Modelling the dependability of CAUTRA, a subset of the French air traffic control system

Karama Kanoun, Marie Borrel, Thierry Morteveille, Alain Peytavin

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

LABORATOIRE D'ANALYSE ET D'ARCHITECTURE DES SYSTEMES

Modeling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System

Karama Kanoun, Marie Borrel, Thierry Morteveille, Alain Peytavin

LAAS Report no. 95.515 December 1995

Modeling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System

Karama Kanoun[◊] **Marie Borrel**[◊*] **Thierry Morteveille**[*] **Alain Peytavin**[+]

[+]CENA
7, Avenue Edouard Belin
31055 Toulouse Cedex - France

[◊]LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex - France

* SRTI SYSTEM
Rue Max Planck B.P. 457
31315 Labège Cedex - France

Abstract:

The aim of our work is to provide a quantified means helping in the definition of a new architecture for CAUTRA, a subset of the French Air Traffic Control system. To do this, we define alternative architectures for the CAUTRA whose availability is compared in order to select the architecture with the highest level of availability. Modeling is carried out following a modular and systematic approach, based on the derivation of block models at a high level of abstraction. In a second step, the blocks are replaced by their equivalent Generalized Stochastic Petri Nets to build up the detailed model of the architecture. Block models allow identification of those blocks that can be re-used in several architectures. Modularity and re-usability make this approach very efficient; moreover the progressive derivation and validation of the models considerably enhance confidence in the resulting models. Emphasis is placed on modeling interactions between hardware and software components. The evaluations performed permit identification of a set of architectures whose availability meets the dependability requirements and also identification of the best architecture among this set.

Key Words:
Dependability modeling, Generalized Stochastic Petri Nets, Markov Chains, model validation.

Approximate number of words: 7900

This paper has been cleared from authors' affiliations.

ABSTRACT:

The aim of our work is to provide a quantified means helping in the definition of a new architecture for CAUTRA, a subset of the French Air Traffic Control system. To do this, we define alternative architectures for the CAUTRA whose availability is compared in order to select the architecture with the highest level of availability. Modeling is carried out following a modular and systematic approach, based on the derivation of block models at a high level of abstraction. In a second step, the blocks are replaced by their equivalent Generalized Stochastic Petri Nets to build up the detailed model of the architecture. Block models allow identification of those blocks that can be re-used in several architectures. Modularity and re-usability make this approach very efficient; moreover the progressive derivation and validation of the models considerably enhance confidence in the resulting models. Emphasis is placed on modeling interactions between hardware and software components. The evaluations performed permit identification of a set of architectures whose availability meets the dependability requirements and also identification of the best architecture among this set.

Key words:
Dependability modeling, Generalized Stochastic Petri Nets, Markov Chains, model validation.

1. Introduction

Due to the growth in air traffic and the saturation of computational facilities, the French Directorate of Air Navigation has commissioned the design and implementation of new architectures for the Air Traffic Control system (ATC). The work presented in this paper is part of an overall program aimed at ATC automation. It more specifically addresses the sub-system referred to as CAUTRA (Coordinateur AUtomatique du TRafic Aérien). The ultimate aim is to provide a quantified means helping in the definition of a new architecture. To do this, starting from the current system, we define alternative architectures whose availability is compared.

CAUTRA is a distributed fault-tolerant system whose functions are vital to ATC. It specifically belongs to the category of real-time computer systems demanding a high level of availability. Fault tolerance techniques (i.e., hardware redundancy and software replication) enable the availability of this system to be at the required level. The complexity of the system behavior results from several interactions between hardware components and software replicas (e.g., error propagation, stopping of the software replicas following a failure of the hosting hardware computer, or sharing of common resources). These interactions induce dependencies that are usually stochastic in nature making modeling more difficult: thus the associated models have to account for the components' behavior and their interactions. To do this, we follow a modular and structured modeling approach based on Generalized Stochastic Petri Nets (GSPNs) in which interactions are explicitly considered. Emphasis is put on clearly defining the interactions between the hardware and software components and on deriving GSPNs that are as generic as possible so as to be used in the largest number of alternatives.

Several papers dealt with the dependability of ATC systems. These are mainly focused on the specification or design of fault tolerance procedures [1, 3, 11, 17], while papers on dependability evaluation are few (see e. g., [14] where a qualitative evaluation of an ATC system is performed). Likewise, a number of papers dealing with performance and dependability modeling of real-life systems using Markov chains, GSPNs or their offspring's have been published [7, 23, 25, 27-30, 33, 34]. Most of them only consider the hardware part of a system and, when the software is considered, the interactions between hardware and software are not explicitly modeled. Even though dependability evaluation of combined hardware and software systems are not yet of current practice, some relevant papers have been published [9, 12, 21, 22]. To the best of our knowledge, detailed modeling of the interactions between hardware and software components has not been previously investigated despite its importance for fault-tolerant systems.

The paper is organized in seven sections. Section 2 describes CAUTRA. The modeling approach and assumptions are presented in Section 3. GSPNs of the most critical function of CAUTRA are detailed in Section 4. Section 5 outlines the benefits and drawbacks of the modeling approach. Section 6 discusses some numerical results and Section 7 concludes.

2. System description

CAUTRA gathers together the computerized processing means for flight plans and the radar data of a Regional Center for Air Navigation. Two main functions can be distinguished within CAUTRA: Flight Plan Processing (FPP) and Radar Data Processing (RDP). The former handles and updates the flight plans in the Regional Center. It provides air traffic controllers with the data regarding the planes crossing their airspace and handles any information that may be supplied to another air traffic controllers via the Digitatron touch-sensitive screen. Based on the radar data, the RDP builds up a synthetic picture which is representative of the air traffic situation. Through the RDP-FPP dialogue, flight plan correlation allows the RDP to enhance the picture by supplying more data derived from the flight plans to those planes detected by the radar. We first present the current architecture of CAUTRA; then the alternatives are defined taking the latter as reference.

The **current architecture** comprises two Data General computers, DG1 and DG2 (duplex architecture). The software of each application is replicated leading to four components: RDP principal (denoted RDPpal), RDP standby (RDPsec), FPP principal (FPPpal) and FPP standby (FPPsec). Replicas are distributed as follows: RDPpal and FPPsec run on the same computer while RDPsec and FPPpal run on the other. FPPpal carries out a preliminary processing which is transmitted to FPPsec; also FPPpal dialogs both ways with RDPpal and RDPsec for the flight plan correlation. Connections between the four components are shown in figure 1.

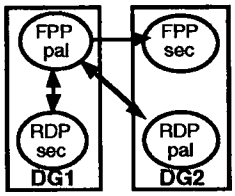


Figure 1: Current system organization

Error processing in each software replica (provided by exception handling mechanisms) allows recovery of temporary software faults. For each application, permanent faults in pal replica are tolerated by switching from the replica pal to the replica sec. Reconfiguration is carried out as follows: after switching the roles of the replicas and restart of the sec replica, the replicas switch back to resume their initial roles as shown in Figure 2. Also, the switch of RDPpal is required after the failure of the communication medium (without a failure of FPPpal).

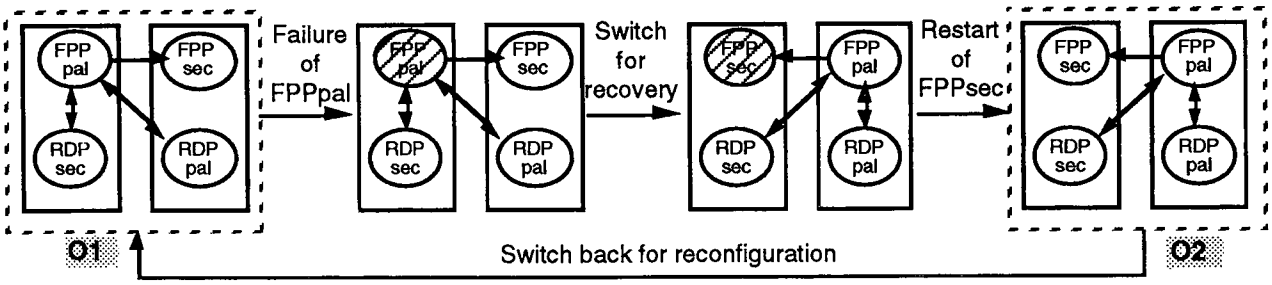


Figure 2: Reconfiguration after software failure

Starting from this architecture, several **alternative architectures** are proposed with the same composition but differing in terms of organization (distribution of the software replicas onto the hardware components), the software reconfiguration and the hardware fault tolerance procedures.

