



HAL
open science

Dependability of Fault-Tolerant Systems - Explicit Modeling of the Interactions Between Hardware and Software Components

Karama Kanoun, Marie Borrel

► **To cite this version:**

Karama Kanoun, Marie Borrel. Dependability of Fault-Tolerant Systems - Explicit Modeling of the Interactions Between Hardware and Software Components. 2nd Annual IEEE International Computer Performance and Dependability Symposium (IPDS'96), 4-6 septembre 1996, pp.252-261, Sep 1996, Urbana-Champaign, United States. hal-01976182

HAL Id: hal-01976182

<https://laas.hal.science/hal-01976182>

Submitted on 9 Jan 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Dependability of Fault-Tolerant Systems — Explicit Modeling of the Interactions Between Hardware and Software Components

Karama Kanoun and Marie Borrel

LAAS-CNRS

7, Avenue du Colonel Roche
31077 Toulouse Cedex - France

Abstract

This paper addresses the dependability modeling of hardware and software fault-tolerant systems taking into account explicitly the interactions between the various components. It presents a framework for modeling these interactions based on Generalized Stochastic Petri Nets (GSPNs). The modeling approach is modular: the behavior of each component and each interaction is represented by its own GSPN, while the system model is obtained by composition of these GSPNs. The composition rules are defined and formalized through the identification of the interfaces between the component and the dependency nets. In addition to generality, the formalism brings flexibility and reusability. This approach is applied to a simple, but still representative, example.

Introduction

In the context of computer system dependability, the need for addressing simultaneously both hardware and software dependability aspects has now been recognized. However, even though a number of publications have been devoted to the dependability of combined hardware and software systems (see e.g. [5, 6, 13, 14]), work on aspects dealt with at the same time is not prevalent. In fact, it is noteworthy that, when they are considered either for real-life systems, the interactions between the components are not usually modeled explicitly (see e.g. [16, 20]).

This paper addresses the dependability modeling of hardware and software fault-tolerant systems taking into account the interactions between the various components. These interactions result for example from components communications for functional purposes (i.e., *functional interactions*), or from the structure of the system, mainly the distribution of software components onto the hardware components (i.e., *structural interactions*), or from fault tolerance and maintenance strategies

induce dependencies between at least two components that are usually stochastic in nature. As a result, system dependability cannot simply be obtained by combining the dependability of its components. An overall model accounting for these dependencies is thus needed. Our aim is to model explicitly these dependencies so as to quantify their influence on system dependability. This is of prime importance during the design of a new system while upgrading an already existing one. The designer can make different assumptions about the interactions between the components and compare the dependability of the resulting alternative solutions through sensitivity studies. As the nature of interactions is strongly linked to the modeling level considered and the assumptions made at the considered level, it is not possible to model all interactions that could take place for any fault-tolerant system. Rather, we define a framework for modeling these interactions in a systematic way and, more generally, we define a framework to build up the dependability model of a fault-tolerant system explicitly taking into account these interactions. To do this, we follow a modular approach based on Generalized Stochastic Petri Nets (GSPNs) due to their ability to handle modular and hierarchy. Note that modular approaches using GSPNs or their offsprings are widely used (see e.g., [18]). Our contribution lies in modeling the interactions between hardware and software components and giving a formal description of these dependencies.

The paper is organized in five sections: Section 1 presents the framework for modeling interactions between hardware and software components. Section 2 gives a formal description of the various types of dependency nets while Section 3 illustrates the approach on a duplex system with several interactions. Section 4 concludes.

2. Modeling Framework

The modeling approach consists in identifying, based on the analysis of the system's behavior, dependencies

al or structural interactions or by interactions due to
 om reconfiguration and maintenance. Some examples
 dependencies due to these interactions are given in the
 wing. Error propagation between two software com-
 nents is an example of stochastic dependency resulting
 functional interactions (exchange of data or transfer
 intermediate results from one component to another).
 halting of the software activities following a perma-
 failure of the hosting hardware is an example of
 stastic dependency induced by a structural interaction.
 ing of a single repairman by the two hardware com-
 rs leads to a maintenance dependency whereas
 ching from an active component (hardware or
 vare) to a spare component following a permanent
 re of the active component leads to a reconfiguration
 ndency. In this paper, we consider interactions that
 driven by events occurring in a component whose
 rrence may impact the behavior of other components.
 A high level model of the system is first derived based
 re previous analysis. It is made of blocks and arrows:
 ock stands for the component model (*component net*)
 dependency model (*dependency net*), and an arrow
 vs the direction of the dependency. The system model
 us obtained by composition of the component and
 ndency models. In a second step, each block is
 iced with its detailed GSPN. To allow for a
 amatic build up of dependency nets, rules that will
 to be followed during model construction are
 ed. These rules manage the interfaces between the
 ndency and component nets and are prerequisite for
 ularity, hierarchical modeling and re-usability (re-
 ility is a valuable concept when it comes to doing
 itivity studies about certain assumptions regarding a
 om's behavior or when several alternative solutions
 being considered). Also, these rules allow an easy
 lation of the global model. In the rest of the section,
 give the characteristics of the component and
 ndency nets and present the various types of
 ndency nets together with the rules that have to be
 wed to build up the GSPNs.

Components nets: A component net represents the
 vior of a component as resulting from the activation
 ults in this component and the subsequent error
 essing, restart or repair actions. The assumptions
 e and the degree of detail considered are usually
 ed by the interactions with other components one
 is to exploit (such as the consequences of non
 cted errors or activation of temporary faults). A
 ponent net is designed to be a standalone net with its
 al marking, it is live and bounded. It can be
 ected with dependency nets only following well de-

efined rules as explained hereafter; connections must
 alter the initial structure of the component nets.

Dependency nets: A dependency net is linked to at le
 two adjacent nets: an *initializing* and a *target* net th
 could be component or dependency nets. To forma
 describe dependency nets and to promote re-usability,
 define as much common characteristics as possible. A
 result, whatever is the kind of interaction modeled, all
 dependency nets are initialized and interfaced with
 adjacent nets following the same rules; they only differ
 their *effects* on the target net. The common characterist
 and the different effects on the target nets are introduc
 in figure 1 where a hypothetical dependency net with
 types of effects is given (the notations are introduced w
 the formalism). They are summarized hereafter.

