An analysis of hardware and software availability exemplified on the IBM 3725 Communication Controller

by P. I. Pignal

Because of the growing commercial, governmental, and scientific requirements for system availability, evaluating this factor has become increasingly important. This paper presents a unified approach to hardware and software availability of a system in the operational phase. The aim is to evaluate the availability in a given time interval, to show how to improve it, and to determine the probability that a specified level is met over the period. The inputs are the failure and repair rates of the system elements, and the functional relationship between them. Field tracking provides the failure and repair data, and Markov-chain techniques make it possible to construct, reduce, and solve the model. Availability is computed by the program package System Availability Estimator (SAVE). The model has been used and validated with actual field data for the IBM 3725 Communication Controller.

1. Introduction

This paper is a continuation of previous work dedicated to the modeling of computer and communication systems and the architecture and design of highly available communication systems [1–3]. The present purpose is to show a real example with its quantitative results. The paper describes the method used in IBM to evaluate, improve, and guarantee the combined hardware and software availability of the IBM 3725 Communication Controller. The communication controller is considered in operation, not during testing. The chosen measure of availability is the expected interval availability, evaluated for one-month intervals.

The method used to improve the availability consists of two steps. The first phase is the identification of the components—hardware and software—that most contribute to system unavailability. These bottlenecks are not

necessarily the less reliable components. The failure of a vital reliable component may indeed cause a greater unavailability of the system than the failure of a harmless unreliable component. The second phase consists in undertaking appropriate selective actions, such as replication of a given component or improvement of the error-detection and fault-isolation capability of another component. To guarantee the computed value of the availability within a specified period, we determine the probability that the availability is greater than a given level over the period.

Due to the high availability of the IBM 3725, an analytical method, the Markovian method, is chosen to solve the model. Monte Carlo simulations are not used, even if they are more flexible and even if variance-reduction techniques could reduce the run lengths. Markov models have been widely used to predict hardware reliability and availability. Software models, such as reliability growth models [4, 5], usually deal with the testing phase and are typically not Markovian, and distinguish software from hardware [6]. The dichotomy between hardware and software can be justified by emphasizing their different evolution in time: Hardware wears out according to a bathtub-shaped failure-rate curve, whereas software fails because of latent errors, depending on the use and management of system resources and the environment [7]. As an example, workload affects software more than hardware, and a nonlinear increase in software failures with the amount of interactive processing has been observed on the IBM 3081 [8]. However, workload also affects hardware reliability. Besides, from the user viewpoint, there is no difference between hardware and software failures: The system is functioning or it is not. Furthermore, since many failures occur in the hardware–software interface [8], a unified measure is necessary. Thus, there are several combined hardware and software reliability analyses and models [9, 10].

During the operational phase, hardware failure and repair distributions are commonly, though not necessarily, approximated to exponential distributions involving constant failure and repair rates. Due to the heterogeneity of the environment and use, and the occurrence of new product releases, the software failure and repair rates of the IBM 3725 Communication Controller may decrease or increase throughout the operational phase. However, the variations are slow. Chi-square and Kolmogorov–Smirnov goodness-of-fit tests justify the assumption of constant hardware and software failure and repair rates, within one-month intervals. Under this assumption, the system evolves within each one-month interval as a finite-state homogeneous continuous-time Markov chain. Computing the expected interval availability is then possible over periods of one month. The failure and repair data from the end of the current month are used to predict the availability of the next month. At the end of that month, the availability of the month is reviewed with the month's data, and the availability of the third month is predicted; and so on. This procedure allows the representation of a complex situation actually occurring in the field.

The program package called the System Availability Estimator (SAVE) [11] automatically generates the Markov chain from the input modeling language. However, the IBM 3725 Communication Controller has a large number of elements. Moreover, the size of the model increases exponentially with the number of elements and thus rapidly becomes intractable. To decrease this size, we use hierarchical modeling, thereby introducing a slight approximation.

The study of hardware and software failure and repair data is described in Section 2. The model construction, reduction, and solution are presented in Section 3. The methodology for availability improvement is detailed in Section 4. Section 5 explains the method used to guarantee the value of the availability, and gives quantitative results. A summary of the major points covered in the paper, a brief discussion of their implications, and an overview of a further study are given as concluding remarks in Section 6.

2. Study of failure and repair data

The first step of a model is data collection. In particular, it is essential to track software data [12], since projections of software failures are difficult. The emergence of measurement products, such as IBM's System Availability Management—which detects, gathers, and manages hardware- and software-availability data—confirms this point.

Several definitions have been proposed to quantify the quality of the service delivered by a system [13, 14], and to distinguish the degrees of defects. From the viewpoint of the end user, the behavior of a system is perceived as a delivered service. Thus, when the delivered service deviates from the specified service, a failure occurs. A failure is the manifestation of one or several errors in the service, and an error is the manifestation of one or several faults in the system [14]. The moments of fault occurrence and failure occurrence are separated by a time interval, the fault latency, which may vary from a few milliseconds to several months. The event effectively perceived by the user is the occurrence of a failure. Hence, only the failure aspect is considered in the remainder of this paper. The basic characteristics of a failure are its failure rate and its repair rate. Hardware failure and repair rates are defined with regard to time [15]. Currently, software failure rates may be defined with regard to the number of lines of code (released, or new and modified), the number of functions, or the time. Here, software failure and repair rates are defined with respect to time, to evaluate hardware and software availability in equivalent terms.