Based on the current **organization**, denoted O1, we define organization O2 in which RDPpal and FPPpal are hosted by the same computer. Both O1 and O2 are indicated on figure 2; the main difference lies in the communication means between RDPpal and FPPpal: for O2 this communication is internal whereas for O1 it is achieved through an external communication medium.

The existing **software reconfiguration** procedure, denoted R1, requires switching back for reconfiguration. A second reconfiguration procedure is defined, denoted R2, in which the replicas retain their new roles after replica sec restart (the switch back from O2 to O1 in figure 2 is not performed). Note that when R2 is considered, whatever the initial organization, the system alternates between O1 and O2 after recovery of software permanent failures.

The presence of a third computer in the center allocated to background applications can be taken advantage of and used as a spare. Two **hardware fault tolerance** procedures denoted S1 and S2 corresponding to using the spare respectively after the first computer failure or after failure of both of them are considered. Table 1 summarizes the organization, the software reconfiguration and the hardware fault tolerance procedures. Combining the 2 organizations with the 2 reconfigurations and the 3 hardware fault tolerance procedures, leads to 12 alternatives whose availability is evaluated.

Table 1: Organization, reconfiguration and fault tolerance procedures

Organization

O1	FPPpal and RDPsec onto a computer, FPPsec and RDPpal onto the other
O2	FPPpal and RDPpal onto a computer, FPPsec and RDPsec onto the other

Software reconfiguration

R1	after switching the roles of replicas and restart of the sec replica, the replicas switch back for reconfiguration
R2	after switching the roles of replicas and restart of the sec replica, the replicas retain their new roles

Hardware fault tolerance

S0	current hardware fault tolerance procedure (0 spare)
S1	switching of the spare to the main applications is performed after DG1 or DG2 failure (after 1 failure)
S2	switching of the spare to the main applications is performed after DG1 and DG2 failures (after 2 failures)

3. Modeling approach and assumptions

We assume that software and hardware components are in stable reliability, i.e., the failure rates are constant. With respect to the rates associated with hardware maintenance, software restart and fault tolerance procedures, the duration of these procedures is short relative to the times to failures, and previous studies have shown constant rates to be a good assumption [18]. The evaluations are thus based on Markov processes. However, model construction is based on GSPNs due to their ability to cope with modularity and hierarchy. The underlying Markov chains are obtained from the GSPNs. The measures evaluated are unavailability of RDP and unavailability of FPP.

The difficulty of modeling stems from the conjunction of two things: the numerous interactions between the hardware and software components and the number of alternative architectures to be

modeled. Using an ad hoc approach would be cumbersome. These considerations urged us to use a modular and hierarchical approach, taking advantage of the similarities between the alternatives. This is still reinforced by the fact that the alternatives have the same composition and many interactions between components are the same in many alternatives.

From the system composition and the interactions between components, a high level behavioral model (referred to as **block model**) composed of blocks linked by arrows is derived for each alternative. A block represents a GSPN describing either a component behavior (component net) or an interaction (dependency net); the arrows indicate the direction of the links between the nets. The GSPNs of the blocks are derived in a second step to form the global model; i.e., the GSPN of an architecture is thus obtained by **composition** of the GSPNs of the components with the GSPNs representing their interactions. The block models of all alternatives are derived together to clearly identify the blocks that are common or similar in several alternatives, for re-use purpose. Re-use of the blocks is one of the main advantages of the modeling approach. Also, this approach allows for a progressive construction of the global model so as to master its complexity as in [27, 30].

The first step in constructing the models consists thus in identifying the interactions between the components to define the dependency blocks which will link the component blocks. In the remainder of the section, the interactions between the software and hardware components of CAUTRA are first analyzed, then the block models of the alternatives are presented.

3.1 Interactions between the hardware and software components

These 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 [13, 16, 20, 32], both permanent and temporary faults are considered.

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 component may ***propagate*** to the hosted software replicas,
  - an error due to the activation of a permanent fault in a hardware component leads to ***stop*** the hosted software replicas that are restarted after the end of hardware repair.
- Following activation of a software fault: owing to the dialog between the replicas, an error in a replica due to a permanent fault — usually referred to as solid fault — may ***propagate*** to the replicas with which it dialogs [24] (it is assumed that errors due to temporary faults — usually referred to as soft faults — are confined and do not propagate to the other replicas)[1]..
- Following failures of the communication medium: the system has to switch from organization O1 to O2 if it is in O1, and the switching from O2 to O1 is not allowed as long as the communication medium is failed.

---

[1] It is worth noting that replicas pal and sec do not perform exactly the same tasks at a given time. We assume that common mode failures are only induced by error propagation due to the dialog between the two replicas.

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

- Between two software replicas: dependency due to fault tolerance of permanent software faults, i.e., ***reconfiguration*** from sec to pal, following RDPpal or FPPpal failure.
- Between two hardware computers: dependency due to ***fault tolerance*** and ***repair***.
- Between all components: coordination of fault tolerance and maintenance actions to form a global ***recovery strategy*** when several components are in failure. For example, in case of failure of the computer hosting the pal replica and failure of the sec replica, sec is restarted as pal, the new sec is restarted after computer repair.

3.2 Block models of the CAUTRA architectures

Consider first the **current architecture**. Table 2 lists the name of the nets associated with the component blocks. Dependency blocks are directly derived from the dependencies identified above; these dependencies are recalled in table 3 together with the names of the associated nets. Figure 3 shows the block models for the RDP and FPP: all blocks of RDP are used for FPP as well and FPP has two extra blocks, ($2xN'_{Prop}$) for error propagation between replicas. These models are obtained by **composition** of the block nets introduced in tables 2 and 3.

Due to the exchange of information between RDP and FPP (with possible error propagation), the dependability of RDP and FPP cannot be evaluated separately: a global model has to be derived as shown in figure 4. The latter is obtained by combining the models of figure 3 and adding the following blocks: error propagation between RDP and FPP replicas ($4xN'_{Prop}$), the global recovery strategy (N_{Strat}, N_{R1}, N_{Com}), and the communication medium, N_C. The set of blocks (N_{DG1}, N_{DG2}, N_{Rep}) associated with the computers and their repair is considered only once in the global model.

This model has been derived after some iterations and refinements between the block models of the various alternatives. In particular, the global recovery strategy could have been modeled by a single block. It has been split into three blocks (N_{Strat}, N_{R1}, N_{Com}) only for re-usability and clarity: N_{Strat} and N_{Com} are the same for all architectures, they are distinguished only because their roles are different; N_{R1} is to omitted when R2 is considered.

With respect to the **other alternative architectures**, using the notations of table 1, let A.o.r.s denote an architecture; o=O1,O2; r=R1,R2; s=S0,S1,S2. For the sake of simplicity, this is shortened as: o = 1, 2; r = 1, 2; s = 0, 1, 2. We give hereafter the list of blocks that have to be added, removed or simply adapted for each architecture, taking the current one, A.1.1.0, as reference (figure 4):

- A.2.1.0: adapt N_{R1} and N_{SynFPP} to comply with the O2 assumption.
- A.1.2.0 and A.2.2.0: remove N_{R1} (which is associated with R1 only), and adapt N_{SynFPP} for A.2.2.0 to comply with the O2 assumption.
- Set (R2, S1 and R2, S2) = {A.1.2.1, A.2.2.1, A.1.2.2 and A.2.2.2}: remove N_{R1}, adapt N_{SynFPP} for A.2.2.1 and A.2.2.2 to comply with the O2 assumption, adapt N_{Rep} and add N_{Spare}.
- Set (R1, S1 and R1, S2) = {A.1.1.1, A.2.1.1, A.1.1.2 and A.2.1.2}: adapt N_{R1} and N_{SynFPP} for A.2.1.1 and A.2.1.2 according to the O2 assumption, adapt N_{Rep} and add a block

corresponding to the behavior of the spare computer N_{Spare} which is directly linked to the new N_{Rep}.

As a result, modeling the 11 other alternative architectures requires only the addition of one component net (N_{Spare}) and adaptation of 6 dependency nets (those associated with the reconfiguration and fault tolerance strategy).

