Model of the work done by the system.

Rejuvenation subnet (Figure 5): We consider two possible modeling alternatives. In Figure 5a) the decision to initiate a rejuvenation is represented by the timed transition T_{42} whose firing time is independent of other recovery measures, while in in Figure 5b) we model the case considered in [10] where the rejuvenation takes place after n checkpoints and the immediate transition t_{42} fires when n tokens are deposited in p_{43}. In both cases transition T_{41} models the duration of the rejuvenation and its firing rate can depend on various state variables of the model (for example, degradation levels or the time elapsed since the last crash or rejuvenation). We assume that when the system performs a rejuvenation, a checkpoint is forced but not vice versa. Firing of T_{41} sets to zero the degradation levels (fluid places c_{21} and c_{22}, see Figure 3), the work accumulated and the time elapsed since the last renewal event (fluid places c_{31} and c_{32}, Figure 4), and the time elapsed since the last checkpoint (fluid place c_{51}, Figure 6). As represented by the triangular labels r in other submodels (connected to place p_{42} in Figure 5), several activities are blocked when a rejuvenation is under execution (for example, service of customers, transition T_{12} in Figure 2). In case of Figure 5a) rejuvenation is not possible when restart, checkpointing or self

restoration is under execution (triangular labels s, c and e). System and process crash not only inhibit but also preempt execution of rejuvenation (triangular labels n and p). As a result of blocking other activities and setting fluid levels to zero, the system is good as new after rejuvenation.

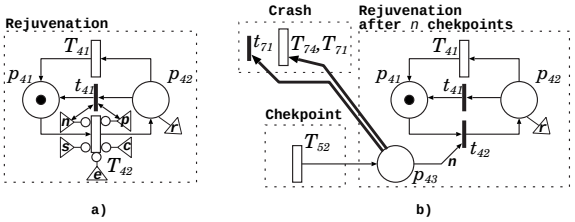

Fig. 5. FSPN Model of the rejuvenation action.

Checkpoint subnet (Figure 6): Fluid place c_{51} measures the time elapsed since the last checkpoint. Transition T_{52} represents the decision that a checkpoint is taken. In this submodel we depict explicitly that the decision depends on the amount of time elapsed since the last checkpoint (fluid level x_{51}) and on the amount of work accumulated since the last checkpoint (fluid level x_{31}). Transition T_{51} models the checkpoint overhead (although not shown explicitly, the overhead can also depend on x_{51}, x_{31} and/or other state variables). Firing of T_{51} flushes out places c_{51} (time elapsed since last checkpoint) and c_{31} (work accumulated since last checkpoint). Rejuvenation and system crash reset the time elapsed since the last checkpoint (transitions T_{41}, T_{71} and t_{71}). As represented by the triangular labels c in other subnets (connected to place p_{52} in Figure 5), several activities are blocked when a checkpoint is under execution. Checkpointing itself is not possible when restart or rejuvenation is under execution (triangular labels s and r). System and process crash do not only inhibit but also interrupt execution of checkpointing (triangular labels n and p).

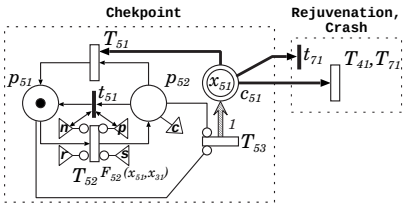

Fig. 6. FSPN Model of checkpointing.

Self Restoration subnet (Figure 7): Transition T_{62} represents the beginning of self restoration while T_{61} its duration. Firing rates of these transitions can depend on other state variables of the system (for example, degradation levels). During the self restoration process degradation is decreased by the pumping out action enabled by transition T_{61} on place c_{22} (see Figure 3). When the self restoration mechanism is active, restart and rejuvenation cannot be started (triangular labels e). Self restoration cannot be started during restart, rejuvenation and checkpointing. System and process crash inhibit and interrupt execution of self restoration.

Crash and Replication subnet (Figure 8): Transition T_{71}

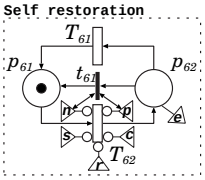

Fig. 7. FSPN Model of system self restoration.

and T_{74} model the crash of the system and the crash of the process under execution, respectively. In the actual setting, their firing rates depend on the number of jobs and on the degradation levels but other state variables could be taken into account as well. We assume that crashes are not observed immediately and the time to discover the crash is modeled by T_{72} for system crash and by T_{75} for process crash. Recovery is modeled by T_{73} for system crash and by T_{76} for process crash. The firing rate of these transitions can depend on various state variables. Recovery from process crash is either successful (T_{76}) or unsuccessful (T_{77}). Number of process crashes without successful recovery is counted in place p_{76} and after r such unsuccessful recoveries the system crashes (firing of t_{71}). Triangular labels n and p are used to block activities in other submodels during crash.

