Support de Cours

UNIVERSITE
DE BATNA 2

DEPARTMENT
D'INFORMATIQUE

Sûreté de fonctionnement

Dr. Chafik ARAR
chafik.arar@univ-batna2.dz

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Sûreté de fonctionnement

Sommaire

1.1 Introduction

La nécessité de la sûreté de fonctionnement n'est apparue que au cours du XXème siècle, suite à la révolution industrielle. Le terme "dependability" est apparu dans une publicité sur des moteurs Dodge Brothers dans les années 30. La sûreté de fonctionnement peut se définir par son objectif principale qui vise d'atteindre l'idéal de la conception des systèmes : zéro accident, zéro arrêt, zéro défaut (et même zéro maintenance). Pour pouvoir y arriver, il faudrait tester toutes les utilisations possibles d'un produit pendant une grande période, ce qui est impensable et impossible dans le contexte industriel.

La sûreté de fonctionnement, c'est la science qui propose des moyens et des techniques qui permettent la réalisation des système sûrs et aussi fiable que possible dans des délais et avec des coûts raisonnables. L'importance et la vitalité de la sûreté de fonctionnement vient du fait que les conséquences que pourrait entrainer une faute dans un système réactif critique sont catastrophiques (perte d'argent, de temps, ou pire de vies humaines). Les moyens qui permettent la conception de systèmes sûrs de fonctionnement sont les techniques de la sûreté de fonctionnement [1] [2] [3] [4] [5] [6].

Dans la littérature, on trouve que plusieurs méthodes utilisent la tolérance aux fautes comme le moyen le plus important parmi plusieurs pour la conception des systèmes sûrs de fonctionnement, que nous appelons dans la suite "systèmes tolérants aux fautes". La tolérance aux fautes permet à un système de continuer à fonctionner et à délivrer un service conforme à sa spécification en présence de fautes. Ils existe d'autres méthodes utilisant une mesure de probabilité, appelée fiabilité, pour concevoir aussi des systèmes sûrs de fonctionnement, que nous appelons "systèmes fiables". A la différence de la la tolérance aux fautes, la fiabilité permet d'évaluer aléatoirement le bon fonctionnement d'un système.

Ce chapitre reprend et actualise les points principaux liés à la sûreté de fonctionnement. Nous commençons tout d'abord par la définition de la sûreté de fonctionnement, ensuite on met l'accent sur les concepts de base de cette dernière. Les spécificités relatives à la tolérance aux fautes et ces techniques, plus spécifiquement la redondance logicielle sont aussi exposés et enfin nous identifions quelques défis pour la mise en œuvre de la tolérance aux fautes.

1.2 Qu'est-ce que la sûreté de fonctionnement ?

La sûreté de fonctionnement ou science des défaillances comme elle est souvent appelée, inclut la connaissance des défaillances, leur évaluation, leur prévision, leurs mesures et leur maîtrise. Dans la plupart du temps l'étude de la sûreté de fonctionnement, relève d'un domaine transverse et souvent multidisciplinaire qui nécessite une connaissance globale du système comme les architectures fonctionnelles et matérielles, les conditions d'utilisation, les risques extérieurs. Beaucoup d'avancées dans ce domaine sont le résultat d'un feed-back d'expérience et des rapports d'analyse d'accidents.

Définition 1 : "La sûreté de fonctionnement d'un système informatique est la propriété qui permet de placer une confiance justifiée dans le service qu'il délivre."[7]

Cette définition met en évidence la notion de "justification de la confiance", qui est une dépendance acceptée (explicitement ou implicitement). La dépendance d'un système par rapport à un autre système est l'influence, réelle ou potentielle, de la sûreté de fonctionnement de ce dernier sur la sûreté de fonctionnement du système considéré.

Définition 2 (SdF) : "La sûreté de fonctionnement (Dependability) consiste à évaluer les risques potentiels, prévoir l'occurrence des défaillances et tenter de minimiser les conséquences des situations catastrophiques lorsqu'elles se présentent."

Il existe beaucoup de définitions et de standards de la sûreté de fonctionnement qui peuvent varier selon les domaines d'application. Le Technical Committee 56 Dependability de l'International Electrotechnical Commission (IEC) développe et maintient des standards internationaux reconnus dans le domaine de la sûreté

de fonctionnement. Ces standards fournissent les méthodes et les outils d'analyse, d'évaluation, de gestion des équipements, services et systèmes tout au long du cycle de développement [8] [9] [10].

1.3 Concepts de base

Dans cette section nous introduisons les concepts de base de la sûreté de fonctionnement [8] [10]. Nous donnons des définitions précises caractérisant les principes qui influencent la sûreté de fonctionnement des systèmes informatiques. Les définitions données sont de caractère générales pour couvrir tout le spectre des systèmes informatiques .

1.3.1 Fonction et service d'un système

Un *système* est une entité qui est en constante interaction avec d'autres systèmes, qui constituent son environnement. La frontière du système est la limite commune entre le système et son environnement.