Table 2: Component Nets

N_{DG1} , N_{DG2}	model computers DG1 and DG2 (they are identical)
N_{RDP1}, N_{RDP2}, N_{FPP1}, N_{FPP2}	model RDP and FPP replicas (they are identical)
N_C	models communication medium

Table 3: Dependency Nets

N_{Prop}	models the propagation of a hardware error to the hosted software replica
N_{Stop}	models the software stop after activation of a permanent fault in the hosting hardware
N'_{Prop}	models propagation of a software error to a communicating software replica
N_{Com}	models the impact of the failures of the communication medium on system organization
N_{RecRDP}, N_{RecFPP}	model RDP and FPP software reconfiguration from pal to sec (identical)
N_{Rep}	models hardware fault tolerance and repair (sharing of a repair man)
N_{SynRDP}, N_{SynFPP}	model the synchronization between hardware and software recovery actions (identical)
N_{Strat}	models the global reconfiguration strategy according to all components and resources states
N_{R1}	models the switch back in the case of reconfiguration R1

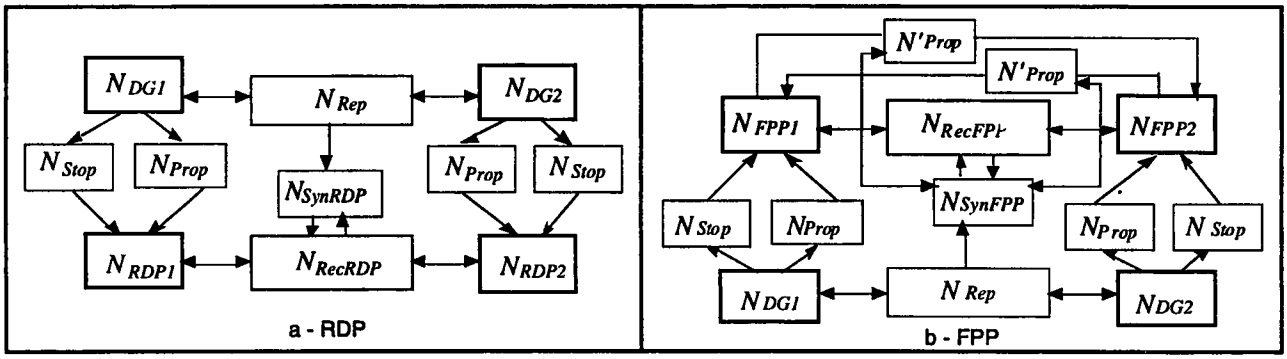


Figure 3: Block models of RDP and FPP considered separately

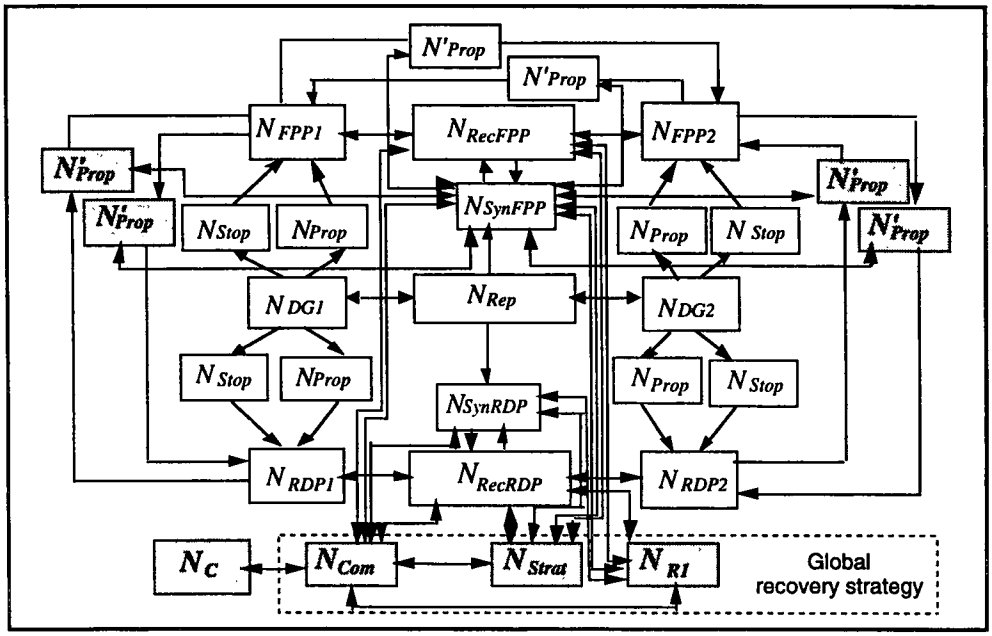


Figure 4: Block model of the current architecture (A.1.1.0)

4. GSPN construction

CAUTRA modeling is based on composition of the GSPNs of the various blocks to form the global GSPN. It is not possible to present the GSPNs of all the blocks related to all architectures. Rather a subset of them is selected to illustrate what precedes, that is, those related to the most critical function: RDP; the model considered is that of figure 3-a.

To assist re-usability, we defined rules for constructing the GSPNs of the blocks. Together with the conventional rules of GSPNs, these rules allow for an easy interfacing of the dependency nets — a prerequisite for modularity, hierarchical modeling and re-usability. These rules are formalized in [6] and can be summarized as follows :

- A **component net** has to be standalone, live and bounded; it models the behavior of a component as resulting from the activation of its own faults and from local error detection, fault tolerance mechanisms, and repair or restart actions.
- A **dependency net** is linked to at least two nets, initializing and target nets; it is formally defined according to the subsequent rules:
  - it is activated by one or more **initializing net(s)**, via **entry place(s)** after firing of specific transitions (**firing transitions**) in the initializing net, the initial marking of these places is 0;
  - it has internal transitions, independent of or conditioned by the marking of either of or both initializing and target nets;
  - it has well defined interfaces with **target net(s)**;
  - additional rules control the creation and absorption of the tokens, for instance each token created at the initialization of the dependency net is absorbed when leaving the net;
  - dependency nets should not alter the structure of the component nets, this is very important because, several dependency nets can be connected to the same component net.

The component and dependency nets of the RDP application, constructed according to these rules, will now be presented.

4.1 Hardware and software components nets

The hardware and software component nets are given in Figure 5.

The hardware model (for computers DG1 and DG2) 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 non detected error, resulting from a permanent fault, may be perceived later (perception rate ζ_h).
- The repair rate (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 exception 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, or 2) if the fault is permanent (probability p_s); the software has to be restarted (restart rate: ν) to eliminate its effects.($1-p_s$) measures the efficiency of fault containment procedures [16, 19].
- The effects of a non detected error may be eliminated (rate δ_s), or perceived (perception rate ζ_s), in which case the software replica has to be restarted.