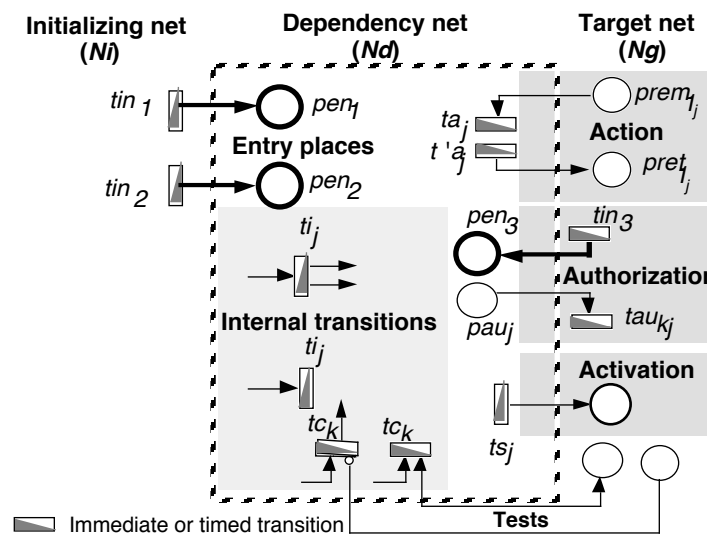


Figure 1: Characteristics of dependency nets

Dependency net initialization

- A dependency net is initialized through the marking of one or more *entry places* by the initializing net following firing of *initializing transitions* in the initializing net(s) (as interactions are event driven).
- The initial marking of the entry places is zero.
- When the initializing net is a component net, the consequences of initializing transition(s) firing on the component behavior are modeled within the component net (marking of one or several internal places) and an additional token is generated and deposited in the entry place of the dependency net to activate the interaction.
- If the initializing net is a dependency net, the token deposited in the entry place could be either the token generated when entering the initializing dependency net (corresponding to a series of successful interactions) or an additional one newly generated (corresponding to the initialization in parallel of two or more interactions).

Internal transitions: The dependency net has internal transitions (timed or instantaneous) whose firing may be dependent from the marking of the adjacent nets (dependent transitions) or conditioned by the marking of places in the adjacent nets. A condition is modeled by an inhibitor arc or an arc from and towards the tested place: the marking of this place is not changed. The effects on the adjacent nets (excluding initializing and effects on the target nets) are thus only conditioned by tests on the marking of specific places. An internal transition could be absorbing (i.e., the tokens are absorbed).

Effects on target nets: These effects are strongly related to the type of interaction modeled. Thus, three such effects have been identified:

• if the interaction consists in changing the state of another component (the target net is necessarily a component), the effect at the GSPN level involves the removal of a token from a stable place (i.e., a place allowed by timed transitions, this condition stems from the fact that only stable places correspond to states of the components) in the target net and return of the token to the same place or to another stable place of the target net (immediately or after firing of internal transitions in the dependency net). This is referred to as an *action net*.

• if the interaction consists in performing or synchronizing reconfiguration or maintenance actions, the effects depend on the nature of the initializing net:

if the initializing net is a component net, the interaction consists in coordinating the component restart (or repair) action with the restart (or repair) action of the components to which it is linked: it requests permission before undertaking internal actions, these actions are enabled by the dependency net (immediately or after firing of internal transitions). The target net is necessarily the same as the initializing net. At the GSPN level, the effect consists of enabling a transition in the component net through the marking of a place in the dependency net. Since the component net is a standalone net, this means that, in the component net, this transition has also to be enabled by the marking of at least an internal place. This is an *authorization net*,

if the initializing net is a dependency net, the interaction consists in activating another interaction with other linked components; at the GSPN level, this consists of initializing another dependency net by depositing a token in its entry place following the firing of an initializing transition in the initializing dependency net. As previously stated, depending on

additional one, both dependency nets run in parallel or in series. This is an *activation net*.

The previous rules are intended to manage the state links between dependency and adjacent nets. Further rules have to be considered to control the *dynamical behavior* of the nets (i.e., the tokens generation and the flow). They are given together with the formalism in the next section. Note that model construction is an iterative process with information flow in both directions from dependency nets to/from components nets: in the component models, care should be taken to include potential dependencies.

Interaction origin and dependency net type: The interactions have been attributed to three possible origins: functional, structural and those due to system configuration and maintenance. Functional and structural interactions are usually accompanied by a state change, the associated nets are thus action nets. Dependencies due to reconfiguration and maintenance may induce a state change and they could be any kind of dependency net.

3. Nets formalization and validation

The aim of this section is to give a formal description of the various rules introduced in the previous section and to address model validation. We first give the main notations, the other notations being defined in table 1.

General Notations

$pen_k \in P_d$	Entry place (EP) of Nd.
$tin_j \in T_i$	Transition of Ni that initializes Nd by marking an EP
M_{ij}	Initializing marking of a dependency net
$ti_j \in T_d$	Internal independent transition in Nd
$tc_k \in T_d$	Internal conditioned transition in Nd

Interfaces for an action net

$ta_j, t' a_j \in T_d$	Removing and returning action transitions in Nd
$prem_k \in T_g$	Input place in Ng of the removing action trans. ta_j
$pret_k \in T_g$	Output place in Ng of the returning action trans. $t' a_j$

Interfaces for an authorization net

$pau_j \in P_d$	Authorization place of Nd
-----------------	---------------------------

Interfaces for an activation net

$ts_k \in T_d$	Activation transition of Nd
----------------	-----------------------------

Table 1: Notations

Let N_i , N_d and N_g denote respectively an initializing dependency, and a target net, and let N_x be any of the nets ($x = i, d, g$). $N_x = (P_x, T_x, I_x, O_x, pr_x, pa_x)$ where:

- P_x is the set of places of N_x ,
- $T_x = T_{im_x} \cup T_{imm_x}$ is the set of transitions of N_x : T_{im_x} is the set of timed transitions and T_{imm_x} is the set of immediate transitions.
- $I_x: P_x \times T_x \rightarrow \mathbf{N} \cup \{-1\}$ is the input function,

$x: T_x \times P_x \rightarrow \mathbf{N}$ is the output function (\mathbf{N} is the set of natural integers).

r_x the set of timed transitions rates, pa_x the set of ring probabilities of immediate transitions.

the set of places and transitions are such that:
 $P_d = \emptyset$, $P_i \cap P_g = \emptyset$, $P_d \cap P_g = \emptyset$, $T_i \cap T_d = \emptyset$,
 $T_g = \emptyset$ and $T_d \cap T_g = \emptyset$

the interfaces of a dependency net N_d with an initializing net N_i and a target net N_g are the input and output functions, I_{id} , I_{dg} , O_{id} , O_{dg} , that connect places and transitions of N_d to places and transitions of N_i or N_g . These functions are defined as follows:

$$\begin{aligned} id: P_i \times T_d &\rightarrow \mathbf{N} \cup \{-1\} & I_{dg}: P_d \times T_g &\rightarrow \mathbf{N} \cup \{-1\} \\ i_d: T_i \times P_d &\rightarrow \mathbf{N} & O_{dg}: T_d \times P_g &\rightarrow \mathbf{N} \end{aligned}$$

When it is not necessary to distinguish between initializing and target nets, indices i, d, g are omitted.