Un système est dit cohérent si :

— La panne de tous les composants entraîne la panne du système,

— Le fonctionnement de tous les composants entraîne le fonctionnement du système,

— Lorsque le système est en panne, aucune défaillance supplémentaire ne rétablit le fonctionnement du système,

— Lorsque le système est en fonctionnement, aucune réparation ne conduit à la panne du système.

La *fonction* d'un système est ce à quoi il est destiné. Elle est décrite par la spécification fonctionnelle, qui inclut les performances attendues du système.

Le *service* délivré par un système est son comportement tel que perçu par ses utilisateurs. Le comportement est ce que le système fait pour accomplir sa fonction, il est représenté par une séquence d'états externes. Un utilisateur est un autre système, éventuellement humain, qui interagit avec le système considéré. Un service correct est délivré par un système lorsqu'il accomplit sa fonction.

1.3.2 Attributs, Entraves et moyens

La sûreté de fonctionnement manipule principalement trois concepts [10] [8] [9] :

— Les Attributs : par quoi la sûreté de fonctionnement est évaluée ;

— Les Entraves : par quoi la sûreté de fonctionnement est affectée ;

— Les Moyens : par quoi la sûreté de fonctionnement est améliorée.

La Figure 1.1 résume les principales notions de la sûreté de fonctionnement.

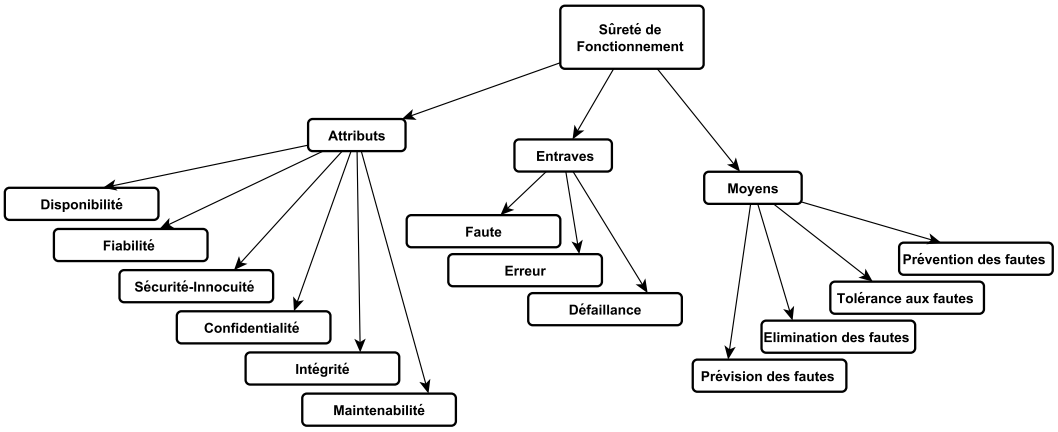

FIGURE 1.1 – Arbre de la sûreté de fonctionnement

1.3.2.1 Les attributs de sûreté de fonctionnement

Selon la, ou les applications auxquelles le système est destiné, l'accent peut être mis sur différentes facettes de la sûreté de fonctionnement, ce qui revient à dire que la sûreté de fonctionnement à des propriétés différentes, mais complémentaires, ces propriétés permettent de définir ses attributs :

— Disponibilité(Availability) : c'est le fait d'être prêt pour l'utilisation, c'est l'aptitude d'une entité à être en état d'accomplir une fonction requise dans des conditions bien déterminées, à un instant donnée ou pendant un intervalle de temps donné, en supposant que la fourniture des moyens extérieurs nécessaires soit assurée.

— Fiabilité (Reliability) : qui est la continuité du service, c'est l'aptitude d'un dispositif à accomplir une fonction requise dans des conditions données pendant une durée donnée.

— Sécurité-innocuité (Safety) : qui est l'absence de conséquences catastrophiques pour l'environnement, c'est l'aptitude d'une entité à éviter de conduire, dans des conditions données à des évènements critiques ou catastrophiques.

— Confidentialité : qui est l'absence de divulgations non autorisées de l'information.

— Intégrité : c'est l'absence d'altérations inappropriées de l'information.

— Maintenabilité (Maintainability) : permettre les réparations et les évolutions, c'est l'aptitude d'une entité à être maintenue ou rétablie, sur un intervalle de temps donnée dans un état dans lequel elle peut accomplir une fonction requise.

C'est l'application à laquelle est destiné le système informatique, qui définit l'importance des attributs de la sûreté de fonctionnement : disponibilité, intégrité, et

maintenabilité sont généralement requises (à des degrés variables), alors que fiabilité, sécurité-innocuité, confidentialité peuvent ou non êtres requises. La mesure dans laquelle un système possède les attributs de la sûreté de fonctionnement doit être considérée de façon relative, probabiliste, et non de façon absolue, déterministe, et ça parceque un système n'est jamais totalement disponible, fiable ou sûr du fait de l'inévitable présence ou occurrence de fautes.