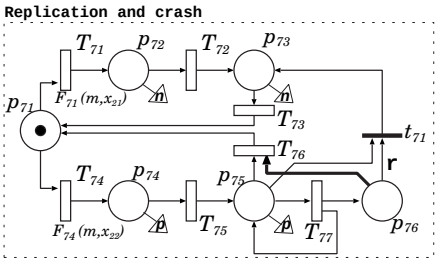

Fig. 8. FSPN Model of crash and replication.

In a recent paper Okamura and Dohi [10] assume that a system crash may manifest in an *active* mode (that is immediately detected activating a rollback recovery from the last checkpoint), and in a *passive mode* in which the detection is deferred until the successive checkpoint. The two failure modes can be modeled by modifying the subnet of Figure 8 in the new subnet of Figure 9. Timed transition T_{71} represents the system crash, that with probability p manifests in the *active* mode (immediate transition t_{72}) and with probability $1-p$ in the *passive* mode (immediate transition t_{74}) and the recovery is deferred until the next checkpoint.

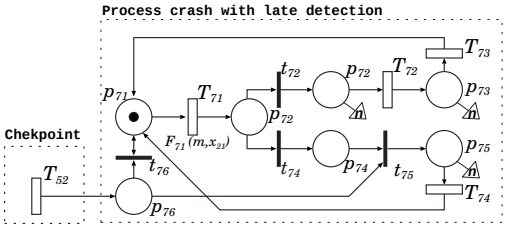

Fig. 9. FSPN Model of system crash with 2 failure modes after [10].

Restart subnet (Figure 10: Transition T_{82} represents the

beginning of restart of a process while T_{81} its duration. Firing rates of these transitions can depend on other state variables of the system (for example, degradation levels). Restart sets to zero the process degradation level by transition T_{81} (see Figure 3). During the restart mechanism the system cannot start rejuvenation, self restoration, checkpointing and the service of the jobs are blocked (triangular labels s). System and process crash inhibit and interrupt execution of self restoration.

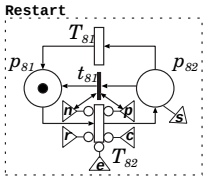

Fig. 10. FSPN Model of the restart mechanism.

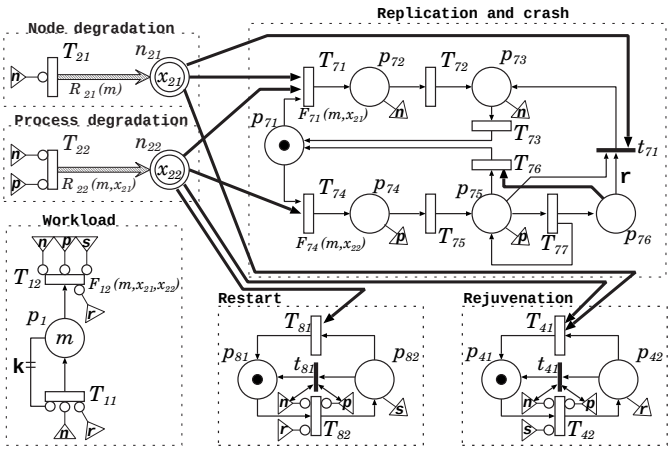

Fig. 11. FSPN model of degrading software system with rejuvenation, restart and replication.

A. Model composition and analysis

The individual FSPN blocks illustrated in the previous section can be combined, according to the user needs, to model the characteristics and the property of the system at hand. A software system might be provided with periodical rejuvenation but not checkpointing or viceversa; a system might not have self-restoration capabilities or the degradation level might not be measurable. This is why the compositional capability offers to the modeler the required level of flexibility.

Furthermore, FSPN may provide a set of performance measures that encompasses those that can be evaluated in discrete SPN models. In fact, we can define measures connected to the discrete part of the FSPN (*discrete performance measures*) and, in addition, measures connected to the continuous part (*continuous performance measures*) [12].