Initialization: Initializing transitions, entry places and initializing marking of a dependency net N_d are defined as follows: let $tin_j \in T_i$ and $p_k \in P_d$ such that $I_{id}(p_k, tin_j) > 0$, $I_{di}(p_k, tin_j) = 0$, if $\forall p_l \in P_d, p_l \neq p_k$, we have $I_{di}(p_l, tin_j) = 0$ then tin_j is an initializing transition of N_d and p_k is an entry place of N_d , denoted pen_k . An entry place can be initialized by several transitions. In order for an initializing transition tin_j of a net N_i to be fired, one must have:
 $\exists p_m \in P_i$ $I_{id}(p_m, tin_j) \geq 0 \Rightarrow Min(p_m) \geq I_{id}(p_m, tin_j)$ (Min : N_d initializing marking) and $I_{id}(p_m, tin_j) = -1 \Rightarrow Min(p_m) = 0^1$.

Internal transitions: Internal transitions of a dependency net can be independent or conditioned by the marking of places in adjacent nets. They are defined as follows:

$ti_j \in T_d$ is an independent transition if $\forall p_k$ such that $I_{id}(p_k, ti_j) > 0$ or $O_{id}(ti_j, p_k) > 0$ then $p_k \in P_d$, $I = I_d$, $O = O_d$.

$tc_k \in T_d$ is a conditioned transition if the two following conditions are verified:

- 1) $\exists p_j \in (P_g \cup P_i)$ such that: $I(p_j, tc_k) = O(tc_k, p_j) > 0$ or $I(p_j, tc_k) = -1$
- 2) $\forall p_n \in (P_g \cup P_i)$ such that $I(p_n, tc_k) > 0$ or $O(tc_k, p_n) > 0$ then $I(p_n, tc_k) = O(tc_k, p_n)$.

An internal transition can be an absorption transition. An independent or a conditioned transition ti_j or tc_j is an absorption transition if: $\forall p_n \in P$, $O(ti_j, p_n) = 0$ or $O(tc_j, p_n) = 0$ with $P = (P_d \cup P_i \cup P_g)$.

Output nets: An action net ends with a transition which removes a token to be removed from a stable place of the target net and to be returned to the same or another stable place of the same net. Removal and return can be done through two distinct transitions (with internal transitions

¹ $I(p, t) \geq 0 \Rightarrow Mn_x(p) \geq I(p, t)$ means that there must be enough tokens in all the input places of t to enable it.
 $I(p, t) = -1 \Rightarrow Mn_x(p) = 0$ means that if there is an inhibitor arc

and places between them) or through the same transition. The number of tokens in the target net remains unchanged. Transitions ta_j and ta_j' are action transitions if the four following conditions are met:

- 1) $\forall p_h \in P_i, I(p_h, ta_j) = 0$ and $\forall p_l \in P_i, O(ta_j, p_l) = 0$
- 2) $\exists p_k \in P_g$ such that: $O_{dg}(ta_j, p_k) = 0$ and $I_{dg}(p_k, ta_j) > 0$ (ta_j : removing transition)
 $\exists p'_k \in P_g$ such that: $O_{dg}(ta_j', p'_k) > 0$ and $I_{dg}(p'_k, ta_j') = 0$ (ta_j' : returning transition)
- 3) $\sum_{i=1}^{|P_g|} O(ta_j, p'_i) = \sum_{i=1}^{|P_g|} I(p_i, ta_j)$
- 4) p_k or p'_k is a stable place: $\exists t \in Tim_g$ such that $I_d(p_k, t) > 0$ and $\forall t \in Tim_g$ such that $I_d(p_k, t) > 0$ then $I_d(p_k, t) = O_d(t, p_k)$, p'_k must verify an equivalent relation.

p_k is then the input place of the removing action ta_j being denoted pre_{mk_j} and p'_k the output place of the returning transition ta_j' , is denoted pre_{tk_j} .

Authorization nets: An authorization net ends with one or several places enabling firing of transitions in the target net(s), authorization places. In this case, the target net is necessarily the initializing net.

$pau_j \in P_d$ is an authorization place of N_d if:
 $\forall t \in T_d$ such that $I(pau_j, t) = 0$ then $\exists t_{jk} \in (T_i \cup T_g)$ such that $I(pau_j, t_{jk}) > 0$ and $O(t_{jk}, pau_j) = 0$; t_{jk} is then an authorized transition in N_c , it is denoted tau_{kj} .

Activation nets: An activation net allows linking dependency nets (i.e. synchronize the relations between interactions). It ends with a transition, synchronization transition, that sends one or several tokens in one or several other dependency nets (but does not remove tokens from these nets). $ts_j \in T_d$ is an activation transition if $\forall p_m \in P_g$ $I_{gd}(p_m, ts_j) = 0$ and if $\exists p_k \in P_g$ such that $O_{dg}(ts_j, p_k) > 0$.

Dynamic behavior: The generation, moving and absorption of the tokens has to be controlled when building up a dependency net so as to ensure that the resulting global net is bounded and live. Each token generated upon dependency net initialization by the marking of an entry place must thus be removed either through the dependency net itself or through the effect on the target net. It is then necessary that as long as a dependency net place is marked, whatever the global marking, there is a transition that can be fired and that removes a token from this place. This condition must be formalized for the internal places of all types of dependency nets.

Let P_y be such that $P_y = P_d$ for action and activation nets, $P_v = \{P_d - P_{aut}\}$ for authorization nets. The inter-

very place has at least one transition that removes tokens: $\forall p \in P_y \exists t \in T_d \mid I(p,t) > 0$ and $O(t,p) = 0$

there exists an arc with multiplicity x from a place to transition, there exist $x-1$ other arcs with multiplicity to $x-1$, from the same place to $x-1$ other transitions with the same input and output as the preceding transition:

$t \in T_d$, and $p \in P_y$ with $I(p,t) > 0$ and $O(t,p) = 0$ if $p' \in P_d$ such that $I(p',t) = x$, $x \in \mathbf{N}$ then $\exists x$ transitions $t_j \in T_d$ such that $I(p',t_j) = j$, $j = 1, \dots, x$, $I(p,t_j) > 0$ and $O(t_j,p) = 0$.

If these transitions are independent internal transitions.

there exists a test arc with multiplicity x between a place and a transition, there exist $x-1$ other arcs with multiplicity 1 to $x-1$, from the same place to $x-1$ other transitions with the same input and output as the preceding transition.

$p \in P_y$ and $\forall p' \in P_y$ with $I(p,t) > 0$ and $O(t,p) = 0$, if $p' \in P$ such that $I(p',t) = O(t,p') = x$, then $\exists x$ transitions t_j such that $I(p',t_j) = O(t_j,p') = j$, $j = 1, \dots, x$ $I(p,t_j) > 0$ and $O(t_j,p) = 0$.