En plus des attributs déjà définis, et qui sont dans la plupart des références qualifiés d'attributs primaires, d'autres attributs peuvent être définis en tant qu'attributs secondaires, dont le rôle est l'affinement ou la spécialisation des attributs primaires. Un exemple d'attribut secondaire spécialisé est :

— Robustesse : la sûreté de fonctionnement par rapport aux fautes externes.
— Responsabilité : la disponibilité et l'intégrité de l'identité de la personne qui a effectuée une opération.
— Authenticité : l'intégrité du contenu et de l'origine d'un message.
— Non-réfutabilité : la disponibilité et l'intégrité de l'identité de l'émetteur d'un message, ou du destinataire.
— Testabilité : le degré d'un composant ou d'un système à fournir des informations sur son état et ses performances.
— Diagnosticabilité : la capacité d'un système à exhiber des symptômes pour des situations d'erreur.
— Survivabilité : la capacité d'un système à continuer sa mission après perturbation humaine ou environnementale.

L'importance donnée à chacun des attributs de la sûreté de fonctionnement influence directement sur le seuil des techniques à mettre en œuvre pour que le système résultant soit sûr de fonctionnement. Ceci est un problème délicat, surtout que certains attributs sont antagonistes (par exemple, disponibilité et sécurité-innocuité, disponibilité et sécurité-immunité), d'où la nécessité de négocier des compromis.

1.3.2.2 Les entraves à la sûreté de fonctionnement

Les entraves qui peuvent affecter le fonctionnement d'un système et dégrader la sûreté de fonctionnement sont classées en trois grandes classes [10], [9] : les fautes, les erreurs et les défaillances, qui s'enchaînent comme illustrées dans la Figure 1.2.

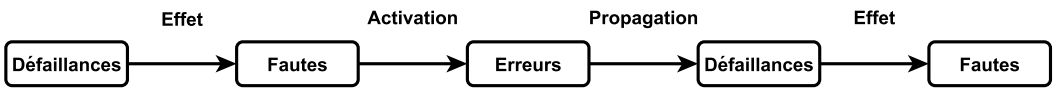

FIGURE 1.2 – Entraves de la sûreté de fonctionnement

Les flèches de la chaîne expriment la relation de causabilité entre fautes, erreurs et défaillances. Elles doivent être interprétées de façon générique : par propagation, plusieurs erreurs sont généralement générées avant qu'une défaillance ne survienne.

Faute (Fault) : Une faute dans un système [11] représente un défaut d'un composant matériel ou logiciel de ce système, elle est générée d'une manière intentionnelle ou accidentelle. Durant l'exécution du système, la faute reste inactive jusqu'à ce qu'un évènement intentionnel ou accidentel provoque son activation.

Quand une faute est active elle produit automatiquement une erreur. Une faute active est soit une faute interne qui était là mais inactive, et c'est le processus de traitement qui l'a activée, soit une faute externe qui a profité d'une vulnérabilité. Les fautes susceptibles d'affecter un système (de nature et de source extrêmement diverses), peuvent être classées selon sept points de vue, permettant de définir les classes de fautes élémentaires, comme indiqué sur le Tableau 1.1.

On combinant ces sept classes de fautes élémentaires, on se trouve avec des classes de fautes combinées, on peut noter que le reclassement de toutes les combinaisons des classes de fautes élémentaires est possible (et il y aurait exactement 192 classes de fautes combinées) ; la plus part de ces combinaisons n'étant pas pertinentes, donc les classes de fautes combinées peuvent être regroupées en trois grandes classes non exclusives :

— Fautes de développement.

— Fautes physiques ou fautes du matériel.

— Fautes d'interaction ou externes.

Reproduire l'activation d'une faute est par définition la possibilité d'identifier les conditions et l'environnement de l'activation d'une faute (source d'une ou plusieurs erreurs). La Figure 1.3 définit les fautes solides ou furtives selon que leurs conditions d'activation sont reproductibles ou non ; elle définit aussi la notion de faute intermittente, en raison de la similitude de manifestation des fautes furtives et des fautes temporaires.

Erreur (Defect) : La présence d'un état interne erroné pendent l'activation d'un système conduit dans des circonstances particulières à une erreur, c'est-à-dire à un résultat incorrect ou imprécis. Donc l'erreur est le résultat d'une faute et la cause d'une défaillance. Une erreur peut être latente tant qu'elle n'a pas été reconnue en tant que telle ou détectée par un algorithme ou un mécanisme de détection qui permet de la reconnaître comme telle. Une erreur peut disparaître avant d'être détectée. Une erreur peut aussi crée des nouvelles erreurs par propagation [9].