IV. Numerical Example

In order to illustrate the use of the model and the associated analysis, we present a simple compositional model with related performance measures. The considered model contains a subset of the features introduced in the previous section, namely, degradation, replication, restart and rejuvenation. Figure 11 presents the FSPN description of the whole model depicting also the way of connecting the submodels.

The model was solved analytically in transient time resorting to the solution method provided in [16]. The numerical values of the model parameters used in the computations are reported in Table I

As a sake of illustration, we have computed a sample of both discrete and continuous performance indices as a function of the time. As an example of continuous measures we have reported in Figure 12 the mean system degradation level and the mean process degradation level. The mean system (process) degradation level at time t is computed averaging the value of the continuous random variable x_{21} (x_{22}) over all the model states. The ripple in the mean process degradation level is mainly due to the restart mechanism.

As an example of discrete measures we have reported in the same Figure 12 the system (process) crash probability, i.e. the probability that discrete place p_{72} (p_{74}) is marked at time t (notice the logarithmic scale on the vertical axis).

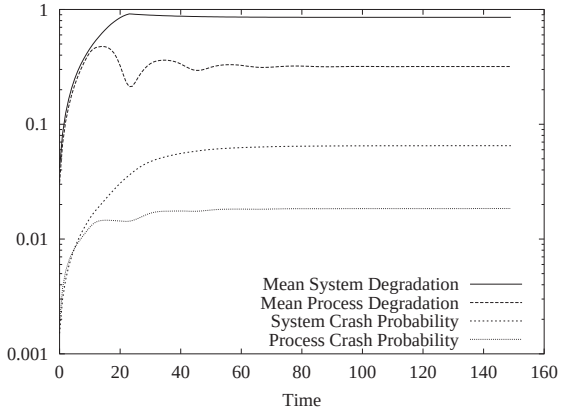

Fig. 12. Transient probability of crash failure.

Figure 13 reports again (in linear scale) the mean process degradation level, and in the same scale the probability of normal operation (token in places $p_{71} \wedge p_{81} \wedge p_{41}$), of restart (token in place p_{82}), of rejuvenation (token in place p_{42}) and of crash (probability of places $p_{72} + p_{73} + p_{74} + p_{75} + p_{76}$). Notice that the last four probability values sum to 1 at any time t. The probability of normal operation can be interpreted as the system availability that tends to the steady state value as the time goes to infinity.

V. Conclusions

The main contribution in this paper is the FSPN based modeling framework, which is modular and able to capture various combinations of fault-tolerance techniques, reactive and proactive to enhance system availability and correct software behavior. The choice of FSPN in each module enables incorporating discrete and continuous variables which can be time dependent. Further, the composition of the FSPN modules via outgoing transitions allows rapid evaluation of different combinations of the techniques. We illustrated the flexibility

Parameters	Activities
m	Number of parallel processes ($m \leq 3$)
r	Cold replication maximum number of retries ($r \leq 3$)
x_{21}	node degradation level ($0 \leq x_{21} \leq 1$)
x_{22}	process degradation level ($0 \leq x_{22} \leq 1$)
Transitions	**Activities**
$T_{11} = 0.2$	arrival of customers to the system
$F_{12}(m, x_{21}, x_{22}) = 1/(m(x_{21} + x_{22} + 1))$	service of customers
$F_{71}(m, x_{21}) = m(8x_{21} + 1)/2400$	system crash rate
$F_{74}(m, x_{22}) = m(8x_{22} + 1)/1920$	process crash rate
$T_{72} = 1$	system crash identification rate
$T_{75} = 2$	process crash identification rate
$T_{73} = 0.1$	system crash repair rate
$T_{76} = 0.32$	process crash repair rate (successful restart)
$T_{77} = 0.08$	process crash repair rate (unsuccessful restart)
$T_{81} = 1$	1 / restart length (rate)
$F_{82}(x_{22}) = \{x_{22} > 0.75, 100, 0.0001\}$	restart rate
$T_{41} = 0.2$	1 / rejuvenation length (rate)
$T_{42} = 1/360$	rejuvenation rate
Fluid transitions	**Activities**
$R_{21}(m) = m/128$	Increase of system degradation level
$R_{22}(m, x_{21}) = m(x_{21} + 1)/96$	Increase of process degradation level