Hardware failures may be permanent or soft; software failures are permanent. A permanent failure is due to a

Table 1 Failure and repair data.

RA_{I_i}	Number of identified RA on component i	(per month)
RA_U	Total number of unidentified RA	(per month)
RA_{M_i}	Number of microcode RA on component i	(per month)
P_{S1}	Number of NCP problems of severity 1	(for T hours)
P_{S2}	Number of NCP problems of severity 2	(for T hours)
E_i	Error detection rate of component i	
P	Power-on hours per month	(in hours)
DOI_{I_i}	DOI for an identified failure on i	(in hours)
DOI_U	Total DOI for unidentified failures	(in hours)
DOI_{M_i}	DOI for the microcode component i	(in hours)
DOI_{S1_i}	Severity 1 DOI for the software component i	(in hours)
DOI_{S2_i}	Severity 2 DOI for the software component i	(in hours)

Table 2 Calculation of failure and repair rates.

COMPONENT: i IN HARDWARE
FAILURE MODE: Identified, Unidentified
FAILURE RATE: RA_{I_i}/P,
$RA_U \cdot (1 - E_i)RA_{I_i}/[\Sigma_i (1 - E_i)RA_{I_i}T]$
REPAIR RATE: $1/DOI_{I_i}$, $1/(DOI_U \cdot DOI_{I_i}/\Sigma_i DOI_{I_i})$

COMPONENT: i IN MICROCODE
FAILURE RATE: RA_{M_i}/P
REPAIR RATE: $1/DOI_{M_i}$

COMPONENT: $i = NCP$
FAILURE MODES: Severity 1, Severity 2
FAILURE RATE: P_{S1}/T, P_{S2}/T
REPAIR RATE: $1/DOI_{S1_i}$, $1/DOI_{S2_i}$

permanent fault. It is continuous and stable. A soft failure is the result of either an intermittent fault (due to unstable or varying hardware) or a transient fault (due to temporary environmental variations) [16]. Permanent failures correspond to slow transitions, and soft failures to time-dependent fast transitions.

Permanent failures with slow transitions are approximately exponentially distributed, and can be modeled with a Markov process. Soft failures with fast transitions are more difficult to characterize. Their rates can be measured experimentally using fault-injection techniques. The transitions are not exponentially distributed; hence, a semi-Markov process is more realistic than a Markov process, because it does not impose limitations on the time the process spends in a given state. In a semi-Markov process, the occurrence of a transient failure is described as a transition between two states, with transition rates defined by distribution functions. The representation of an intermittent failure requires in addition the knowledge of the holding-time distribution related to the supplementary state in which the fault is not active [17]. Recovery procedures in the IBM 3725 Communication Controller allow it to tolerate soft failures with fast transitions. Therefore, these failures are not considered here. Such failures could be modeled. As an example, in the SURE [17] reliability program package of NASA, each transition related to a soft failure is described by the conditional mean and conditional standard deviation of the associated distribution, and the transition probability. The HARP package [18] allows the same modeling, as well as seven different ways of specifying fault- and error-handling behavior.

In the IBM 3725 Communication Controller, there are two classes of data, namely hardware data and software data. The hardware class contains physical components, such as memories, processors, buses, and power supplies. Hardware data may refer to identified or unidentified failures, due to the imperfect fault-isolation capacity of the controller. The software class contains the microcode functions of the different subsystems and the Network Control Program (NCP), which controls the whole communication controller. NCP data are divided into two subclasses, according to the severity of the failures.

The parameters that characterize the numbers of failures differ according to the class. The term "repair action" is used for the hardware class and the microcode subclass, whereas the term "problem" applies to NCP. Numbers of repair actions (RA), planned and unplanned, are given per machine and per month. Numbers of NCP problems are given per license and for a given period T. The repair data are expressed as durations of interrupt (DOI), which include, when appropriate, the different times such as travel time, repair time, part-procurement time, initial program load time, and recovery time. The duration of interrupt is defined in equivalent terms for the different classes. **Table 1** shows all the data available. The error-detection rate is estimated for each hardware component by an IBM design tool. All the other data are field data, either logged in a database or reported by customer engineers.

The basic parameters of a Markov chain to be used for the evaluation of the availability—the failure and repair rates—must be defined with respect to the same time units. The hour is the most appropriate unit for the duration of interrupts. Hence, the repair rates, inverse of the duration of interrupts, are expressed per hour. Consequently, the failure rates are also defined per hour. **Table 2** shows how to compute, from the collected data, the failure and repair rates of a given component.

Failure data, which are given per month, need to be normalized. The failure rate of a component in a given failure mode is expressed as the number of failures of this component in a given failure mode per time unit, that is, the normalized ratio with respect to time of its number of repair

actions or problems. In the case of the unidentified failure mode of hardware components, the number of repair actions per component is unknown. Only the global number of unidentified repair actions is known. The breakdown per component is estimated as follows:

$$RA_{U_i} = RA_U(1 - E_i)RA_{I_i}/\sum_i (1 - E_i)RA_{I_i}.$$

The sum is computed for all N_c hardware components modeled.