Faute	Création ou occurrence	Fautes de développement : (imperfections de développement ou de maintenance).
		Fautes opérationnelles : au moment de l'exploitation du système.
	Frontière du système	Fautes internes : activées par les traitements.
		Fautes externes : suite aux interactions du système avec son environnement extérieur (physique ou humain).
	Cause	Fautes naturelles : sans intervention humaine.
		Fautes dues à l'homme : suite aux actions et imperfections humaines.
	Dimension	Fautes matérielles : elles affectent le matériel.
		Faute logicielle : elles affectent le logiciel (programmes ou données).
	Intention	Fautes malveillantes : elles sont humaines et faites exprès.
		Fautes non malveillantes : elles sont sans intentions malveillantes.
	Capacité	Fautes accidentelles : elles sont créées de manière fortuite.
		Fautes délibérées : elles sont créées délibérément avec une décision.
		Fautes d'incompétence : elles sont le résultat d'un manque de compétence professionnelle.
	Persistance	Fautes permanentes : ce type de fautes est continue dans le temps (jusqu'a réparation).
		Fautes temporaires : ces fautes sont présentées pour une durée limitée.

TABLE 1.1 – Classes de fautes

Défaillance : Quand un service correct est délivré par un système, on dit que ce système a accompli sa fonction. Une défaillance du service ou simplement défaillance, est un évènement qui survient lorsque le service délivré n'est plus correct (parce qu'il ne respecte pas la spécification fonctionnelle ou parceque la spécification fonctionnelle ne décrivait pas de manière adéquate la fonction du système).

Une défaillance est par définition le passage du service de l'état correct à un état incorrect, c'est le résultat d'une erreur qui à changé le comportement du système et provoque le non respect de sa spécification. Le résultat d'une défaillance (d'un composant) est une faute non seulement pour le système qui le contient mais aussi

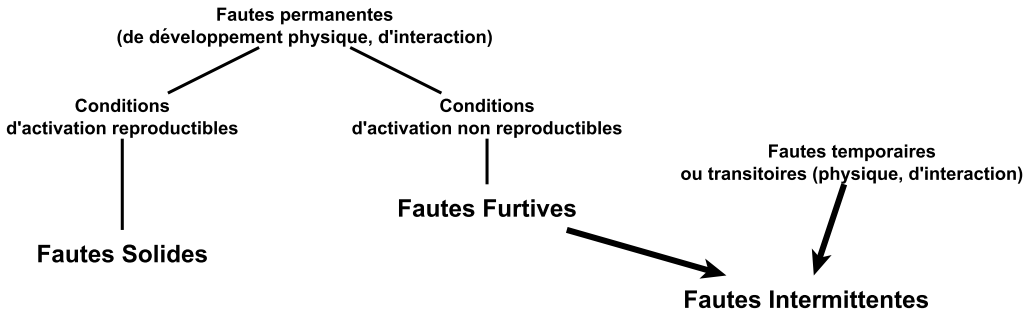

FIGURE 1.3 – Fautes solides, furtives et intermittentes

pour tous les les composants qui interagissent avec lui. La délivrance d'un service incorrect est une panne du service, et la restauration du service est par définition le passage de l'état de service incorrect à l'état d'un service correct [11].

Mode de défaillance (Failure mode) : Un système ne défaille généralement pas toujours de la même façon, ce qui conduit à la notion de mode de défaillance. Un mode de défaillance spécifie l'effet par lequel une défaillance est observée. Un mode de défaillance peut être caractérisé selon quatre points de vue, comme indiqué dans le Tableau 1.2, ce qui défini la classification des défaillances du service.

On notes les trois commentaires suivants sur les modes de défaillance :

— Une défaillance non signalée est le résultat direct d'une défaillance des mécanismes de détection d'une délivrance d'un service incorrect ; un autre cas peut se produire lorsque ces mécanismes détectent et signalent une défaillance inexistante (donc émettent une fausse alarme).

— Un système dont toutes les défaillances sont acceptables avec des défaillances par arrêt est un système à arrêt sur défaillance ; un système dont toutes les défaillances sont acceptables mais avec des défaillances bénignes est un système sûr en présence de défaillance.

— Le déni de service est l'exemple des malveillances qui peuvent affecter la disponibilité du service.

La gravité des défaillances instaure un classement des conséquences des défaillances sur l'environnement du système. Lorsque la fonction du système comporte un ensemble de fonctions élémentaires, la défaillance d'un ou de plusieurs services remplissant ces fonctions peut laisser le système dans un mode dégradé, qui offre encore un sous-ensemble de services à l'utilisateur. Plusieurs de ces modes dégradés peuvent être identifiés, tels que service ralenti, service restreint, service d'urgence, etc. Dans ce cas, le système est dit avoir souffert des défaillances partielles.