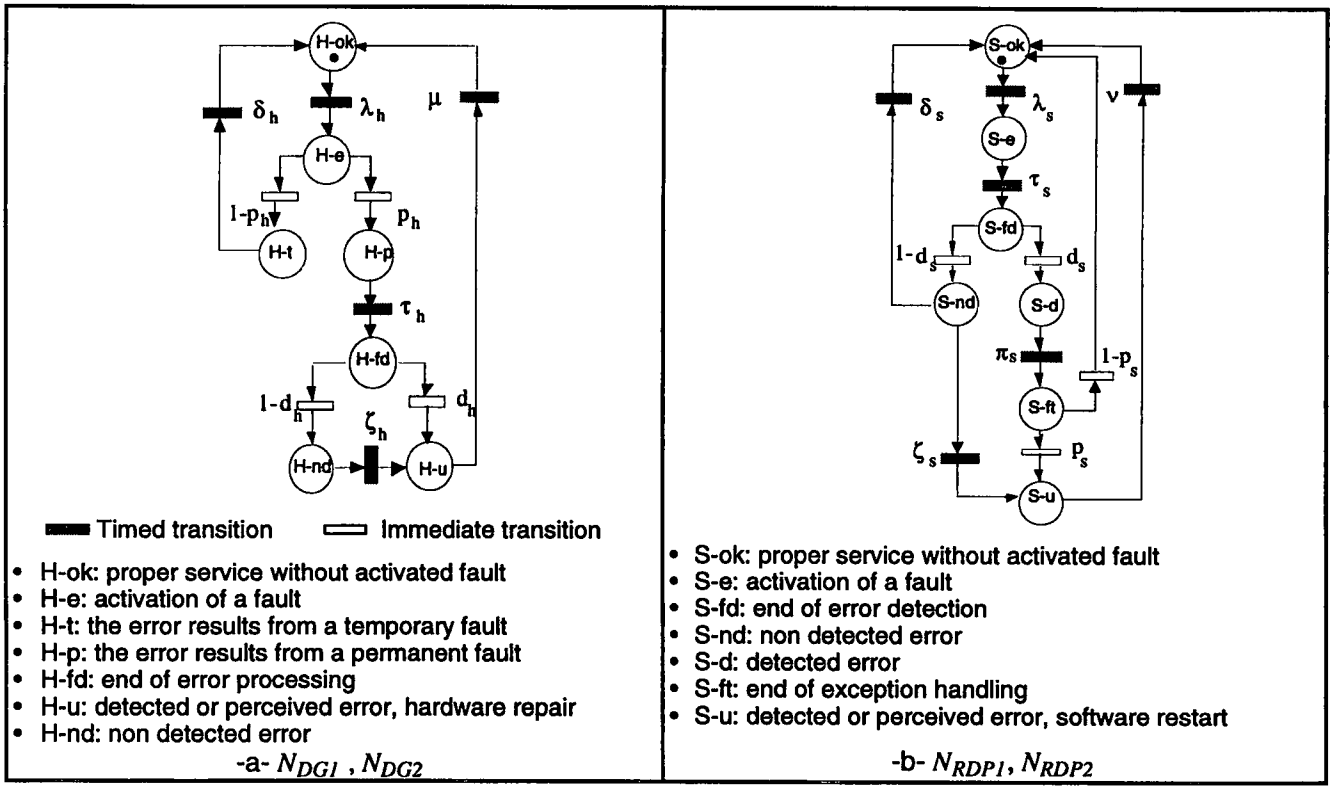


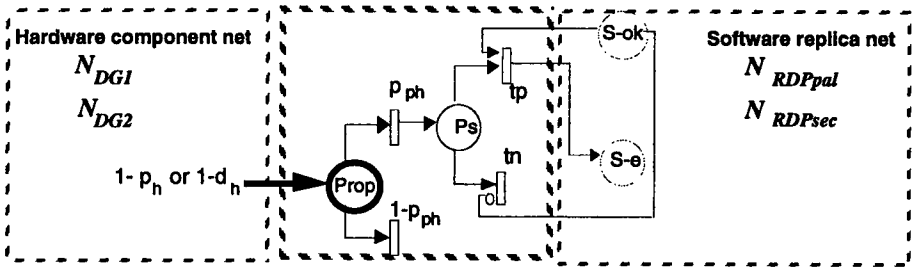
Figure 5: Hardware component and software replica nets

The difference between these nets lies in that for hardware, temporary and permanent faults are differentiated by their respective consequences following activation, whereas for software, they can only be distinguished after specific processing [20].

4..2 Error propagation net, N_{Prop}

The error propagation net of Figure 6 is initialized by the marking of place Prop due to the firing of transition $1-d_h$ (non detected error) or of transition $1-p_h$ (an error due to a temporary fault) in the hardware net (initializing net). With probability $1-p_{ph}$, the error is not propagated and with probability p_{ph} it is propagated. The effect on the software net (target net) is to move the token from S-ok to S-e if it is in S-ok; processing of the induced error is carried out in the same way as when the fault is activated without propagation (i.e., through λ_s in figure 5-b). This is summarized as follows:

Initializing net: hardware component net **Entry place:** Prop	**Firing transitions:** (1-d_h) or (1-p_h) **Actions:** move the token from S-ok to S-e (probability pp_h and transition tp) or no action (transitions (1-p_{ph}) and tn are without actions)	**Target net:** software replica net



The entry places are indicated in bold

Figure 6: Error propagation net, N_{Prop}

4.3 Software stop net, N_{Stop}

The software stop net given in Figure 7 is initialized by the marking of place A) STP following firing of transition d_h (detection of an error) or transition ζ_h (perception of a non-detected error) in the hardware component or b) AR at the end of hardware repair to enable software restart. Its action on the software component net is to remove the token from any stable place (followed by timed transitions) and to deposit it in S-u where the replica is unavailable. Replica restart is enabled only after completion of the hardware repair (place AR marked). This net is also initialized by the software net via the marking of AR after detection of an error due to a permanent fault or after the perception of a non-detected software error, to enable its own restart.

Initializing net: hardware component net **Entry place:** STP	**Firing transitions:** d_h or ζ_h **Actions:** move the token from a stable place to S-u (transition t1, t2, t3 or t4) or remove the token from AR (transition ts)	**Target net:** software replica net
Initializing net: hardware component net **Entry place:** AR	**Firing transitions:** μ **Action:** enable software restart (transition v)	**Target net:** software replica net
Initializing net: software replica net **Entry place:** AR	**Firing transitions:** p_s or ζ_s **Action:** enable software restart (transition v)	**Target net:** same software replica net

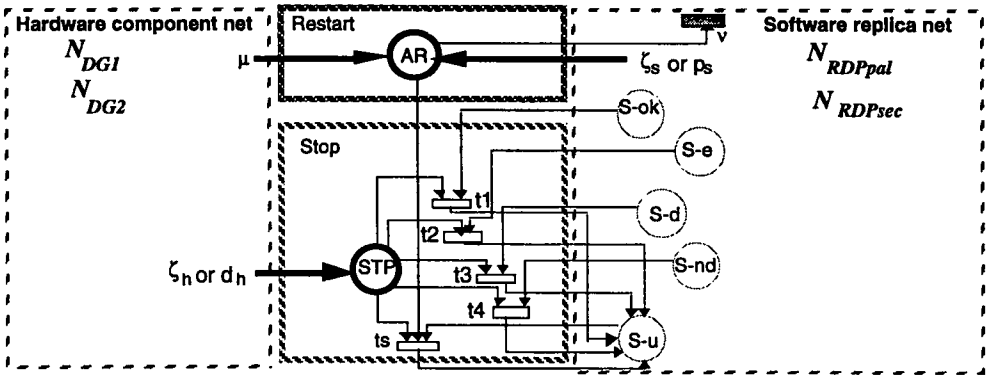


Figure 7: Software stop net, N_{Stop}

4.4 Hardware repair net, N_{Rep}

The repair net of Figure 8 is initialized by the marking of place H1F or H2F after a detected error due to the activation of a permanent fault or after perception of a non-detected error in computers DG1 or DG2 respectively.

Following the marking of H1F (the scenarios after H2F marking are symmetrical):

- If DG2 has not failed (place R2 not marked) a token is put in **HD1** in the synchronization net, another one is put in WT1 (transition tH1). The repair man is available (Rep marked), the tokens are removed from places Rep and WT1. Place R1 is marked with the firing of transition t1 in order to authorize repair of DG1. At the end of repair, the repair man and DG1 are available (places Rep (in N_{Rep}) and H-ok (in N_{DG1}) marked after the firing of μ).
- If the DG2 has failed before DG1 failure, place **HD3** is marked in the synchronization net and a token is put in WT1 (it stays there till the end of DG2 repair).

Initializing net: hardware component nets	**Firing transitions:** p_h or ζ_h	**Target net:** hardware component nets
Entry places: H1F and H2F	**Action:** put a token in R1 or R2 to enable the firing of μ	

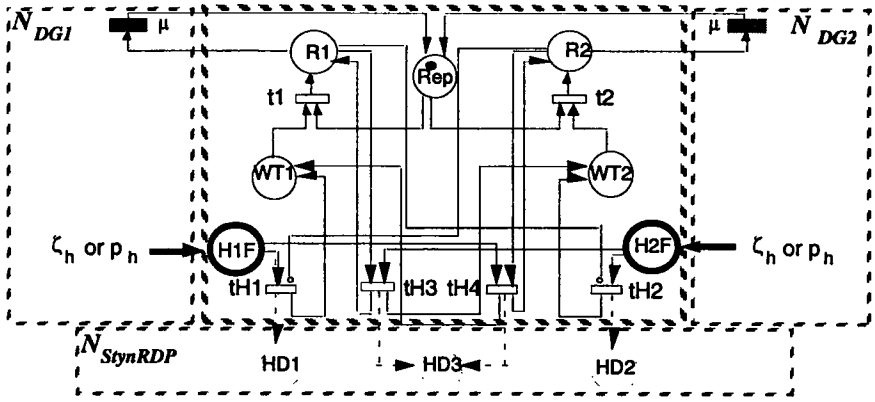


Figure 8: Hardware repair net, N_{Rep}

4.5 Software reconfiguration net, N_{RecRDP}

The software reconfiguration net given in Figure 9 is initialized by the marking of places S1F or S2F following detection of an error due to the activation of a permanent fault or following perception of a non detected error in RDP1 or RDP2 (transitions p_s or ζ_s).