$p' \in P_d$, t is an independent transition, if $p' \notin P_d$, t is a conditioned transition.

there exists an inhibitor arc from a place to a transition then there must exist an arc from the same place to another transition.

$t \in T_d$ and $p \in P_y$ with $I(p,t) > 0$ and $O(t,p) = 0$, if $\exists p' \in P$ such that $I(p',t) = -1$, then $\exists t'$ such that $I(p',t') = 0$, $I(p',t) > 0$ and $O(t',p) = 0$.

$p' \in P_d$, t is an independent transition, if $p' \notin P_d$, t is a conditioned transition.

The sum of firing probabilities of immediate transitions in conflict is always equal to 1:

$p \in P_d$, if there are $\{t_1 \dots t_i \dots t_u\} \subset Timm_d$ such that $I(p,t_i) > 0$, and $\forall p'$ such that $I(p',t_{i=x}) = \text{constant}$ and $I(p',t_{i \neq x}) = 0$, then $\sum_{i=1}^n pa(t_i) = 1$.

$p' \in P_d$, t is an independent transition, if $p' \notin P_d$, t is a conditioned transition.

In some situations, depending on the marking of the target net, the token must be removed by an absorbing transition if it cannot be removed by the target or initializing net: $\forall p \in P$ if $\exists t \in T_d$ such that $I(p,t) \geq 0$ or $I(p,t) = -1$ then $O(t,p) = 0$.

Model validation: Several verifications are needed to have confidence in the model; they are usually grouped into two categories: syntactic and semantic validation. Syntactic validation consists in checking that the model represents the dependability of a system; it mainly

consists of checking that the model represents the dependability of the system under validation; it requires comparing the system and the model behaviors with respect to variations of the underlying assumptions. Usually it is performed through sensitivity studies. Due to the scope of the paper, we concentrate on the structural validation. The rules for interfacing dependency among component nets and for managing the dynamic behavior allow us to obtain, by construction models that are structurally valid (i.e., live and bounded). Structural validation is progressively done, starting from simple component nets and gradually adding dependency relations. Identification of possible problems is thus easy. All these verifications can be achieved automatically through the computation of place- and transition-invariants and checking necessary or sufficient conditions of liveness and boundedness with a tool such as SURF-2 developed at LAAS-CNRS.

4. Application to the duplex system

Let us consider a duplex system composed of two hardware computers (H1 and H2) and two identical software replicas: each replica is implemented on a computer. We assume semi-active replication [17]: the leader replica (L) processes all input messages and provides output messages while the follower replica (F) does not produce output messages. The internal state of F is updated by means of notifications from L completed direct processing. Temporary faults in the software are tolerated by exception handling mechanisms associated with each replica, whereas the activation of permanent faults leads to restart the replica. To reduce system unavailability, after detection of an error due to a permanent fault in L, the software replicas switch their roles: processing is performed on the new leader before restarting the new follower. If L and F fail, L is restarted first. Also, in case of failure of the hardware hosting (identified as H1), the replicas switch their roles; the computer hosting the new follower is then repaired. With respect to hardware repair policy when the two computers are in failure, we consider two assumptions: R1: the two computers share a single repairman and priority of repair is given to H1 and R2: two repairmen are available.

4.1. High level modeling

Interactions are directly related to the assumptions made about the components' behavior. Owing to the importance of the impact of temporary faults on the behavior of hardware and software components [7, 8, 19], both permanent and temporary faults are considered in this example.

It is assumed that the activation of a fault may lead to the following dependencies:

Following activation of a hardware fault:

an error due to the activation of a temporary fault in a hardware computer may *propagate* to the hosted software replica,

an error due to the activation of a permanent fault in a hardware computer leads to *stopping* the hosted software replica that is restarted after the end of hardware repair.

Following activation of a software fault: owing to the notifications sent from the leader to the follower, an error in the leader due to a permanent fault — usually referred to as solid fault — may *propagate* to the follower (it is assumed that errors due to temporary faults — usually referred to as soft faults — are confined and do not propagate).

Dependencies induced by fault tolerance and maintenance strategies are as follows:

Between software replicas: dependency due to fault tolerance of permanent software faults, i.e., *reconfiguration* from F to L.

Between hardware computers: dependency due to *reconfiguration and repair*.

Between all components: coordination of fault tolerance and maintenance actions to form a global *recovery strategy* when several components are in failure.

These dependencies are summarized in table 2 together with the names of the associated nets which are used to build up the high level model of the duplex system. The latter is given in figure 2 where N_{Hard} and N_{Soft} represent a computer and a software replica model respectively.

N_{Prop}	models propagation of a hardware error to the hosted software replica
N_{Stop}	models software stop after activation of a permanent fault in the hosting hardware
N'_{Prop}	models propagation of a software error
N_{Rep}	models hardware reconfiguration and repair
N_{Rec}	models software reconfiguration
N_{Strat}	models the global recovery strategy

Table 2: Dependency nets

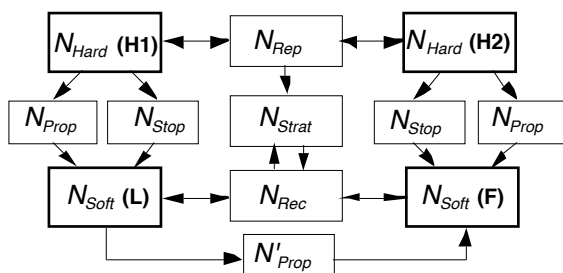


Figure 2: High level model of the duplex system

The corresponding GSPNs are built up following the rules and formal description presented in Sections 2 and 3; they are successively given in the remainder of this section.

4.2. Hardware and software component nets

Figure 3 gives the component nets. The hardware model is based on the following assumptions:

- Faults are activated with rate λ_h .
- With probability p_h the fault is permanent, (probability of a temporary fault $(1-p_h)$).
- The effects of an error due to a temporary fault are eliminated within a short time $1/\delta_h$.
- An error due to a permanent fault is either detected with probability d_h , or non detected $(1-d_h)$; error processing rate: τ_h .
- The effects of a permanent, non detected error may be perceived later (perception rate ζ_h).
- The repair rate including software restart (following detection or perception of an error) is μ .