Le service délivré étant une séquence d'états externes, une défaillance du service signifie qu'au moins un état externe dévie du service correct. La déviation est une

Défaillance	Domaine	Défaillance en valeur : l'information délivrée ne cordonne pas avec la fonction du système.
		Défaillance temporelles : le moment de la délivrance de l'information ne permet pas l'accomplissement de la fonction du système.
	Détectabilité	Défaillance signalées : la possibilité de détecter et de signaler que le service délivré est incorrect.
		Défaillance non signalées : La délivrance d'un service incorrect n'est pas détectée.
	Cohérence	Défaillances cohérentes : tous les utilisateurs aperçoivent le service incorrect identiquement.
		Défaillances incohérentes ou défaillances byzantines : certains ou tous les utilisateurs n'aperçoivent pas le service incorrect de la même manière.
	Conséquences	Défaillances bénignes : le dommage causé au système ou à son environnement est négligeable et sans présenter de risque pour l'homme.
		Défaillance critique (hasardeuse) : elle entraîne la perte d'une ou des fonctions importantes du système et cause des dommages importants au système.
		Défaillances catastrophiques

Table 1.2 – Modes de défaillance

erreur. La cause adjugée ou supposée d'une erreur est une faute. Les fautes peuvent être internes ou externes au système. La présence antérieure d'une vulnérabilité, c-à-d d'une faute interne qui permet à une faute externe de causer des dommages au système, est nécessaire pour qu'une faute externe entraîne une erreur, et, éventuellement, une défaillance.

Généralement, une faute cause d'abord une erreur dans l'état interne d'un composant, l'état externe du système n'étant pas immédiatement affecté. Il s'ensuit la définition d'une erreur : partie de l'état total du système qui est susceptible d'entraîner sa défaillance, qui survient lorsque l'erreur affecte le service délivré.

1.3.2.3 Les moyens de la sûreté de fonctionnement

Toute solution développée dans le cadre d'un système sûr de fonctionnement doit utilisée une ou une combinaison d'un ensemble de méthodes qui sont des solutions éprouvées pour casser les enchaînements Faute $\longmapsto$ Erreur $\longmapsto$ Défaillance et donc

améliorer la fiabilité du système. Ces méthodes sont les moyens de la sûreté de fonctionnement [9] [12] et peuvent être classées en :

— Méthode de prévention des fautes : le principe de ces méthodes est d'empêcher l'occurrence ou l'introduction de fautes qui auraient pu être introduites pendant le développement du système. Cela peut être réaliser par des bonnes techniques d'implantation et de développement.
— Tolérance aux fautes : elle consiste à mettre en place des mécanismes qui maintiennent le service fourni par le système, même en présence de fautes et on peut même accepter un fonctionnement dégradé. les outils de base de la tolérance aux fautes sont les mécanismes de redondance, l'idée est de réaliser la même fonction par des moyens différents.

On distingue plusieurs types de redondance :

1. Redondance homogène : on réplique plusieurs composants identiques.
2. Redondance avec dissemblance : les sous-systèmes réalisent les mêmes fonctions mais sont différents.
3. Redondance froide : les composants sont activés quand ceux déjà actifs tombent en panne.
4. Redondance chaude : les composants tournent en parallèle avec une politique de prise en main.
5. Redondance tiède : les composants sont idle (non actif, tourner au ralenti) avant de prendre la main.

A la base de toutes les solutions existantes, la récupération de plusieurs valeurs calculées par redondance est de déterminer laquelle est la plus proche de la réalité. c'est l'idée de base des mécanismes comme les comparateurs ou les voteurs.

1. Élimination des fautes : elle consiste à réduire la présence (nombre, sévérité) des fautes, l'élimination de faute peut être divisée en 2 catégories :
   — Elimination pendant le développement.
   — Elimination pendant l'utilisation.
2. Prévision des fautes : elle consiste à estimer la présence, le taux futur, et les possibles conséquences des fautes (leur impact sur le système).

La Figure 1.4 suivante illustre les différents groupements des moyens pour la sûreté de fonctionnement,

1.4 Tolérance aux fautes

Un système dont les programmes peuvent être exécutés correctement même en présence de fautes est un système tolérant aux fautes, dans la littérature plusieurs travaux traitent le concept de la tolérance aux fautes [13] [14].

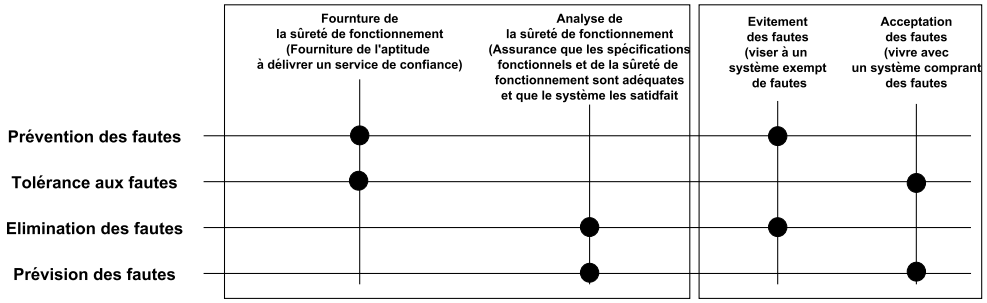

FIGURE 1.4 – Groupements des moyens pour la sûreté de fonctionnement

1.4.1 Techniques de la tolérance aux fautes

La tolérance aux fautes à comme objectif principal d'éviter les défaillances. Sa mise en œuvre est réaliser par la détection des erreurs et le rétablissement du système [9]. La Figure 1.5 liste les techniques de tolérance aux fautes. Habituellement le traitement de fautes est complété par des opérations de maintenance corrective, afin d'éliminer les composants passives.