When S1F is marked (for S2F, replace 1 by 2) :

- if replica 2 is not failed (RST2 marked):
  - if replica 1 is the sec, it can be restarted (RST1 is marked by t3).
  - if replica 1 is the pal, switching is enabled (SW marked, switching rate $1/\beta_{RDP}$):

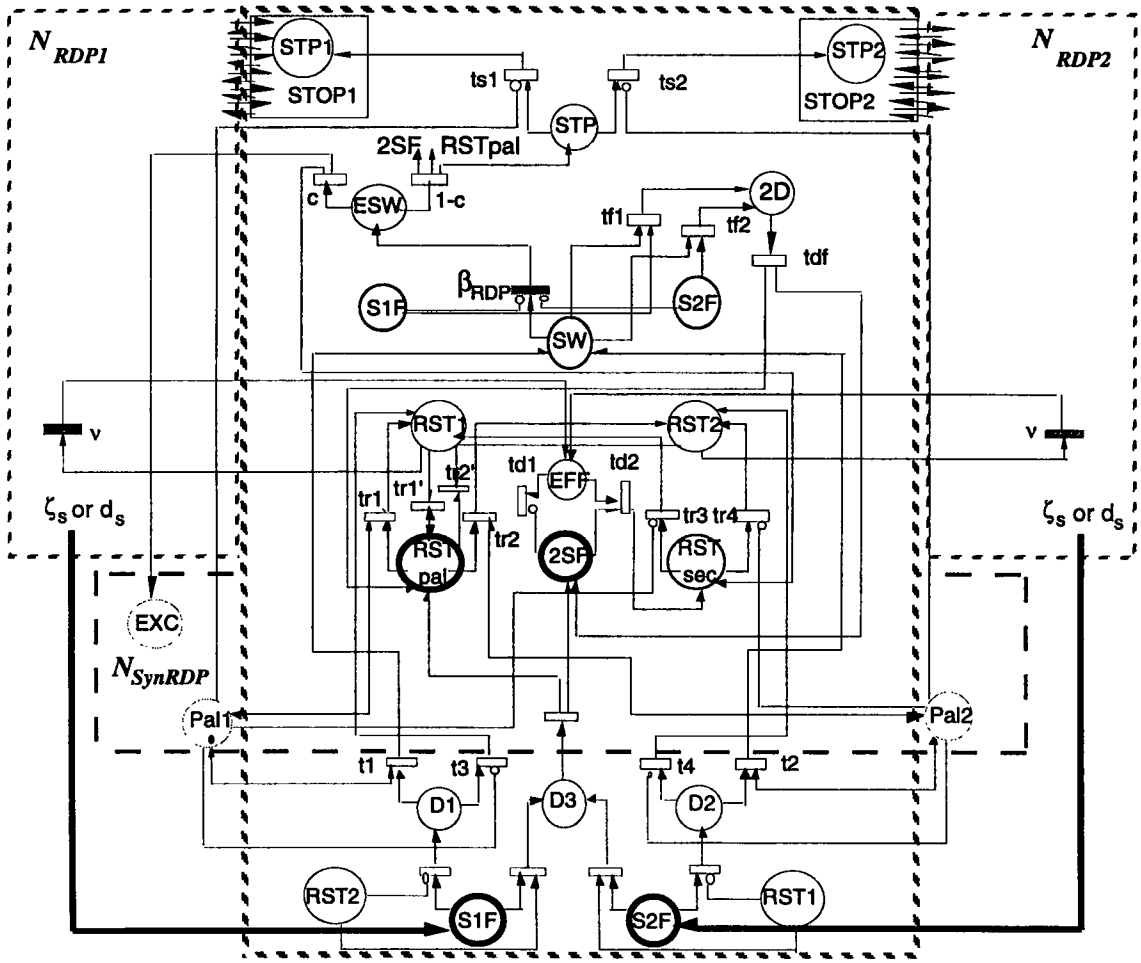
1) Switching may succeed with probability c, the replicas exchange their roles via the synchronization net (marking of **EXC**) and sec (former pal) is restarted (marking of RSTsec).
2) Switching may fail with probability 1-c, restart of the pal is enabled first (RSTpal and 2SF are marked), the sec is stopped via a stop net identical to N_{Stop} (via marking of STP2).
3) If the sec fails before the end of switching, pal is restarted first (marking of 2SF and RSTpal by tf2 followed by tdf).

- in case of replica 2 failure, pal is restarted then sec (RSTpal and 2SF are marked by t2f).

When 2SF and RSTpal are both marked, if replica 1 is pal it is restarted (marking of RST1); at the end of the restart, EFF is marked following the firing of transition v, the sec is then restarted (RSTsec is marked following the firing of td2 and RST2 is marked following the firing of tr4).

Also, the software reconfiguration net can be initialized by the synchronization net via S1F, S2F, RSTpal and 2SF, the subsequent scenarios are those described above: switching or restart of replicas depending on their roles and their states. The firing transitions are identified in Section 4.6 and the corresponding entry places are indicated in bold in the text.

Initializing net: software replica nets **Entry places:** S1Fand S2F	**Firing transitions:** d_s or ζ_s **Actions:** a token is put in RST1 (or RST2) to enable the restart of the replica(s) a token is put in STP2 (or STP2) to stop the other replica	**Target nets:** Software replica nets
Initializing net: software replica nets **Entry places:** S1F and S2F	**Firing transitions:** d_s or ζ_s **Actions:** a token is put in EXC to exchange the roles of replicas	**Target nets:** Synchronization net



Places S1F, S2F, RST1 and RST2 are pictured in several locations to make the net more readable.

Figure 9: Software reconfiguration net, N_{RecRDP}

4.6 Synchronization between software reconfiguration and hardware repair, N_{SynRDP}

The Synchronization net of Figure 10 is initialized: from the hardware repair net via places HD1, HD2 or HD3, following DG1 or DG2 failure (transition tH1 or tH2) or both (transitions tH3, tH4), or from the software reconfiguration net via place EXC after a successful switching (transition c).

Its role is to synchronize repair, stop, restart, switch and role exchange actions of the software replicas depending on all the component states. A place is attached to each replica, denoted Pal1 and Pal2: their markings indicate the replica roles (1 for pal, 0 for sec). The initial markings are: 1 for Pal1 and 0 for Pal2. They are modified when replicas exchange their roles (EXC marked and te1 or te2 firing).

After the marking of HD1 (the scenarios after HD2 marking are symmetrical):

- if both software replicas were available before DG1 failure, a token is deposited in **S1F** in the software reconfiguration net (transition th0);
- if software replica 1 is not available, the token is removed from HD1 (transition th1);
- if software replica 2 is not available, place H1 is marked (transition th2 or th3), **2SF** is marked if it was empty (transition th2) and pal is restarted if replica 1 was sec (transition ts1), or replicas switch if replica 1 was pal (transition tp1 and te1 or te2).
- if switching is under processing (SW marked), if replica 1 is sec, S1F is marked (via transition tsw1) to stop the switching and restart pal first, if replica 1 is pal the switching is continued (the token is removed via transition tsw2).

Initializing net: hardware repair net **Entry places:** HD1, HD2 and HD3	**Firing transitions:** tH1 to tH4 **Action:** move a token in places RSTPal, 2SF, S1F, S2F	**Target net:** software reconfiguration net
Initializing net: software reconfiguration net **Entry places:** EXC	**Firing transitions:** c **Action:** move a token in place RSTPal to restart pal first	**Target net:** software reconfiguration net