Equivalent assumptions are made regarding the behavior of the software replicas:

- Faults are activated with rate λ_s .
- An error is either detected with probability d_s , or non detected $(1-d_s)$; detection rate τ_s .
- The detected error is processed by means of exceptional handling mechanisms during a short time $1/\pi_s$. At the end of error processing, 1) if the fault is temporary (probability $(1-p_s)$) its effects are eliminated and the software resumes its normal mode of operation, 2) the fault is permanent (probability p_s); the software has to be restarted (rate: ν) to eliminate its effects. $(1-\nu)$ measures the efficiency of fault containment procedures [8, 11].
- The effects of a non detected error may be eliminated

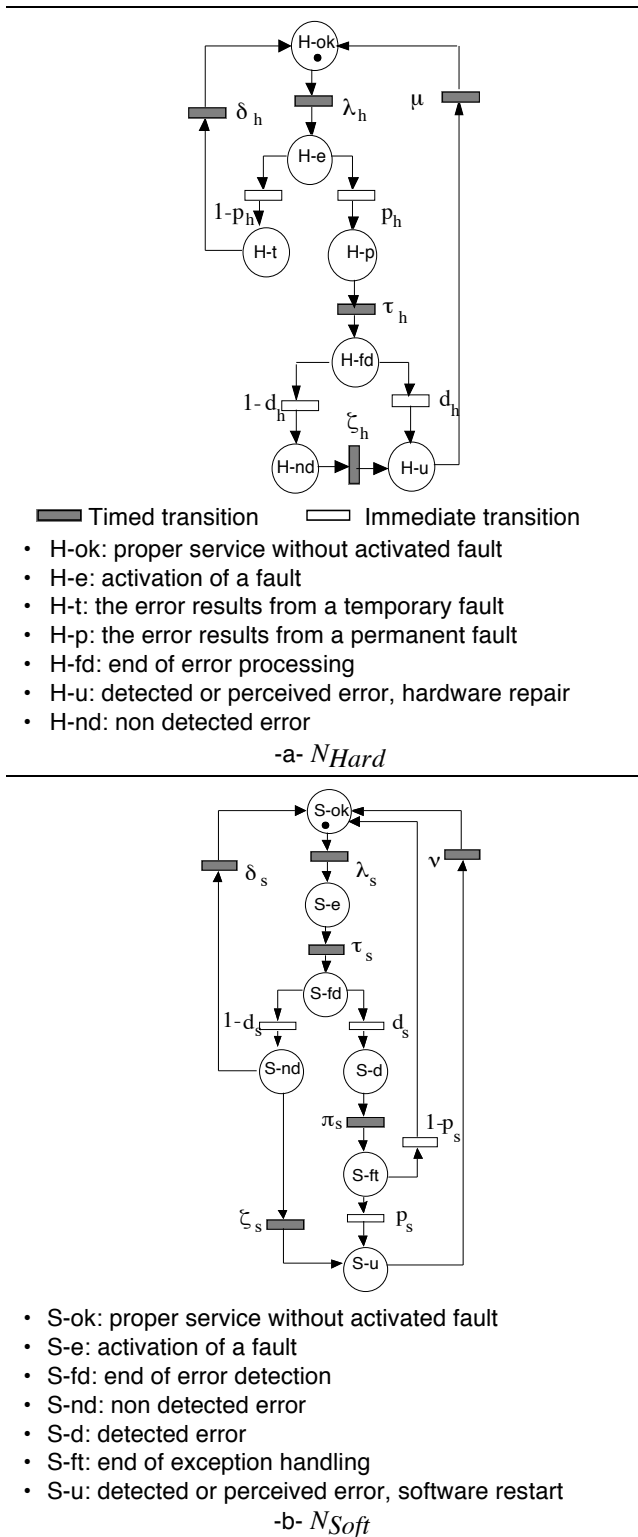


Figure 3: Hardware computer and software replica nets

elimination rate δ_s), or perceived (perception rate ζ_s) which case the software replica has to be restarted. The difference between these nets lies in that for hardware, temporary and permanent faults are reinitiated by their respective consequences following activation, whereas for software, they can only be extinguished after specific processing [12].

Error propagation nets

From hardware to software: It is assumed that only

propagate from a hardware computer to the host software replica. The error propagation net, shown in figure 4, is initialized by the marking of place P_i following the firing of transition $1-d_h$ (undetected error) or of transition $1-p_h$ (an error due to a temporary fault) in the hardware net (initializing net). With probability $1-p$ the error is not propagated and with probability p it is propagated. N_{Prop} is an action net, whose effects on the software net are as follows:

- If the token is in S-ok, it is returned to S-e, the induced error is then processed in the same way as when a fault is activated without propagation (through λ_s in figure 3-b).
- If the token is in S-e, since a fault is already activated in the software, the probability of error detection may be reduced ($d'_s \leq d_s$), if the errors are detected, the token is returned to S-d; if they are non detected (with probability $1-d'_s$) the token is returned to S-nd.
- If the token is in S-nd (an internal error is non detected in the replica) the propagated error and the internal error are detected with probability d''_s ($d''_s \leq d'_s$, owing to the perturbation due to the first error) the token is returned to S-d; the errors remain undetected with probability $1-d''_s$.
- If the token is in S-d the propagated error compromises error processing and prevents the recovery of an error due to a temporary fault. The internal propagated errors are recovered with probability $1-p_p$ ($1-p_p < 1-p_s$).
- If the token is in S-u, the software replica is already under restart, the token of N_{Prop} is absorbed through $tp-u$ and the token of N_{Soft} is kept in S-u.

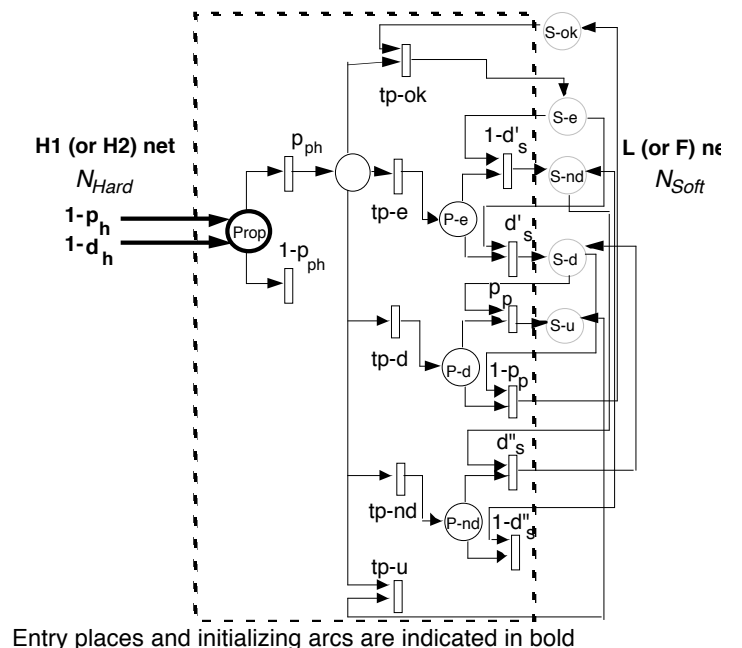


Figure 4: Error propagation net, N_{Prop}

in L to F: The dependency net, the target net and the transitions on the target net are exactly the same but the initializing net is that of the software leader. It is assumed only undetected errors in L and detected errors of L to permanent faults, can propagate. The error propagation net is then initialized following the firing of ξ_h or ζ_h . The probability of error propagation is p_{ps} .