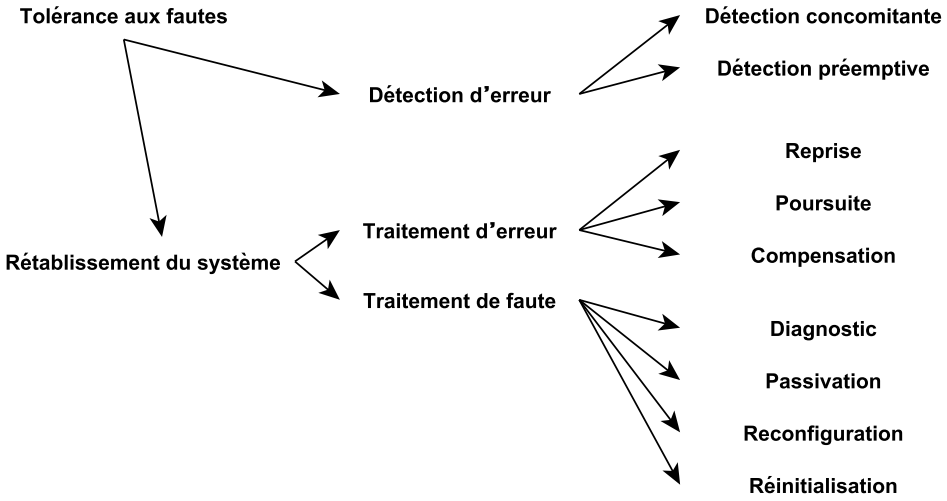

FIGURE 1.5 – Technique de tolérance aux fautes

La reprise et poursuite sont deux états qui sont invoquées à la demande, après qu'une ou plusieurs erreurs aient été détectées, alors que la compensation peut être appliquée à la demande ou systématiquement, indépendamment de la présence ou de l'absence d'erreur. Le traitement d'erreurs est à la demande, suivi du traitement

de faute constituent le rétablissement du système, d'où le qualificatif de la stratégie de tolérance aux fautes correspondante : détection et rétablissement.

Le masquage de fautes résulte de l'application systématique de la compensation. Un tel masquage peut entraîner une diminution non perçue des redondances disponibles, sa mise en œuvre pratique comporte généralement une détection d'erreur, conduisant au masquage et détection.

Il est à noter que :

— Reprise et poursuite ne sont pas exclusives : une reprise peut d'abord être tentée ; si l'erreur persiste, une poursuite peut alors être entreprise ;
— Les fautes intermittentes ne nécessitent ni passivation, ni reconfiguration. Le traitement d'erreur est le moyen le plus utilisé pour identifier si une faute est intermittente ou non (la récurrence d'une erreur indique que la faute n'est pas intermittente) ou par diagnostic de faute dans le cas de la poursuite.

1.4.1.1 Détection d'erreur

La redondance permet la détection d'erreurs, elle peut prendre plusieurs formes : redondance au niveau information ou composant, redondance temporelle ou algorithmique. La forme la plus sophistiquée de la détection d'erreur consiste à construire des composants auto testables en adjoignant au composant purement fonctionnel des éléments de contrôle permettant de vérifier que certaines propriétés entre les entrées et les sorties du composant sont satisfaites [14] [15].

Les formes de détection d'erreur les plus couramment utilisées sont les suivantes :

— Codes détecteurs d'erreur ;
— Doublement et comparaison ;
— Contrôles temporels et d'exécution ;
— Contrôles de vraisemblance ;
— Contrôles de données structurées.

1.4.1.2 Rétablissement du système

Trois formes de rétablissement du système ont été identifiées, cette identification se fait par rapport au moyen utilisé pour reconstruire un état correct. Les trois formes sont : la reprise, la poursuite et la compensation d'erreur.

Reprise : La reprise est la technique la plus utilisée pour le rétablissement du service d'un système. Elle consiste en la sauvegarde périodique de l'état du système de façon à pouvoir, après avoir détecté une erreur, ramener le système vers un état dit point de reprise.

La sauvegarde périodique de l'état du système doit s'effectuer au moyen d'un mécanisme de mémorisation, ou support stable qui protège les données contre les effets des fautes [16] [17].