The repair rate of a component in a given mode is the inverse of the corresponding duration of interrupt. In the unidentified mode, an estimator of the duration of interrupt per component is

$$DOI_{U_i} = DOI_U \cdot DOI_{I_i}/\sum_i DOI_{I_i}.$$

Thus, the failure and repair rates can be evaluated for each component in each failure mode from the available data.

3. Availability model

The IBM 3725 Communication Controller is modeled as a set of interconnected hardware and software components. There are a few hundred components to be modeled. The Markovian method involves an exponential growth of the state space with the number of components modeled. As an example, 10 different components create 2^{10} possible states, and 50 components create 2^{50} possible states. Thus, the size of the model must be reduced. So far, there are three main approaches to the large-state-space problem:

1. A structural decomposition, which consists in dividing the global system into physical subsystems, solving each subsystem separately, and combining the partial solutions to get the overall system solution. This is the approach used in the SHARPE [19] hierarchical modeling tool for reliability models.
2. A behavioral decomposition, which consists in separating the fault-occurrence and fault-handling behavior into distinct submodels, solving each submodel in the most appropriate way—for instance, simulating one submodel and solving the other submodel analytically, and incorporating the results of the fault-occurrence submodel into the fault-handling submodel. This approach is selected in the CAST [20], CARE III [21], and HARP [18] reliability-evaluation packages.
3. An approximate solution (using, for example, an aggregation technique), which consists in decomposing the original problem into smaller and more convenient subproblems, analyzing the subproblems separately, and solving the global problem through the iterated solution of its subproblems. The main methods are the iterative aggregation/disaggregation technique and the decomposition technique for nearly completely decomposable systems [22] in the case of steady-state analysis. For the transient analysis of stiff Markov chains, a method based on the conversion of a stiff Markov chain into a smaller nonstiff chain has been proposed [23].

To model the IBM 3725 Communication Controller, we use the structural decomposition at the lowest hierarchical level, that is, at the field-replaceable-unit level. The field-replaceable unit is the lowest level at which there are hardware failure and repair data. Physically, several field-replaceable units are grouped together to perform a given function, such as line interface or maintenance. Within each of these sets of atomic elements, we encapsulate those that are independent. Thus, each function is viewed as a set of capsules containing independent atomic elements. A capsule may contain from one to more than a hundred elements, an example of which is the power supply. The advantage of encapsulating independent elements is the possibility of using combinatorial models, such as fault trees and reliability block diagrams, which do not involve an exponential growth of the number of events with the number of elements, and which are easier to solve.

We have developed a program package called HEAVEN, which provides a block-diagram model for general structures, not necessarily series-parallel. In HEAVEN, each component modeled is labeled as the cumulative distribution function (CDF) for one of its dependability [14] measures, such as instantaneous availability, reliability, or time to failure, or one of its performance measures. The class of function chosen has the following form:

$$F(t) = \sum_i P_i(t) \cdot e^{Q_i(t)}, \tag{1}$$

where $P_i(t)$ and $Q_i(t)$ are polynomials of the positive real variable t (time). It is a generalization of the class of functions taken in SHARPE [19] and is closed under the convolution operations performed.

In this study, we assign to each field-replaceable unit its instantaneous availability. Assuming that the field-replaceable units are initially operational, the instantaneous availability of any of them is [15]

$$A_i(t) = [RR_i/(FR_i + RR_i)] + [FR_i/(FR_i + RR_i)]\, e^{-(FR_i+RR_i)t}, \tag{2}$$

where FR_i and RR_i denote respectively the failure and repair rate of the field-replaceable unit i.

The model gives as output the cumulative distribution function of the capsule as a whole, which has the form

$$A(t) = B + \sum_i C_i \cdot e^{-D_i \cdot t}, \tag{3}$$

where B and all C_i and D_i are independent of t.

Let D_j be the minimum of the D_i for $i = 1$ to n. Considering the dominant terms of $A(t)$, as t approaches infinity, we obtain

$$B + C_j \cdot e^{-D_j \cdot t}. \tag{4}$$

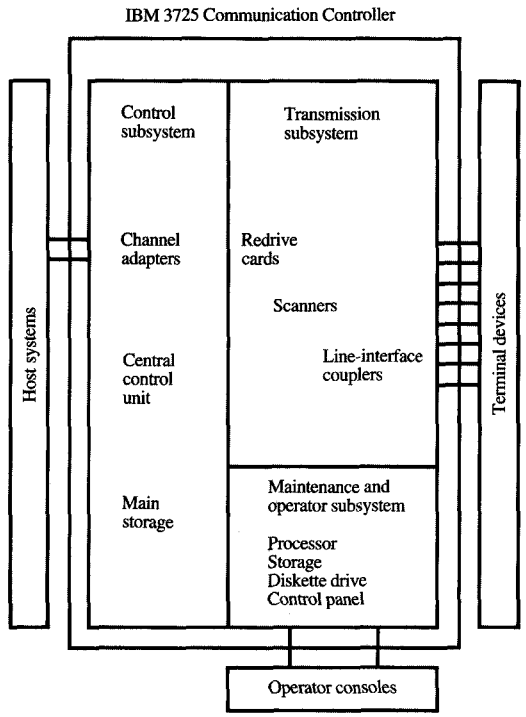

Figure 1

Structure of IBM 3725 Communication Controller.