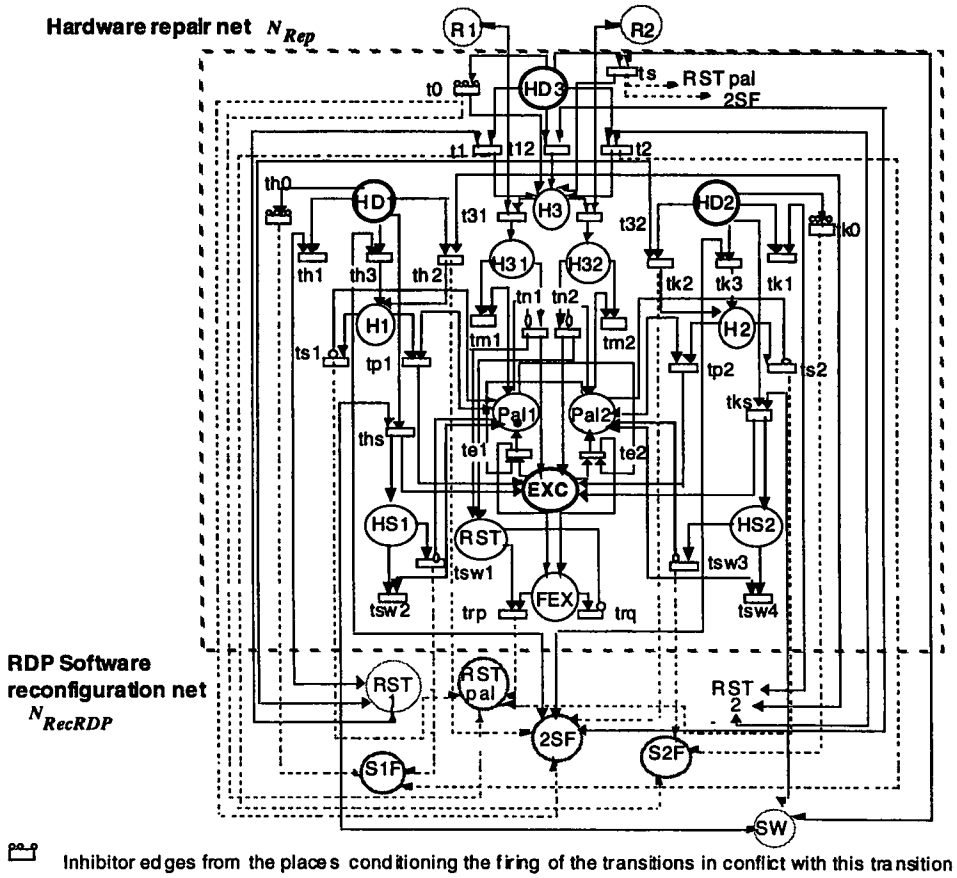


Figure 10: Synchronization net, N_{SynRDP}

After the marking of HD3, **RSTPal** and **2SF** are marked in the software reconfiguration net if they had not been marked before DG1 and DG2 failure (transitions t0, t1, t2, t12, ts) and H3 is marked. If DG1 repair has begun, H31 is marked or H32 if DG2 is being repaired. After marking H31 (replace 1 by 2 for the marking of H32): if pal runs onto DG1 the token is removed by tm1 (pal is on DG which will be repaired first); if the sec runs onto DG1, EXC and **RSTpal** are marked by tn1. The replicas exchange their roles and pal is restarted first (transition trp).

5. Benefits and drawbacks

Even though all GSPNs for all alternatives have not been presented, due to space limitations, the GSPNs of the RDP give an idea of the **complexity** of the global models of the alternative architectures. Based on the GSPNs presented in the previous section, we can outline the benefits and drawbacks of the block modeling approach.

One of the major benefits of the approach is the ease of **model validation**. Validation of the global model is performed progressively starting from the component nets and gradually adding the dependency nets. Each dependency net is validated with its initializing and target nets. Possible problems can thus be detected early before building up the whole GSPN. Dependency nets are validated only once when they are re-used without adaptation for other alternative architectures. By way of example, consider the RDP model of figure 3-a. It can be verified that the hardware and software GSPNs are live and bounded. For the dependency nets, verification of these properties can be carried out as follows: N_{Prop} is validated with N_{DG1} and N_{RDP1}; N_{Stop} is validated with N_{DG1} and N_{RDP1}; N_{Rep} is validated with N_{DG1} and N_{DG2}; N_{RecRDP} is validated with N_{RDP1} and N_{RDP2}, and N_{SynRDP} is to be validated with all the other GSPNs (i. e., the whole model, in which all the blocks except N_{SynRDP} have been validated).

Model composition and validation are facilitated by **re-usability** (with or without adaptation of the block nets). Indeed, the explicit modeling of the interactions, leads us to analyze them in detail and generally to fractionate them into elementary operations. Most of the time, the adaptation of a GSPN in order to model similar interactions consists in modifying a part of these elementary operations. This reinforces further the advantages of the modeling approach.

However, due to the presence of several immediate transitions, the **state space** of the resulting reachability graph may be very large. Fortunately, several techniques are available for suppressing the immediate transitions, (see e. g., [2, 8, 26]). Another problem arises from the fast transitions, leading to stiff Markov chains and making it more difficult to evaluate dependability. The state aggregation technique proposed in [5] and the place aggregation technique achieved at the GSPN level in [2] allow a non-stiff Markov chain with a smaller state space to be obtained. These aggregation techniques are approximations; the result accuracy is conditioned by the ratio of the slow-to-fast transition rate: the lower the ratio, the more accurate the results become [10]. Processing of the resulting models can be performed either by simulation or analytical treatments (see e.g.,[4, 15, 31]).

6. Some results

Considering the models of CAUTRA alternatives, using the composition rules of Section 4 and suppressing fast transitions[2], the number of states of the Markov chains obtained is 104 states for the architectures without a spare and 390 states for those with a spare. These models have been constructed and processed with the help of SURF-2 [4]. Several verifications have been performed to check their validity before model processing and also by sensitivity studies for semantic verifications. Examples of such studies are given in the remainder of this section.

As stated earlier, the measures chosen for evaluating the CAUTRA dependability is the unavailability of RDP[3] and FPP. In this paper, we report the respective impact of the organization, software reconfiguration and hardware fault tolerance procedures.

Additional parameters have to be introduced since they do not appear in the RDP model. They correspond to: β_h, the switching rate of computer DG1 or DG2 onto the spare, c_h, the associated coverage factor, p_{ps}: the probability of propagation of a permanent software fault (the equivalent of p_{ph} for hardware faults) and β_{FPP}, the switching rate from FPPpal to FPPsec (the equivalent of β_{RDP}).

With respect to the numerical values, some of them are derived from observations on the current system, the others are assigned nominal values from which sensitivity studies are performed. Unless otherwise stated, the numerical values considered are those of Tables 4 and 5.

Table 4: Nominal values of transition rates

λ_h	λ_{RDP} (λ_S)	λ_{FPP} (λ_S)	λ_C	$1/\mu$	$1/\nu_{RDP}$	$1/\nu_{FPP}$	$1/\beta_{RDP}$[4]	$1/\beta_{FPP}$[4]	$1/\beta_h$
0.01 / h	0.01 / h	0.05 / h	1E-5	10 h	1 mn	10 mn	1 s	1 mn	1 mn

Table 5: Nominal values of probabilities

p_h[5]	p_s	p_{ph}	p_{ps}	c	c_h	d_h	d_s
0.02	0.02	0.85	0.7	0.98	0.95	1	1

6.1 System organization

For a given hardware fault tolerance procedure, the FPP unavailability remains unaffected by the organization whatever the failure rate of the communication system. This can be accounted for by the fact that the failure of the communication between RDPpal and FPPpal does not cause FPP switching. With respect to RDP, the impact of the organization on unavailability is not easily perceived for a communication system failure rate less than 10^{-6}/h, irrespective of the hardware fault tolerance procedure. This is illustrated in figure 11 and table 6 for the current hardware fault tolerance procedure (S0). The three fault tolerance procedures behave in a similar manner.

---

2 They correspond to the hardware and software error detection and processing rates (τ_h, δ_h and τ_s, π_s, δ_s). The durations of the associated events are in the range of seconds, to be compared with the times to failures.

3 The requirement for the future architecture is that unavailability does not exceed 5 mn per year for RDP.

4 In the current system, RDP switching is automatic, whereas for FPP it is performed after acknowledgment of the operator.

5 The nominal permanent failure rate is 2 10^{-4} /h for the computers and RDP replicas, and 10^{-3} / h for FPP replicas.