At a higher modeling level, error propagation from L to F is regarded as common mode failures.

Software stop and restart net

Following a detected error or the perception of an undetected error in an hardware computer, the hosted software replica is stopped and is restarted after repair of hardware. We assume that the repair includes the hardware restart. The software stop and restart net (Figure 5) is initialized by the marking of STP following the firing of transition ξ_h (perception of a non detected error due to a permanent fault) or d_h (detection of an error due to a permanent fault). Transitions $t1$ to $t5$ move the token from places S-ok, S-e, S-d, S-nd or S-u respectively. After repair of the hardware (including software restart), RST is marked and the token is returned to S-ok.

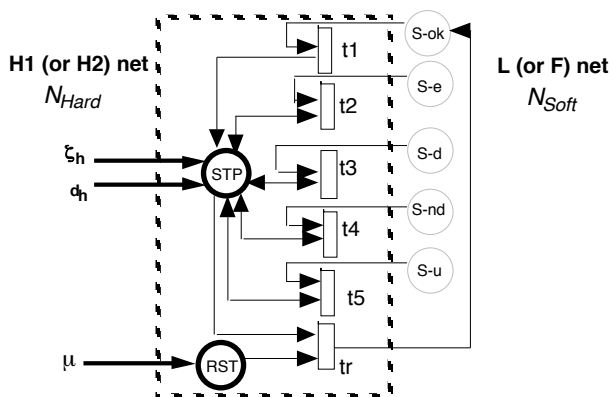


Figure 5: Software stop and restart, N_{Stop}

Hardware reconfiguration and repair net

As previously stated, we consider two different options: A1 assumes a single repairman, while A2 assumes the presence of two repairmen. The corresponding nets are given in figure 6. Each net is composed of two parts corresponding respectively to hardware reconfiguration (the shaded parts on the figures) and repair. They are grouped together because the hardware reconfiguration is automatically followed by a repair. Since the reconfiguration strategy is the same, the corresponding nets are the same. The two nets are commented together and, when they are different, the figure number is specified.

N_{Rep} is initialized by the marking of H (respectively H2F) following the firing of d_h , detection of an error due to a permanent fault or ζ_h perception of an undetected error, in the hardware hosting L, (respectively H2):

- if H1F is marked, H1 is in failure (H1 is the initializer, H2 the target):
 - if H2 is not in failure (REP2 not marked) switching is attempted, (β_h) and HSW is marked:
 - 1) Switch can succeed with probability c_h , places SSW is then marked,

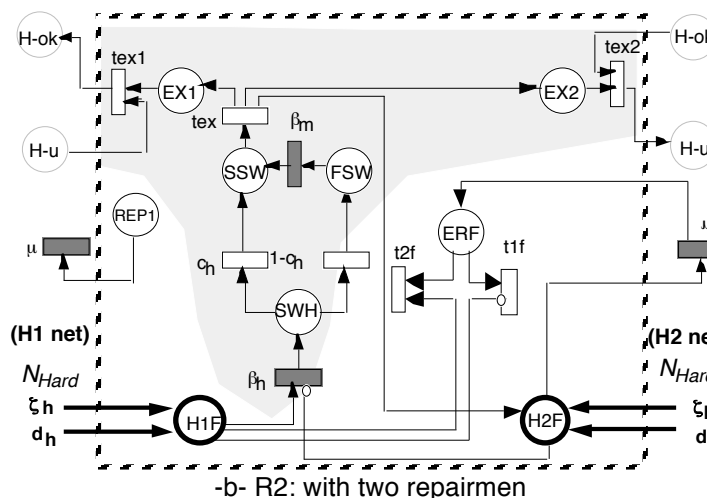
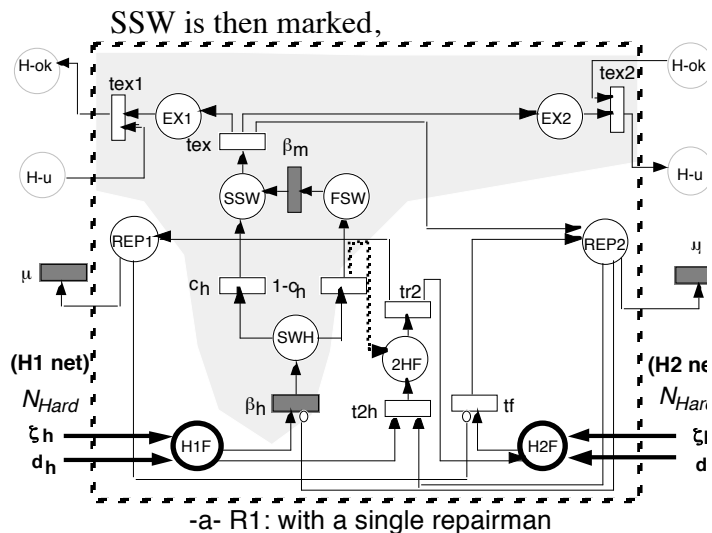


Figure 6: Hardware reconfiguration and repair nets, N_{Rep}

- 2) It can fail with probability $1-c_h$, FSW is marked and switching is done manually² (β_m), SSW is then marked; tex can be fired, places EX1, EX2 and REP2 are marked; $tex1$ and $tex2$ can be fired, they remove the token from H-u to H-ok and from H-ok to H-u, F becomes the new leader, L the new follower and it can be restarted (REP2 is marked in figure 6-a for R1, H2F marked in figure 6-b for R2),

² Other possible assumption: it can be assumed that the manual switch is not attempted. In this case, transition $1-c_h$ leads to place 2HF (dashed area in figure 6-a); place FSW and transition β_h are

for R1: if H2 is in failure (REP2 marked): t2h is fired removing the token from REP2 to 2HF; tr2 can then be fired returning a token in REP1 and one in H2F in order to repair H1, then H2, (for R2: if H2 is in failure (H2F marked): repair of H1 and H2 are enabled; at the end of H2 repair, if H1 is still under repair H2 is restarted with the leader),

for R1 if H2F is marked: H2 is in failure (H2 is alizer and target): if H1 is not in failure (REP1 not ced), tf can be fired and REP2 is marked, authorizing repair of H2; else the token stays in H2F until the end l1 repair; repair of H2 in then allowed through the ring of REP2 (for R2: repair of H2 is enabled without condition on H1).

R1 and R2: if N_{Rep} is initialed by H1 only, its is an n, activation and an authorization net; when alized by H2 only, it is an authorization net. If it is alled by H1 then H2 (or H2 then H1) it is an orization net.