Table 3 Components remaining after encapsulation.

Hardware components	
CA	Channel adapter
CCU	Central control unit
LIC(8)	8 line-interface couplers
MOSS	Maintenance and operator subsystem
PS	Power supply
RDV	Redrive card
SCAN	Scanner
Software components	
MOSSCODE	MOSS microcode
TSSCODE	Transmission subsystem microcode
NCP	Network control program

Equation 4 should be used in both failure modes (identified and unidentified) to determine the equivalent failure rate FR_e and the equivalent repair rate RR_e in each mode. To solve Equation 4, we introduce the probability P_e, such that at time $t = 0$ the system is operational with probability P_e. Then, the availability of the equivalent element is

$$A_e = [RR_e/(FR_e + RR_e)] + \{P_e - [RR_e/(FR_e + RR_e)]\}\, e^{-(FR_e+RR_e)t}.$$

Finally, we determine the equivalent failure and repair rates in each hardware failure mode by solving a set of three equations with three unknowns FR_e, RR_e, and P_e. We obtain for FR_e and RR_e

$$FR_e = B + C_j,$$

$$RR_e = B \cdot D_j.$$

In the IBM 3725 Communication Controller, there are three software components: the Network Control Program (NCP) running in the control subsystem, the microcode of the transmission subsystem (TSSCODE), and the microcode of the maintenance and operator subsystem (MOSSCODE). **Figure 1** gives an overview of the structure of the controller. The software failure and repair data are available for each of the three software components. Each software component has a specific interaction with a hardware capsule. Thus, each software component constitutes a capsule.

Consequently, at the end of the first hierarchical step, the system is modeled as a set of capsules with estimated failure and repair rates assigned to each capsule. The definition of the capsules from the field-replaceable units is given in a specific file in the model (CONCEN FILE). The capsules are the atomic entities in the SAVE model, where they are called components. The IBM 3725 Communication Controller is modeled with ten different SAVE components, of which seven are hardware components and three are software components, as shown in **Table 3**. Note that there are eight identical components of type LIC.

The behavioral decomposition is not used here. There are currently no estimates of the parameters required, such as the transient restoration, permanent coverage, simple-point failure, and near-coincident fault [18].

An approximate solution may be used in SAVE to solve models with more than 10 000 states, or simply to accelerate the computation time for smaller models. The method consists in decreasing the number of concurrent failures modeled, based on the observation that it does not affect the precision of the result significantly. In the IBM 3725 Communication Controller study, with one, two, or three concurrent failures modeled, the numbers of states are 21, 198, and 1116. The variations of system availability involved by modeling one instead of two concurrent failures, and two instead of three concurrent failures, are 1.61×10^{-7} and less than 10^{-9}, respectively. This shows that modeling more concurrent failures would not change the availability numbers. Hence, the system can be solved with reduced state space.

Once the model is reduced to a tractable size, we refine the failure and repair behavior of each component and define the interactions among the components. The refinement of

the failure behavior consists in describing the dormant mode of a component: the state in which a component is neither operational nor down, but is unable to operate because other components have failed [11]. Parameters which characterize the effectiveness of a recovery, such as coverage factors [13] and other parameters related to the fault-occurrence behavior, can be specified here. The refinement of the repair behavior consists in describing specific repairman classes (such as Field Engineer, Operator, or Software Retry) and specifying within each class a specific strategy [such as random-order service (ROS) or different priority levels].

Along with the refinement of the inner behavior of a capsule, we must define the interactions between the capsules. We model interactions of all three types between capsules, that is, hardware–hardware, software–software, and hardware–software. There are two types of interactions, operational dependency and repair dependency; both are modeled. They correspond to the fact that a given component may require another component to be operational for it to operate (operational dependency) or for it to be repaired (repair dependency).

When a component is operating, it fails with its operational-failure rate. However, when it is dormant, it does not fail if it is a software component, and it fails with its dormant-failure rate if it is a hardware component. Here, the dormant-failure rate is supposed to be the same as the operational-failure rate. To make sure that a component becomes dormant when it is not operating, we draw an operational-dependency graph, as shown in **Figure 2**. For example, since the central control unit (CCU) becomes dormant when the power supply (PS) fails, there is a directed edge from CCU to PS in the operational-dependency graph.

The nodes of the operational-dependency graph are the components of the system, and the edges represent the operational dependencies between these components. If a given component i requires a component j to be operational for it to operate, we draw an edge from i (slave) to j (master). The operational-dependency relation is transitive. It is not necessary to draw an additional edge, and consequently to specify an additional dependency in the SAVE input language, between components such as SCAN (scanner) and PS (power supply).

The system as a whole is considered operational when all the components are operational. This can be written as a Boolean expression. There is no need to make assertions about components which have incoming edges. Hence, the operational condition may be written as follows, using the SAVE language:

NCP and MOSSCODE and TSSCODE and CA and LIC(8)

We could also model the degraded modes by specifying the components that must be operational in each degraded mode, such as

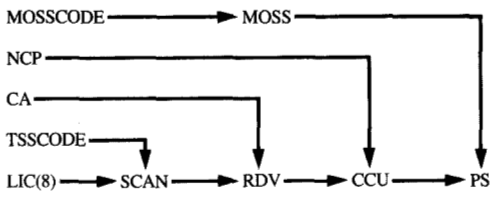

Figure 2

Operational-dependency graph.