La sauvegarde opère par des mécanismes matériels ou logiciels permettant de sauvegarder automatiquement les données modifiées entre deux points de reprise. Si la couverture de détection n'est pas totale, les points de reprise peuvent être eux aussi sujet d'une erreur avant même qu'elle ne soit détectée. Dans ce cas, la reprise ne pourra être efficace que s'il est possible de restituer un état exempt d'erreur, ce qui implique qu'il doit exister plusieurs points de reprise successifs ou que la structure de l'application permette de conserver des points de reprise emboîtés.

Les inconvénients des techniques de reprise sont : premièrement, elles sont généralement incompatibles avec les applications ayant des contraintes temps réel strictes. Deuxièmement, la taille des points de reprise et le surcoût temporel nécessaire à leur établissement imposent souvent des contraintes structurelles qui doivent être prises en compte pendent le développement de l'application, avec un support spécifique du système d'exploitation. Cela interdit généralement l'usage de systèmes d'exploitation et de tout logiciel qui n'aurait pas été développé spécialement pour l'architecture en question.

Poursuite : La poursuite se définie comme une approche alternative ou complémentaire à la reprise, après avoir détecté une erreur, et après avoir éventuellement tenté une reprise, la poursuite consiste en la recherche d'un nouvel état acceptable pour le système à partir duquel celui-ci pourra fonctionner même en mode dégradé. L'une des plus simple réalisation de la poursuite consiste à réinitialiser le système et à acquérir un nouveau contexte à partir de l'environnement lui même (comme par exemple la relecture des capteurs). Une autre façon de faire est le traitements d'exceptions, en se basant sur des primitives offertes par certains langages de programmation [16].

Compensation : Avec la compensation d'erreur, le système doit comporter suffisamment de redondance, pour permettre sa transformation en un état exempt d'erreur, en dépit des erreurs qui pourraient l'affecter [9]. La compensation ne nécessite ni la ré-exécution d'une partie de l'application (comme dans le cas de la reprise), ni l'exécution d'une procédure dédiée (comme dans le cas de la poursuite), pour permettre la continuation du service. Ce type de recouvrement est donc relativement transparent par rapport à l'application elle même, ce qui permet l'utilisation de systèmes d'exploitation et de progiciels standard (il n'est pas nécessaire de structurer l'application en vue d'un éventuel traitement d'erreur).

La compensation d'erreur peut être par détection d'erreur (détection et compensation), ou systématique (dans le cas du masquage). Enfin, elle peut être aussi

fournie par les codes correcteurs d'erreur, spécialement pour la transmission et ou le stockage de l'information.

1.4.2 Tolérance aux fautes logicielles et matérielles

Un système peut défaillir à cause d'une faute logicielle ou matérielle.

1.4.2.1 Tolérance aux fautes logicielles

Le problème de conception et développement (écriture) de logiciels est très difficile [18]. Il y a beaucoup de difficultés essentielles ou accidentelles pour produire un logiciel correct et sans erreurs. A l'origine de difficultés essentielles, le défi d'essayer de comprendre l'application et l'environnement de fonctionnement qu'ils sont souvent complexes. Pour le cas des difficultés accidentelles, l'origine est qu'on ne peut pas produire un bon logiciel du moment ou les programmeurs peuvent faire des erreurs dans les programmes. Un exemple intéressant de faute logicielle est l'explosion de la fusée Ariane 5 en juin 1996 à cause des erreurs de conception du logiciel.

Il existe deux techniques de base pour la tolérance aux fautes logicielles :

— **Uni-version du logiciel :** l'objectif principal est de tolérer les fautes logicielles en utilisant une seule version du logiciel, ou d'un de ses composants. Pour tolérer la faute de chaque uni-version du logiciel, le concepteur doit modifier cette version en lui ajoutant des mécanismes de détection et de traitement d'erreurs [1] [2] [3] [4]. Des exemlpes des approches qui utilisant cette technique : le traitement d'exceptions, la détection d'erreur, le point de reprise (check-point and restart),

— **Multi-version du logiciel :** cette technique est basée sur le principe de la redondance logicielle, où chaque composant logiciel est répliqué pas identiquement mais en plusieurs versions programmées différemment. Le premier avantage par rapport à l'approche uni-version est que ces versions logicielles peuvent être exécutées en séquence ou en parallèle pour tolérer les fautes de certaines versions. Un autre avantage est que ces différentes versions logicielles peuvent être, développés par différents développeurs sur des outils différents. Parmi les approches utilisant cette technique, on trouve : N-version de programmation, et c'est la cas des commandes de vol des avions Airbus et Boeing.

1.4.2.2 Tolérance aux fautes matérielles

La tolérance aux fautes matérielles est le domaine le plus développé de la tolérance aux fautes générale. Beaucoup de techniques ont été développées et utilisées. On peut tolérer les fautes matérielles soit par logiciel soit par matériel. La solution matérielle consiste à répliquer des composants matériels, et même si le coût

des processeurs ne cesse de décroître, cette solution a un notre inconvénient qu'est l'augmentation de la consommation de l'énergie, c'est pourquoi on opte généralemnt pour les solutions basées sur la redondance logicielle. Dans la suite du manuscrit, on désigne par faute une faute matérielle. Nous ne ferons pas un état de l'art sur la tolérance aux fautes logicielles qui est un domaine de recherche en soi. On fait donc l'hypothèse que le logiciel est sans fautes et donc dans la suite on ne considèrera que la tolérance aux fautes matérielles.