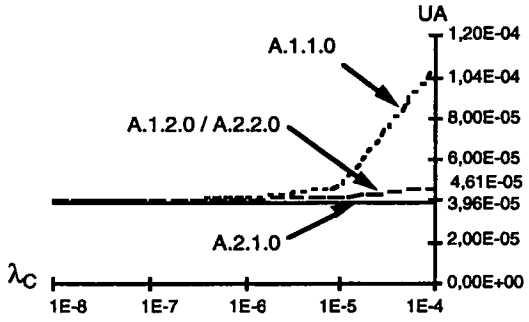


Figure 11: RDP Unavailability, for S0 (probability)

Table 6: RDP Unavailability, for S0 (per year)

λ_c	A.1.1.0	A.2.1.0	A.1.2.0 A.2.2.0
10^{-8}/h	21 mn 17 s	20 mn 49 s	21 mn 01 s
10^{-7}/h	21 mn 17 s	20 mn 49 s	21 mn 01 s
10^{-6}/h	21 mn 36 s	20 mn 49 s	21 mn 11 s
10^{-5}/h	24 mn 39 s	20 mn 49 s	22 mn 11 s
10^{-4}/h	54 mn 40 s	20 mn 49 s	24 mn 14 s
10^{-3}/h	5 h 55 mn	20 mn 55 s	25 mn 04 s

6.2 Hardware fault tolerance

Table 7 shows the FPP unavailability to be very little affected by the hardware fault tolerance procedure. Using a spare computer does not reduce unavailability by more than 20%, even for S1 where the spare is used after the failure of the first computer. On the other hand, the unavailability of RDP is quite sensitive to the variation of the hardware fault tolerance procedure, whatever the value of the communication failure rate. By way of example, table 8 shows that the unavailability of RDP is divided by almost 10 from S2 to S1. Therefore, using a spare computer, it is more advantageous to switch after the failure of DG1 or DG2 than to wait for both DG1 and DG2 to fail. Note that the dependability requirement of less than 5 mn per year for the RDP is met only for S1 whatever the organization and the software reconfiguration.

Table 7: Unavailability per year of FPP

Hardware fault tolerance	A.1.1.s	A.2.1.s	A.1.2.s	A.2.2.s
Without a spare (s = 0)	2 h 42 mn	2 h 41 mn	2 h 42 mn	2 h 42 mn
With a spare, switch after 1 failure (s = 1)	2 h 17 mn	2 h 17 mn	2 h 17 mn	2 h 17 mn
With a spare, switch after 2 failures (s = 2)	2 h 36 mn	2 h 36 mn	2 h 36 mn	2 h 36 mn

Table 8: Unavailability per year of RDP

Hardware fault tolerance	A.1.1.s	A.2.1.s	A.1.2.s	A.2.2.s
Without a spare (s = 0)	21 mn 36 s	20 mn 49 s	21 mn 11 s	21 mn 11 s
With a spare, switch after 1 failure (s = 1)	**1 mn 48 s**	**1 mn 25 s**	**1 mn 36 s**	**1 mn 36 s**
With a spare, switch after 2 failures (s = 2)	16 mn 18 s	15 mn 40 s	15 mn 56 s	15 mn 56 s

6.3 Software reconfiguration

For both RDP and FPP, with R2 the organization has no impact, whatever the hardware fault tolerance procedure (A.1.2.s and A.2.2.s in tables 7 and 8). This is due to the fact that after a long operational time, and without a strategy based on systematically exchanging the role of replicas after system reconfiguration to retain the initial distribution of the software corresponding to the architecture organization, the time spent in O1 is equivalent to that spent in O2. For RDP, the software reconfiguration procedure impacts differently the unavailability for both organizations, whatever the hardware fault tolerance procedure. For O1, moving from R1 to R2 decreases the unavailability, while for O2, it increases it (this is confirmed by the table 6). Thus, it may be inferred that for O1, the best software reconfiguration is R2, and for O2, R1.

6.4 Tradeoff between hardware coverage factor and repair duration

Table 9 shows that RDP unavailability is sensitive to the hardware fault tolerance coverage c_h, only for S1, because switching after the failure of DG1 or DG2 occurs more frequently than switching after the failure of DG1 and DG2. Also S1 and S2 unavailabilities are consistently lower than that of S0 (this is in accordance with the results of figure 11 and tables 6 and 8). Tables 8 and 9 show that for the nominal value of repair duration (10 hours) the unavailability of S1 and S2 is always lower than that of S0. Indeed fault tolerance strategy is influenced by the coverage factor, c_h, and at the same time by the repair duration, $1/\mu$ as shown by table 10 and figure 12.

When the repair duration decreases, table 10 shows that the unavailability of S0 can be slightly lower than that of S1: for $c_h = 0.5$, S0 is better than S1 for a repair duration less than 2h, whereas for $c_h = 0.8$, S0 is better when the repair duration is less than 1h 30 (the results are given for organization O1 and reconfiguration R2). Figure 12 and table 10 confirm the classical result that the benefit derived from using a spare increases when the repair duration increases; they allow quantification of this tradeoff.

Table 9 : Unavailability of RDP per year for S1 and S2 according to c_h

c_h	A.1.1.1	A.2.1.1	A.1.2.1 / A.2.2.1	A.1.1.2	A.2.1.2	A.1.2.2 / A.2.2.2
0.7	6 mn 44 s	6 mn 34 s	6 mn 07 s	15 mn 59 s	15 mn 37 s	15 mn 49 s
0.9	**2 mn 32 s**	**2 mn 27 s**	**2 mn 29 s**	15 mn 59 s	15 mn 37 s	15 mn 49 s
0.95	**1 mn 29 s**	**1 mn 25 s**	**1 mn 27 s**	15 mn 59 s	15 mn 37 s	15 mn 46 s
1	**26 s**	**23 s**	**25 s**	15 mn 59 s	15 mn 37 s	15 mn 45 s

Table 10: Unavailability of RDP per year for S0 and S1 according to c_h and $1/\mu$

$1/\mu$	c_h	A.2.1.1	A.2.1.0
1 h	0.5	2 mn 20 s	2 mn 16 s
	0.8	2 mn 18 s	2 mn 16 s
1 h15	0.5	2 mn 51 s	2 mn 45 s
	0.8	2 mn 47 s	2 mn 45 s
2 h	0.5	4 mn 21 s	4 mn 20 s
	0.8	4 mn 07 s	4 mn 20 s

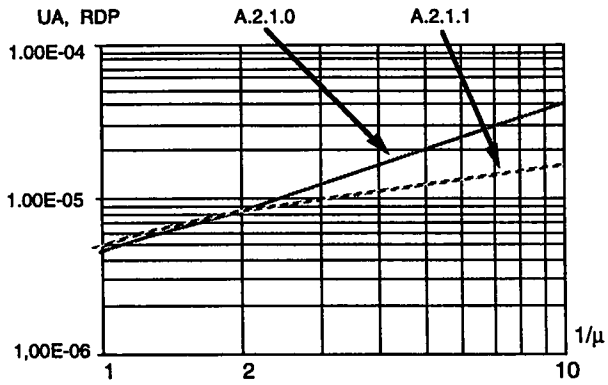


Figure 12: Unavailability of RDP for c_h= 0.5

6.5 Influence of software restart and switching durations on FPP availability

One of the main objectives is to decrease the unavailability of FPP as much as possible. Indeed availability is limited by the duration of switching from sec to pal and by the restart time (if the switching fails or after the two replicas' failure). It is not possible to have a restart time less than 10 to 5 minutes owing to the FPP data stream: it is created once, infrequently updated and difficult to reconstitute (while the RDP data stream is more frequently created, updated at a faster rate, and easier to reconstitute, leading to a short restart time of less than 3 mn). Besides the failure rate, switching time exerts a great influence. So far, the FPP switching time is about 1 mn. Tables 11 and

11 show that reducing switching time is more worthwhile than reducing the restart duration. From table 11 it is observed that if FPP switching time were the same as that of RDP (1 s), unavailability would be almost one hour per year for S0 and almost half an hour per year for S1. From 10 mn to 6 mn the unavailability gain is about 30% whereas from 6 mn to 1 mn it almost reaches 80%.

Table 11: Unavailability of FPP per year according to the switching duration $1/\beta_{FPP}$ ($1/\nu_{FPP}$ = 10 mn)

$1/\beta_{FPP}$	A.2.1.0	A.2.1.1	A.2.1.2
10 mn	18 h 39 mn	18 h 02 mn	18 h 34 mn
6 mn	11 h 33 mn	11 h 02 mn	11 h 28 mn
1 mn	**2 h 41 mn**	**2 h 17 mn**	**2 h 36 mn**
1 s	56 mn 46 s	33 s	50 mn 59 s

Table 12: Unavailability of FPP per year according to the restart duration $1/\nu_{FPPP}$ ($1/\beta_{FPP}$ = 1 mn)

$1/\nu_{FPP}$	A.2.1.0	A.2.1.1	A.2.1.2
1 h	6 h 20 mn	5 h 34 mn	6 h 10 mn
10 mn	**2 h 41 mn**	**2 h17 mn**	**2 h 36 mn**
6 mn	2 h 27 mn	2 h 05 mn	2 h 22 mn
1 mn	2 h 10 mn	1 h 50 mn	2 h 05 mn

7. Conclusion

Only part of the results obtained for CAUTRA have been discussed in this paper. The overall results provide significant information about its behavior. They highlight the particular importance of selecting a global fault tolerance and maintenance strategy right from the beginning. For example, the tradeoff which has to be made between the repair duration and hardware fault tolerance procedures has been quantified. It has been shown that only the four architectures using a spare with switching on the spare after the failure of the first computer failure meet the requirement of less than 5 mn for RDP unavailability. Among these, the best one is A.2.1.1 where the two principal replicas run onto the same computer (organization O2) and where the replicas switch back their roles after failure of the principal software replica and restart (reconfiguration R1).