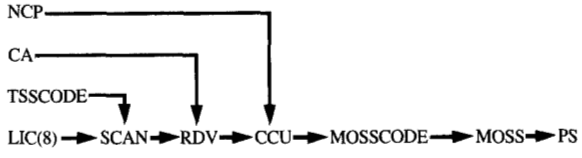

Figure 3

Repair-dependency graph.

NCP and MOSSCODE and TSSCODE and CA and LIC(7)
NCP and MOSSCODE and TSSCODE and CA and LIC(6)

. . .

NCP and MOSSCODE and TSSCODE and CA and LIC(1)

Typically, but not always, if operation of a component depends upon certain other components, then repair also depends upon at least the same components. Here, MOSS and MOSSCODE must be operational for the other components, except PS, to be repaired. For instance, operation of CCU depends upon PS, and CCU cannot be repaired without PS (and in addition MOSS and MOSSCODE) being operational. Therefore, a repair dependency graph is also necessary. **Figure 3** shows the repair-dependency graph for the present model.

The repair-dependency relation is not transitive. The repair-dependency graph may be drawn as described in Figure 3, but the related SAVE construct must contain all the components needed for the component modeled to be repaired. As an example, the repair of CA depends on RDV, CCU, MOSSCODE, MOSS, and PS.

When a component fails, it may fail in different modes dependent upon what the repair rate is and who repairs the component. When there are two failure modes, the failure rates fr1 and fr2 related to each mode are indirectly described: First, the total failure rate fr1+fr2 is written

```
MODEL: 3725
METHOD: numerical
CONSTANTS: cafr1, cafr2, carr1, carr2
  ...
COMPONENT: ca
  OPERATION DEPENDS UPON        : rdv
  REPAIR DEPENDS UPON           : rdv,ccu,mosscode,moss,ps
  FAILURE RATE                  : cafr1+cafr2
  DORMANT FAILURE RATE          : cafr1+cafr2
  FAILURE MODE PROBABILITIES    : cafr1/(cafr1+cafr2),cafr2/(cafr1+cafr2)
  REPAIR RATE                   : carr1,carr2
  REPAIRMAN CLASS USED          : fe, op
COMPONENT: ccu
  OPERATION DEPENDS UPON        : ps
  REPAIR DEPENDS UPON           : mosscode,moss,ps
  ...
COMPONENT: lic(8)
  OPERATION DEPENDS UPON        : scan
  REPAIR DEPENDS UPON           : scan,rdv,ccu,mosscode,moss,ps
  ...
COMPONENT: moss
  OPERATION DEPENDS UPON        : ps
  REPAIR DEPENDS UPON           : ps
  ...
COMPONENT: ps
  ...
COMPONENT: rdv
  OPERATION DEPENDS UPON        : ccu
  REPAIR DEPENDS UPON           : ccu,mosscode,moss,ps
  ...
COMPONENT: scan
  OPERATION DEPENDS UPON        : rdv
  REPAIR DEPENDS UPON           : rdv,ccu,mosscode,moss,ps
  ...
COMPONENT: mosscode
  OPERATION DEPENDS UPON        : moss
  REPAIR DEPENDS UPON           : moss,ps
  ...
COMPONENT: ncp
  OPERATION DEPENDS UPON        : ccu
  REPAIR DEPENDS UPON           : ccu,mosscode,moss,ps
  ...
COMPONENT: tsscode
  OPERATION DEPENDS UPON        : scan
  REPAIR DEPENDS UPON           : scan,rdv,ccu,mosscode,moss,ps
  ...
EVALUATION CRITERIA: blockdiagram
 ncp and mosscode and tsscode and ca and lic(8)
REPAIRMAN CLASS: op(1)
   REPAIR STRATEGY: ros
REPAIRMAN CLASS: fe(1)
   REPAIR STRATEGY: ros
END
```

Figure 4

Extract of the SVINPUT file using the numerical method.

in the FAILURE RATE construct, then the percentages fr1/(fr1+fr2) and fr2/(fr1+fr2) are put in the FAILURE MODE PROBABILITIES constructs related to mode 1 and mode 2, respectively. Two classes of repairmen, Field Engineer (fe) and operator (op), with one repairman per class, allow the repair of components in their respective failure modes. To repair failed components, repairmen first follow the repair dependencies, and then, if more than one component can be repaired, they randomly select a component for repair. All these characteristics are symbolically represented in a configuration file (CONFIG FILE).

The next step is the creation of the input file (SVINPUT FILE) for SAVE. A program, called CREATE, picks up the collected data (DATA FILE) from the field or from a design tool, the definition of the capsules (CONCEN FILE), and finally the configuration of the system at the component (capsule) level and the dependencies among the components (CONFIG FILE). **Figure 4** shows an extract of the input file obtained for SAVE.

Once the model is constructed and reduced, SAVE evaluates the interval availability. We select one-month intervals, since in the example one month is the maximum time during which the component-failure and -repair rates are approximately constant. Then, a graphical program draws the variation of the interval availability with time, as shown in **Figure 5**, with arbitrary input data. In particular, we see that the availability becomes almost constant at the end of the month. Thus, if the failure and repair rates were constant throughout, the permanent mode would appear after one month. In practice, we observe small fluctuations, such as availability increases, after the first month of the product's life, due to the fluctuations of the failure and repair rates. **Figure 6** summarizes the model.