TABLE I
MODEL PARAMETERS OF THE FSPN OF FIGURE 11

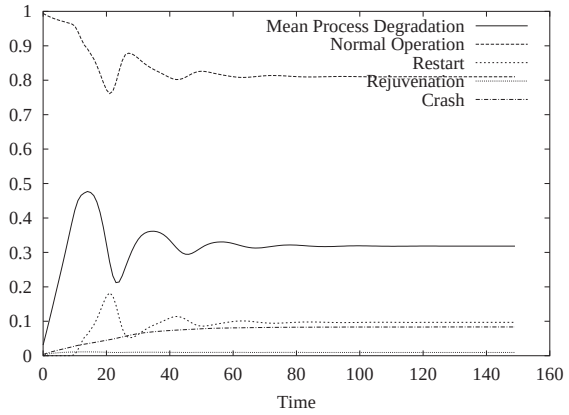

Fig. 13. Transient probability of various system degradation and restoration events.

of the framework by constructing a representative model and gave numerical solution for the model.

As new age of Internet and Web services pervade, it is becoming increasingly common to have "software system" be not composed of a software process running on a software node. Rather, the system is now a set of software modules running on a set of nodes with failure interdependencies. One potential extension of the framework is to extend the formalism to capture this. There has been prior work in evaluating dependability of component based software systems. However, a unified framework, which allows for composition of fault-tolerance techniques on a per component basis of a software system has not been explored.

ACKNOWLEDGMENTS

This work have been partially supported by grant MIUR-PRIN No. 2007J4SKYP and OTKA No. K61709.

REFERENCES

[1] R. Chillarege, "What is software failure ?" *IEEE Transactions Reliability*, vol. 45, pp. 354–355, 1996.

[2] M. Grottke and K. Trivedi, "Fighting bugs: Remove, retry, replicate and rejuvenate," *IEEE Computer*, pp. 107–109, February 2007.

[3] A. Avritzer and E. J. Weyuker, "Monitoring smoothly degrading systems for increased dependability," *Journal of Empirical Software Engineering*, vol. 2, no. 1, 1997.

[4] S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of preventive maintenance in transaction based software systems," *IEEE Transactions on Comupters*, vol. 47, pp. 96–107, 1998.

[5] A. van Moorsel and K. Wolter, "Analysis of restart mechanisms in software systems," *IEEE Transactions Software Engineering*, vol. 32, no. 8, pp. 547–558, 2006.

[6] K. Wolter, "Stochastic models for restart, rejuvenation and checkpointing," Habilitation Thesis, Humboldt-University, Institut Informatik, Berlin, Tech. Rep., 2008.

[7] J. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, no. 9, pp. 530–531, 1974.

[8] V. Nicola, "Checkpointing and modeling of program execution time," in *Software Fault Tolerance*, M. Lyu, Ed. John Wiley & Sons, 1995, pp. 167–188.

[9] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: analysis, module and applications," in *Proceedings of the 25-th Fault Tolerant Computing Symposium (FTCS-25)*, 1995, pp. 381–390.

[10] H. Okamura and T. Dohi, "Analysis of a software system with rejuvenation, restoration and checkpointing," in *Proc ISAS 2008*. Springer Verlag - LNCS, Vol 5017, 2008, pp. 110–128.

[11] S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi, "Minimizing completion time of a program by checkpoint and rejuvenation," in *Proc. 1996 ACM SIGMETRICS Conference*, Philadelphia, PA, May 1996, pp. 252–261.

[12] A. Bobbio, S. Garg, M. Gribaudo, A. Horvát, M. Sereno, and M. Telek, "Modelling software systems with rejuvenation restoration and checkpointing through Fluid Stochastic Petri Nets," in *PNPM '99*. IEEE CS Press, Sept 1999, pp. 82–91.

[13] G. Horton, V. Kulkarni, D. Nicol, and K. Trivedi, "Fluid stochastic Petri nets: Theory, application and solution techniques," *European J Operational Research*, vol. 105, pp. 184–201, 1998.

[14] M. Gribaudo, M. Sereno, A. Horváth, and A. Bobbio, "Fluid stochastic petri nets augmented with flush-out arcs: Modelling and analysis," *Discrete Event Dynamic Syst*, vol. 11, pp. 97–111, 2001.

[15] A. Bobbio, M. Sereno, and C. Anglano, "Fine grained software degradation models for optimal rejuvenation policies," *Performance Evaluation*, vol. 46, pp. 45–62, September 2001.

[16] M. Gribaudo and A. Horváth, "Fluid stochastic petri nets augmented with flush-out arcs: A transient analysis technique," *IEEE Transactions Software Engineering*, vol. 28, pp. 944–955, 2002.