The explicit modeling of the interactions between hardware and software components showed the strong dependency between components. For example, the activation of a temporary hardware fault, may propagate an error to the hosted software components, which in turn may propagate to other software components communicating with them (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, or to switching replicas in case of software replication for fault tolerance. Even if this has already been observed on real-life systems, it has not been modeled explicitly in previous work.

The number of alternatives considered for CAUTRA and the complexity of the models induced by the numerous interactions between the components urged us to follow a modular and systematic approach which is particularly efficient for modeling several alternatives and well-suited for mastering this complexity. For instance, we have shown that the modeling of 11 alternatives requires only the addition of a component net to those of the current architecture and adaptation of six dependency nets among them (thore related to reconfigutration and fault tolerance strategy). Even if

building up the block models and validating the block's GSPNs (designed according to specific rules and to be as generic as possible) is time consuming, it is still worthwhile however since the time saved and the amount of confidence gained in creating and validating the overall model is not commensurable. We believe that this approach can be applied to other complex systems. In our work, we placed the emphasis on permanent and temporary faults due to their important impact for CAUTRA. It is reasonable to assume other interactions that are more important for other applications such as common mode failures or sharing of common disks. There are no major impediments for modeling such interactions using the modular approach presented in this paper, together with the rules defined for constructing the dependency nets.

Acnowledgements

The authors would like to thank Jean Arlat, Alain Costes, Mohamed Kaâniche and Jean-Claude Laprie from LAAS-CNRS for their useful comments when reading earlier versions of this paper. Their comments hepled improving the quality of the presentation.

References

[1] E. Amadio, P. Iaboni, M. Lamanna and P. Mariano, "Implementation of High Availability Mechanisms in the Air Traffic Control SIR-S System", in *24th IEEE Int. Symp. on Fault-Tolerant Computing,* (Austin, Texas, USA), pp. 134-6, 1994.

[2] 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.

[3] A. Avizienis and D. E. Ball, "On the Achievement of a High Dependable and Fault-Tolerant Air Traffic Control System", *IEEE Computer*, 20 (2), pp. 84-90, 1987.

[4] 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.

[5] A. Bobbio and K. Trivedi, "An Aggregation Technique for the Transient Analysis of Stiff Markov Chains", *IEEE Trans. on Computers*, C-35 (9), pp. 803-14, 1986.

[6] M. Borrel, *Interactions between Hardware and Software Components: Characterization, Formalization and Modeling — Application to CAUTRA Dependability*, PhD Dissertation, N°In French, 1995.

[7] C. Chen, H. Asada, Y. Kakuda and T. Kikuno, "Comparison of Hybrid Modular Redundant Multiprocessor Systems with respect to Performabilities", in *23rd IEEE Int. Symp. Fault-Tolerant Computing,* (Toulouse, France), pp. 66-75, 1993.

[8] G. Chiola and S. Donatelli, "GSPN versus SPNs: What is the Actual Role of Immediate transitions?", in *Int. Workshop on Petri Nets and Performance Models,* (Los Alamitos, CA), pp. 20-31, 1991.

[9] 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.

[10] P.-J. Courtois and P. Semal, "Computable Dependable Bounds for large Markov Chains", in *Predictable Dependable Computing Systems* pp. 507-18, Springer, Basic Research Series, 1995.

[11] F. Cristian, B. Dancey and J. Dehn, "Fault-Tolerance in the Advanced Automation System", in *20th IEEE Int. Symp; on Fault-Tolerant Computing,* (Newcastle, UK), pp. 6-17, 1990.

[12] 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.

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

[14] J.-M. Garot and T. Hawker, "Evaluating Proposed Architectures for the FAA's Advanced Automation System", *IEEE Computer*, 20 (2), pp. 33-45, 1987.

[15] A. Goyal, P. Shahabuddin, P. Heidelberger, V. F. Nicola and P. W. Glynn, "A Unified Framework for Simulating Markovian Models of Highly Dependable Systems", *IEEE Trans. on Computers*, 41 (1), pp. 36-51, 1992.

[16] 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.

[17] V. R. Hunt and G. V. Kloster, "The Federal Aviation Administration's Advanced Automation Program", *IEEE Computer*, 20 (2), pp. 14-7, 1987.

[18] J.-C. Laprie, "Dependability Evaluation of Software Systems in Operation", *IEEE Trans. on Software engineering*, SE-10 (6), pp. 701-14, 1984.

[19] J.-C. Laprie (Ed.), *Dependability: Basic Concepts and Terminology,* Dependable Computing and Fault-Tolerant Systems, 5, 265 p. , Springer Verlag, Wien-New York, 1992.

[20] J.-C. Laprie, "On The Temporary Character of Operation-persistent Software Faults", in *4th Int. Symp. Software Reliability Engineering,* (Denver, Colorado), pp. 125, 1993.

[21] J.-C. Laprie, C. Béounes, M. Kaâniche and K. Kanoun, "The Transformation Approach to Modeling and Evaluation of Reliability and Availability Growth of Systems", in *20th IEEE Int. Symp. Fault-Tolerant Computing,* (Newcastle, UK), pp. 364-71, 1990.

[22] J.-C. Laprie and K. Kanoun, "X-ware Reliability and Availability Modeling", *IEEE Trans. on Software Engineering*, SE-18 (2), pp. 130-47, 1992.

[23] J.-C. Laprie and K. Medhaffer-Kanoun, "Dependability Modeling of Safety Systems", in *10th IEEE Int. Symp. on Fault-Tolerant Contuting,* (Kyoto, Japan), pp. 245-50, 1980.

[24] I. Lee and R. K. Iyer, "Faults, Symptoms, and Software Fault Tolerance in Tandem GUARDIAN90 Operating System", in *23rd IEEE Int. Symp. Fault-Tolerant Computing,* (Toulouse, France), pp. 20-9, 1993.

[25] J. F. Meyer, K. H. Muralidhar and W. H. Sanders, "Performability of a Token Network under Transient Fault Conditions", in *19th IEEE Int. Symp. on Fault-Tolerant Computing,* (Chicago, Illinois, USA), pp. 175-82, 1989.

[26] J. F. Meyer and W. H. Sanders, "Specification and Construction of Performability Models", in *Int. Workshop on Performability Modeling of Computer and Communication Systems,* (Mont Saint Michel, France), pp. 1-32, 1993.

[27] J. K. Muppala, A. Sathaye, R. Howe, C and K. S. Trivedi, "Dependability Modeling of a Heterogeneous VAX-cluster System Using Stochastic Reward Nets", in *Hardware and Software Fault Tolerance in Parallel Computing Systems* (D. R. Avresky, Ed.), pp. 33-59, 1992.

[28] P. I. Pignal, "An Analysis of Hardware and Software Availability Exemplified on the IBM-3725 Communication Controller", *IBM Journal of Research and Development*, 32 (2), pp. 268-78, 1988.

[29] K. H. Prodromides and W. H. Sanders, "Performability Evaluation of CSMA/CD & CSMA/DCR Protocols Under Transient Faul Conditions", *IEEE Trans. on Reliability*, 42 (1), pp. 116-27, 1993.

[30] W. Sanders and J. Meyer, "Reduced Base Model Construction Methods for Stochastic Activity Networks", *IEEE Trans. on Selected Areas in Communications*, 9 (1), pp. 25-36, 1991.

[31] W. H. Sanders and W. D. Obal II, "Dependability Evaluation Using UtraSAN", in *23th Int. Symp. Fault-Tolerant Computing (FTCS-23),* (Toulouse, France), pp. 774-9, 1993.

[32] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design,* Digital Press, 1992.

[33] G. E. Stark, "Dependability Evaluation of Integrated Hardware/Software Systems", *IEEE Trans. on Reliability*, R-36 (4), pp. 440-4, 1987.

[34] L. A. Tomek and K. S. Trivedi, "Analysis Using Stochastic Reward Nets", in *Software Fault Tolerance* (M. Lyu, Ed.), pp. 138-65, J. Wiley, 1995.