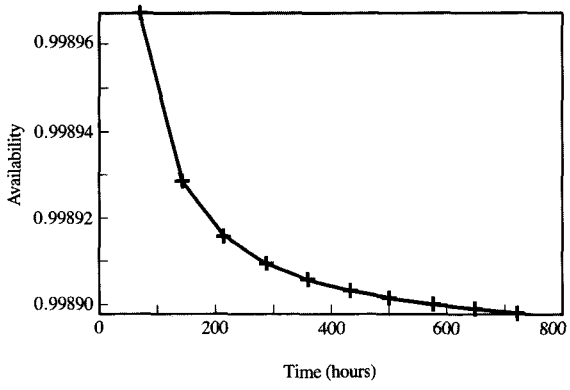

Figure 5

Variation of the interval availability with time.

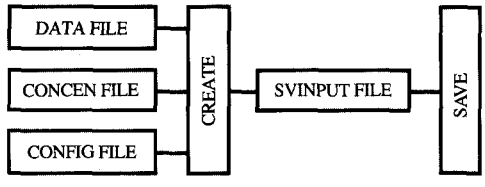

Figure 6

Summary of the model.

4. Availability improvement

SAVE allows the calculation of the sensitivity (normalized partial derivative) of the steady-state (limiting) availability with respect to input parameters. The absolute value of the sensitivity is a metric. Hence, it quantifies the impact of parameters, such as failure rates or error-detection rates, on the system availability. To cope with a large absolute value of the sensitivity with respect to the failure rate of a component in the identified-failure mode, the solution is to decrease the dependency of the system on this component; whereas to minimize the effect of the failure of a component in the unidentified-failure mode, we may maximize its error-detection capability.

In the present study, the sensitivity analysis shows that the power supply (PS) is the most critical component. The next most critical component is the CCU. Duplicating PS reduces unavailability by 89%. Duplicating CCU reduces unavailability by 3%. Duplicating all other components has an even smaller effect on unavailability. Thus, sensitivity analysis is extremely useful in working out a cost–availability trade-off. Since perfect duplication of any component is not possible, the power-supply duplication will have some common component, which should be separately modeled.

Improving software availability requires a thorough analysis of the system use and workload at the time of the occurrence of failures. The basic idea is to distinguish different failure types, and to point out those which involve the most system failures, and the areas where recovery procedures are not satisfactory. Once these areas are identified (deadlock, I/O or data management, error handling, etc.), appropriate changes can be undertaken. The interaction between hardware and software is also studied. In particular, the recovery of software errors due to hardware failures may be analyzed.

5. Guaranteed availability

It has been shown that the calculation of the availability is useful within one-month intervals. A way to guarantee the

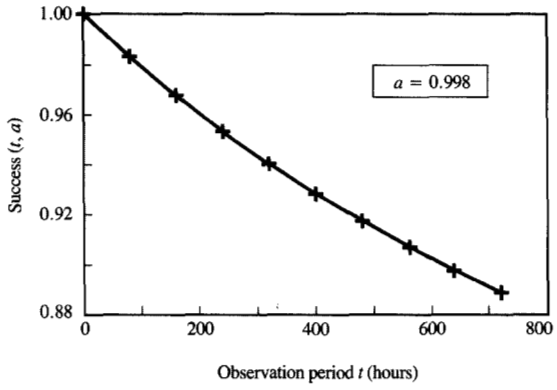

Figure 7

Variation of Success (t, a) with time.

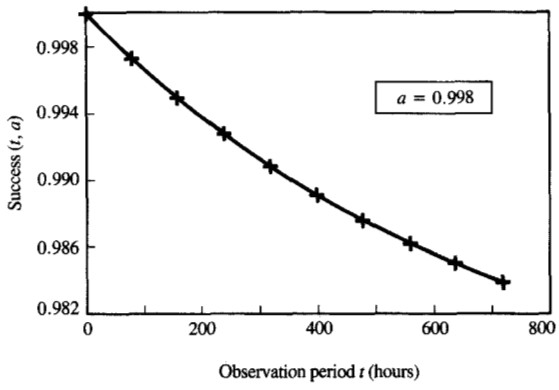

Figure 9

Variation of Success (t, a) with time, duplicate PS.

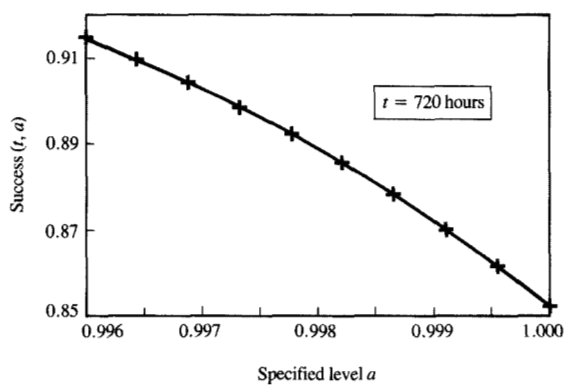

Figure 8

Variation of Success (t, a) with specified level.