Software reconfiguration net

he software reconfiguration net is given in figure 7. initialized by the marking of S1F (respectively S2F) wing the firing of transition p_s , a detected error due permanent fault or perception of an undetected error (respectively F):

S1F is marked, L is in failure (L is the initializer and the target):

if F is not in failure (RSTF not marked) switching is attempted (β_s) and SWS is marked.

- 1) Switch can succeed with probability c_s , places EXL, EXF and RSTF are then marked. Marking of EXL allows firing of tex that removes the token from S-u to S-ok in L. Marking of EXF allows the firing of one of transitions t1 to t4 that removes the token in the leader net from places S-ok, S-e, S-d or S-nd and return it to S-u. Marking of RSTF enables transition v (in F) to restart it.
- 2) Switch can fail with probability $1-c_s$, places EXF and 2SF are then marked. Marking of 2SF allows transition tr2 firing that marks places RSTL and S2F. Marking of place RSTL enables transition v in the leader to restart it. Marking of S2F allows the follower restart only after the end of the leader restart.

if F is in failure (RSTF marked) t2s is fired and 2SF is marked allowing the firing of tr2 that marks RSTL and S2F. Marking of place RSTL enables transition v in the leader to restart it. Marking of S2F allows the restart of F only after the end of the leader restart.

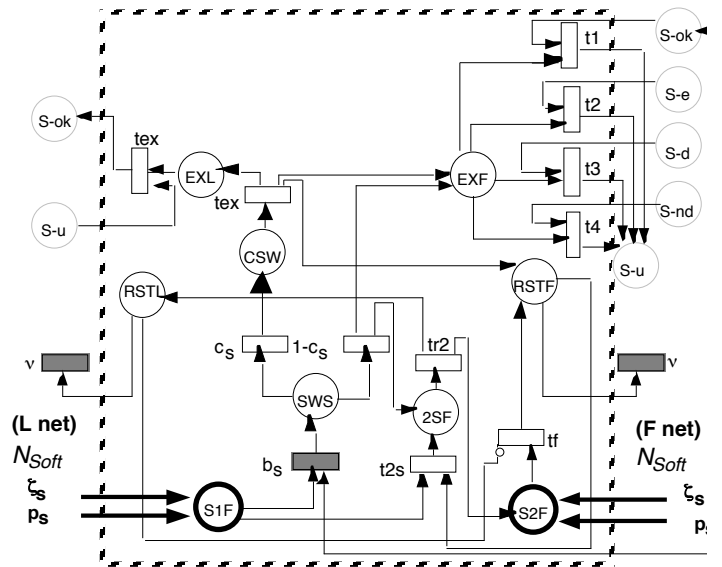


Figure 7: Software reconfiguration net, N_{Rec}

- if S2F is marked, F is in failure (F is initializer a target): if L is not in failure (RSTL not mark transition tf can be fired and RSTF is mark authorizing the restart of F; else the token stays in S until the end of L restart, restart of F is then allow via RSTF marking.

If N_{Rec} is initialed by L only, its is an action and authorization net. If N_{Rec} is initialed by F only, it is authorization net. If N_{Rec} is initialed by L then F (or then L) it is an authorization net.

4.7. Global recovery strategy net

The global recovery strategy net is initialized by N_j through F1H following the firing of tex. If F is in fail (RSTF marked) t2 removes the token from RSTF a deposits a token in RSTL and another one in S2F in or for L to be repaired first. If F is not in failure (RSTF marked) transition t1 deposits a token in CSW in or that the roles of the follower and leader to be exchanged N_{Strat} is an action net if place RSTF is marked and activation net if RSTF is not marked.

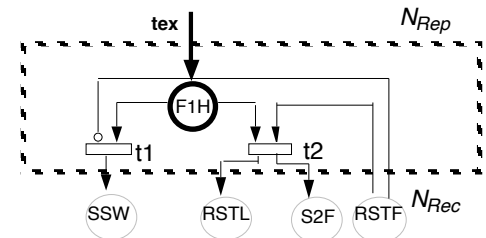


Figure 8: Global recovery strategy net, N_{Strat}

4.8. Concluding remarks and global model

Due to lack of space the formal description of previous nets is not presented. It can be checked that hardware and the software GSPNs are live and bound

erties have to be done with the adjacent nets as stated in figure 2, as follows

N_{Prop} has to be validated connected with N_{Hard} and N_{Soft} , (N'_{Prop} is identical to N_{Prop}),

N_{Stop} has to be validated with N_{Hard} and N_{Soft} ,

N_{Rep} has to be validated with two N_{Hard} ,

N_{Rec} has to be validated with N_{Soft} ,

N_{Strat} has to be validated with all the other nets (that have already been validated).

The overall model obtained by replacing the blocks of figure 2 with their GSPNs given in figures 3 to 8 has been processed by SURF-2. The marking graph has 1200 markings and the Markov chain 500 states without any aggregation due to symmetry.

It could be argued that the state space may be very large for more complex systems, this is inherent to the complexity of the system to be modeled and to the level of detail considered. The only difficulty due specifically to our modeling approach is the number of markings; it can be overcome by using an aggregation technique at the N level to suppress immediate (see e.g. [1]).

Considering again the duplex system, taking into account the fact that the transition rates associated with error detection and processing mechanisms are very high compared to failure, repair and restart rates (the durations for error detection and processing is of the order of the minutes whereas the intervals to failures are several hundreds of hours), the model can be reduced to 9 states shown in figure 9. This model is to be considered as a simplified case allowing verification of the complete model in a specific case.

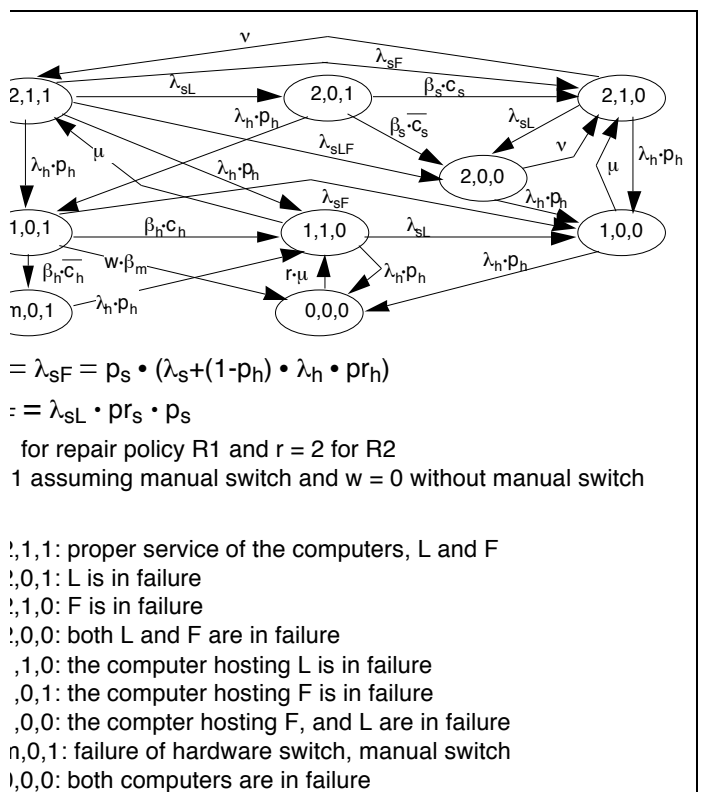


Figure 9: Reduced Markov chain of the duplex system

5. Conclusion

This work presented in this paper has allowed various types of dependencies between hardware and software components of a fault-tolerant system to be identified. These dependencies may result from functional or structural interactions as well as interactions due to reconfiguration and maintenance strategies. The dependability model of the system is obtained by the composition of the components models with the dependencies associated with the dependencies. The rules for interfacing the models have been clearly defined and formally described to build up easily validable system models. The formal description facilitates the composition of the various GSPNs.