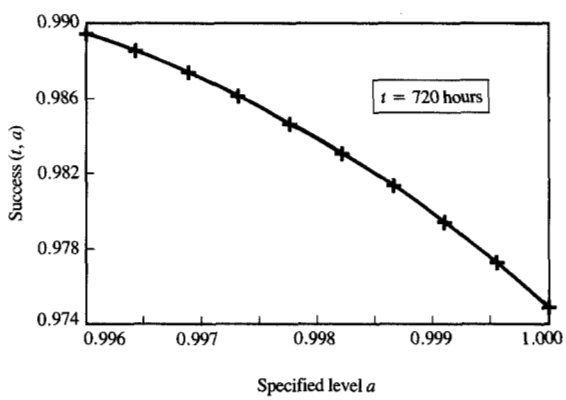

Figure 10

Variation of Success (t, a) with specified level, duplicate PS.

value of the availability over a given month is to compute the probability that a given level is met over the month; that is,

$$\text{Success}(t, a) = \Pr[A(t) \geq a].$$

There are several methods of evaluating Success(t, a) [24, 25]. SAVE evaluates Success(t, a) with a uniformization method, which converts the original Markov chain into an equivalent Markov chain where all transition rates are equal [24]. **Figures 7–10** show the variations of Success(t, a) with respect to the observation period and with respect to the specified level.

The first finding is the decrease of Success(t, a) with respect to both t and a. This decrease is perfectly reasonable: It is much greater in the case of duplicate power supply than in the case of single power supply. Here, a 0.4% increase of the guaranteed level causes a 7% and a 1.4% decrease of Success(t, a) per month with single and duplicate power supply, respectively. The decrease of Success(t, a) with time results from the increase of the probability that the system goes into failed states as the length of the observation period increases. The decrease is, of course, greater in the case of a single power supply. During one month, Success(t, a) decreases by 11% and by 2% with single and duplicate

power supply, respectively. In particular, the decreases of Success(t, a) with time throughout the observation period, given that the guaranteed level is less than the steady-state availability, accord with the fact that the availability does not tend to steady-state availability after one month. This strengthens the argument for a long tracking period. If we focus on the availability improvement provided by the duplication of the power supply, we obtain, after an observation of one month with a guaranteed level 0.998,

$$\text{Success}(720, 0.998) = 0.889,$$

and after duplicating power supply,

$$\text{Success}(720, 0.998) = 0.984.$$

Hence, after power supply is duplicated, the value of the interval availability can be guaranteed with a precision greater by 11% than the precision obtained with a single power supply.

6. Conclusion

This paper has proposed a model for evaluating the combined hardware and software interval availability of the IBM 3725 Communication Controller in operation. In particular, the interactions between hardware and software have been considered. Encapsulation of components and reduction of the number of concurrent failures allowed a reduction of the initial model with more than 2^{50} states to a tractable size, without affecting the precision of the result. Moreover, sensitivity calculations demonstrated that the power supply was by far the most critical component, and that its duplication increased the availability notably. In addition, the evaluation of the probability of meeting a specified availability level provided a means to guarantee the result of the availability calculation.

Classical combinatorial models which do not model component dependencies tend to produce extremely low unavailability numbers, reducing the belief in the availability modeling process in general. Modeling component dependencies using Markov methods tends to produce realistic unavailability numbers, as has been the experience in IBM 3725 models. We have successfully validated such numbers with actual measurements.

More details could be included in the availability model. For example, details of error-recovery and spare switch-over times could be modeled. Such models have been evaluated experimentally. Error-recovery and spare switch-over times are extremely small compared to failure and repair times. However, the probability of successful recovery or successful switch-over must be modeled, as it could affect availability significantly.

The present method did not handle a critical life stage of a product, which is the transition period between testing and operational phases, during which the statistical behavior of the system undergoes great modifications. A further study is being undertaken to investigate that area.

Acknowledgment

The author gratefully acknowledges his discussions with and the review of the preliminary version of this paper by A. Goyal of the IBM Thomas J. Watson Research Center.