The modeling approach has been illustrated by a simple example, including all the types of dependencies identified: the duplex system. Modeling of this system showed the strong dependency between components. For example: the activation of a temporary hardware fault may propagate an error to the hosted software component, which in turn may propagate to other components communicating with it (without being necessarily on the same computer). Thus the activation of a hardware fault, may lead to the restart of one or more software components. Even if this has already been observed on real-life systems, it has not been modeled explicitly in previous work. Also, we have shown how the modification of one or several assumptions can be performed without modifying all GSPNs, considering different repair policies and two switching policies (with and without manual switch).

The main advantage of the modeling approach, based on considering explicitly the interactions, lies in its efficiency for modeling several alternatives for the same system. These alternatives may differ by their composition (number of computers or replicas) or their organization (distribution of software components on the hardware) or by the fault tolerance and maintenance strategies. One can clearly identify from the beginning the components and interactions that are specific to those that are common to all alternatives. The GSPNs that are common are thus developed and validated only once.

This approach has been applied to the French Air Traffic Control system (the subset associated with the Flight Plan Processing and Radar Data Processing) in which twelve alternative architectures have been modeled and their unavailability compared to identify the most suitable one. Based on these results, additional and more detailed architectures have been modeled in [4]. The

ication showed all the power of the modeling approach with the explicit modeling of the interactions.

Acknowledgments

The work presented in this paper has been partially supported by the French Civil Aviation Authority and SRTI TEM while Marie Borrel was with SRTI SYSTEM and by European Commission through the OLOS Network (A traffic approach to the dependability analysis and evaluation of control systems involving hardware, software and human resources).

References

H. H. Ammar, Y. F. Huang and R. W. Liu, "Hierarchical Models for Systems Reliability, Maintainability, and Availability", *IEEE Trans. on Circuits and Syst.*, CAS-34 (6), pp. 629-38, 1987.

G. Balbo, "On the Success of Stochastic Petri Nets", in *6th International Workshop on Petri Nets and Performance Models*, (Durham, NC, USA), pp. 2-9, 1995.

C. Béounes, M. Aguéra, J. Arlat, S. Bachman, C. Bourdeau, J. E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. Moreira de Souza, D. Powell and P. Spiesser, "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software systems", in *23rd IEEE Int. Symp. Fault-Tolerant Computing*, (Toulouse, France), pp. 668-73, 1993.

M. Borrel, *Interactions between Hardware and Software Components: Characterization, Formalization and Modeling — Application to CAUTRA Dependability*, PhD Dissertation, N°96-001, In French, 1996.

A. Costes, C. Landrault and J.-C. Laprie, "Reliability and Availability Models for Maintained Systems Featuring Hardware Failures and Design Faults", *IEEE Trans. on Computers*, C-27 (6), pp. 548-60, 1978.

J. B. Dugan and M. Lyu, "System-level Reliability and Sensitivity Analysis for Three Fault-tolerant Architectures", in *4th IFIP Int. Conference on Dependable Computing for Critical Applications*, (San Diego), pp. 295-307, 1994.

W. R. Elmendorf, "Fault-tolerant Programming", in *2nd IEEE Int Symp. Fault-Tolerant Computing*, (Newton, Massachusetts), pp. 79-83, 1972.

J. Gray, "Why Do Computers Stop and What Can be Done About it ?", in *5th Int. Symp. on Reliability in Distributed Software and Database Systems*, (Los Angeles, CA), pp. 3-12, 1986.

K. Kanoun, M. Borrel, T. Moreteveille and A. Peytavin, "Modeling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System", in *26th Int. Symp. Fault-Tolerant Computing (FTCS-26)*, (Sendai, Japan), pp. LAAS-Report, 95-515, 1996.

J.-C. Laprie, "Trustable Evaluation of Computer Systems Dependability", in *Applied Mathematics and Performance/Reliability Models of Computer/ Com-*

- [11] J.-C. Laprie (Ed.), *Dependability: Basic Concepts and Terminology*, Dependable Computing and Fault-Tolerant Systems, 5, 265 p., Springer Verlag, Wien-New York, 1992.
- [12] J.-C. Laprie, "On The Temporary Character of Operation-persistent Software Faults", in *4th Int. Symp. on Software Reliability Engineering*, (Denver, Colorado), pp. 125, 1993.
- [13] J.-C. Laprie and K. Kanoun, "X-ware Reliability and Availability Modeling", *IEEE Trans. on Software Engineering*, SE-18 (2), pp. 130-47, 1992.
- [14] J.-C. Laprie, K. Kanoun, C. Béounes and M. Kaânic, "The KAT (Knowledge-Action-Transformation) Approach to the Modeling and Evaluation of Reliability and Availability Growth", *IEEE Trans. Software Engineering*, SE-17 (4), pp. 370-82, 1991.
- [15] J. K. Muppala, A. Sathaye, R. Howe, C and K. Trivedi, "Dependability Modeling of a Heterogeneous VAX-cluster System Using Stochastic Reward Nets". *Hardware and Software Fault Tolerance in Parallel Computing Systems* (D. Avresky, Ed.), pp. 33-59, 1992.
- [16] P. I. Pignal, "An Analysis of Hardware and Software Availability Exemplified on the IBM-370 Communication Controller", *IBM Journal of Research and Development*, 32 (2), pp. 268-78, 1988.
- [17] D. Powell, "Distributed Fault Tolerance: Lessons from Delta-4", *IEEE Micro*, 14 (1), pp. 36-47, 1994.
- [18] W. Sanders and J. Meyer, "Reduced Base Model Construction Methods for Stochastic Active Networks", *IEEE Trans. on Selected Areas in Communications*, 9 (1), pp. 25-36, 1991.
- [19] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1999.
- [20] G. E. Stark, "Dependability Evaluation of Integrated Hardware/Software Systems", *IEEE Trans. Reliability*, R-36 (4), pp. 440-4, 1987.