References

1. P. I. Pignal, "Analysis of a Communication System with Imperfect Repair," *Microelectron. & Reliabil.* **27,** No. 1, 165–169 (January 1987).
2. P. I. Pignal, "Modelling Techniques: A Case Study," *Modelling Techniques and Performance Evaluation*, G. Pujolle, Ed., North-Holland Publishing Co., Amsterdam, 1987.
3. P. I. Pignal, "Dependability of Fault-Tolerant Systems with Multiple and Imperfect Recovery Modes," *Proceedings of the IASTED (International Association of Science and Technology for Development) International Conference on Reliability and Quality Control*, Paris, France, June 1987, pp. 13–16.
4. J. D. Musa and K. Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," *Proceedings of the 7th International Conference on Software Engineering*, Orlando, FL, March 1983, pp. 230–237.
5. M. L. Shooman, "Software Reliability Measurement and Models," *Proceedings of the Annual Reliability and Maintainability Symposium*, Washington, DC, January 1975, pp. 485–491.
6. B. Littlewood, "How to Measure Software Reliability and How Not to," *IEEE Trans. Reliabil.* **R-28,** 103–110 (January 1979).
7. H. Hecht and M. Hecht, "Software Reliability in the System Context," *IEEE Trans. Software Eng.* **SE-12,** 51–58 (January 1986).
8. R. K. Iyer and D. J. Rossetti, "Effect of System Workload on Operating System Reliability: A Study on IBM 3081," *IEEE Trans. Software Eng.* **SE-11,** No. 12, 1438–1448 (December 1985).
9. A. Costes, C. Landrault, and J.-C. Laprie, "Reliability and Availability Models for Maintained Systems Featuring Hardware Failures and Design Faults," *IEEE Trans. Computers* **C-27,** 548–560 (June 1978).
10. J.-C. Laprie, "Dependability Evaluation of Software Systems in Operation," *IEEE Trans. Software Eng.* **SE-10,** No. 6, 701–714 (November 1984).
11. A. Goyal and S. Lavenberg, "Modeling and Analysis of Computer System Availability," *IBM J. Res. Develop.* **31,** No. 6, 651–664 (November 1987).
12. S. Gerhart and L. Yelowitz, "Observations of Fallibility in Applications of Modern Programming Methodologies," *IEEE Trans. Software Eng.* **SE-2,** 195–207 (May 1976).
13. W. C. Carter, "A Time for Reflection," *Proc. FTCS-12* (Symposium on Fault-Tolerant Computing), Los Angeles, June 1982, p. 41.
14. J.-C. Laprie and A. Costes, "Dependability: A Unifying Concept for Reliable Computing," *Proc. FTCS-12* (Symposium on Fault-Tolerant Computing), Los Angeles, June 1982, pp. 18–21.
15. K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall Publishing Co., Englewood Cliffs, NJ, 1982.
16. D. P. Siewiorek and R. S. Schwarz, *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, MA, 1982.
17. R. W. Butler, "The SURE Reliability Analysis Program," *NASA Technical Memorandum 87593*, Langley Research Center, Hampton, VA, February 1986.
18. J. B. Dugan, K. S. Trivedi, M. K. Smotherman, and R. M. Geist, "The Hybrid Automated Reliability Predictor," *AIAA J. Guidance, Control & Dynam.* **9,** No. 3, 319–331 (May–June 1986).
19. R. A. Sahner and K. S. Trivedi, "A Hierarchical, Combinatorial-Markov Method of Solving Complex Reliability Models," *Proceedings of the Fall Joint Computer Conference*, Dallas, TX, November 1986, pp. 817–825.

20. R. B. Conn, P. M. Merryman, and K. L. Whitelaw, "CAST—A Complementary Analytic-Simulative Technique for Modeling Complex Fault-Tolerant Computing Systems," *Proceedings of the Computers in Aerospace Conference* (American Institute of Aeronautics and Astronautics), Los Angeles, November 1977, pp. 6.1–6.27.
21. J. J. Stiffler, "Computer Aided Reliability Estimation," *Proceedings of the Computers in Aerospace Conference* (American Institute of Aeronautics and Astronautics), Los Angeles, November 1977, pp. 427–434.
22. P. J. Courtois and P. Semal, "Bounds for Positive Eigenvectors of Nonnegative Matrices for Their Approximations by Decomposition," *J. ACM* **31,** No. 4, 8.4–8.25 (October 1984).
23. A. Bobbio and K. S. Trivedi, "An Aggregation Technique for the Transient Analysis of Stiff Markov Chains," *IEEE Trans. Computers* **C-35,** No. 9, 803–814 (September 1986).
24. A. Goyal and A. N. Tantawi, "Numerical Evaluation of Guaranteed Availability," *Proc. FTCS-15* (Symposium on Fault-Tolerant Computing), Ann Arbor, MI, June 1985, pp. 324–329.
25. V. G. Kulkarni, V. F. Nicola, R. M. Smith, and K. S. Trivedi, "Numerical Evaluation of Performability and Job Completion Time in Repairable Fault-Tolerant Systems," *Proc. FTCS-16* (Symposium on Fault-Tolerant Computing), Vienna, Austria, July 1986, pp. 252–257.

Received July 7, 1986; accepted for publication August 24, 1987

Pierre I. Pignal *IBM Communication Products Division, La Gaude Laboratory, 06610 La Gaude, France.* Mr. Pignal is a system designer in the Communication Controller Products Architecture Department at the La Gaude Laboratory. In 1985, he received the Engineer degree from the Ecole Nationale Supérieure des Télécommunications, Paris, France. Mr. Pignal joined IBM in 1985 and worked in the areas of performance and reliability modeling and evaluation of communication controllers. Since 1987, he has been a system designer. He is currently a lecturer in the IBM International Education Center, La Hulpe, Belgium, where he teaches the design of highly available systems. He is also a lecturer at the Department of Computer Science in the Ecole Centrale des Arts et Manufactures, the Ecole Nationale Supérieure des Mines de Paris, and the Ecole Nationale Supérieure des Télécommunications, all in Paris, where he teaches software reliability. Mr. Pignal has published more than ten papers in the area of reliability. He is a member of the Association Française pour la Cybernétique Economique et Technique, the Association for Computing Machinery, and the Institute of Electrical and Electronics Engineers. As a member of the IEEE Computer Dictionary Project Committee, he has been involved in the Networking Hardware, Computer Networking, Modeling and Simulation, and Data Management working groups. Mr. Pignal received the IBM France Technical Vitality Award in 1987.