1.4.3 Techniques logicielles de tolérance aux fautes matérielles

1.4.3.1 Reprise locale

Minimiser l'impact négatif de la défaillance temporaire d'un processus par un redémarrage rapide, est la forme la plus simple de la tolérance aux fautes dans un système réparti. Cela peut être facile à mettre en œuvre si ce processus ne sauvegarde aucun état entre deux requêtes. Pas d'état interne implique la non nécessité de la restauration des données lors d'une reprise. Cela implique aussi qu'aucune information n'est conservée [9]. C'est cette stratégie qui permet, par exemple, la conception du système de fichier réseau de Sun (NFS). Si au contraire un processus serveur comporte un état interne, il faut que celui-ci soit sauvegardé sur une mémoire stable en tant que point de reprise. Une reprise locale est effectuer par un processus lorsque il n'a pas interagi avec d'autres processus depuis le dernier point de reprise, ou si d'éventuelles interactions peuvent être rejouées. Si cela n'est pas le cas, une reprise répartie de plusieurs processus est nécessaire.

1.4.3.2 Reprise répartie

Une reprise répartie est faite si une communication interne provoque à la suite d'une reprise d'un processeur, la reprise de d'autres processeurs. Les processus doivent reprendre leurs exécutions depuis un ensemble de points de reprise qui constituent un état global cohérent.

Un effet domino peut survenir lorsque le seul état global cohérent antérieur est l'état initial du système. La définition d'un état global cohérent dépend du positionnement du protocole de reprise par rapport au protocole assurant la fiabilité des communications (Figure 1.6) [19].

Lorsque le protocole de reprise est situé au-dessus d'un protocole de communication fiable (Figure 1.6(a)), un ensemble de points de reprise ne constituent un état global cohérent que si la « ligne de reprise » qui représente cet ensemble n'est traversée par aucun message (A et B sur la Figure1.6(b)). Dans le cas contraire (Figure 1.6 (c)), une ligne de reprise traversée éventuellement par des messages de gauche à droite (C sur la Figure 1.6 (b)) représente aussi un état global cohérent, car ces messages « émis » mais pas encore reçus, seront réémis plus tard par le protocole

de communication fiable (dont l'état fait parti de l'état sauvegardé par les points de reprise).

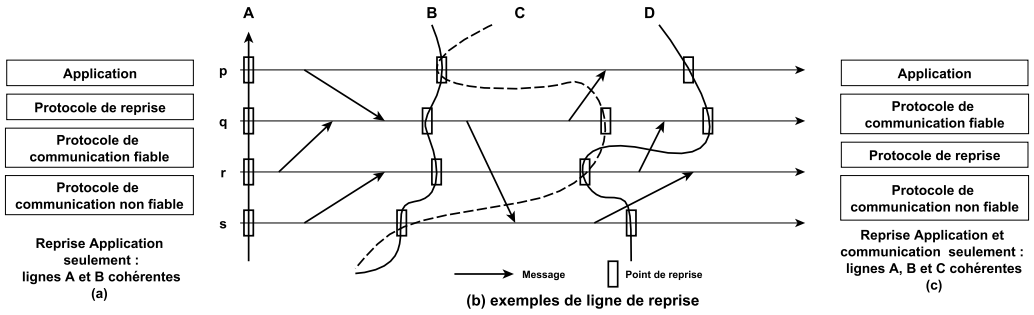

FIGURE 1.6 – Processus Communicant Reprise

La ligne D de la Figure 1.6(b) ne correspond à aucun état global cohérent, car elle est traversée par un message de droite à gauche, ce qui correspondrait à un message reçu mais pas encore émis.

On distingue généralement trois approches pour la reprise répartie :

— La création indépendante des points de reprise : chaque processus est autorisé à créer des points de reprise au moment opportun pour lui ;

— La création coordonnée des points de reprise permet de créer uniquement des lignes de reprise cohérentes ;

— La création de points de reprise induite par la communication est un compromis entre les deux approches précédentes.

Les techniques de reprise considérées jusqu'à maintenant peuvent êtres utilisées pour tolérer des fautes permanentes si les points de reprise sont enregistrés sur une mémoire stable accessible depuis le réseau de communication. En effet, cela autorise la création de nouveaux processus sur des processeurs non défaillants qui peuvent être initialisés à partir de ces points de reprise.

1.4.3.3 Données répliquées

Du point de vue des données, l'utilisation de la redondance permet d'assurer la disponibilité de données, puisque une donnée dupliquée est accessible même si certaines copies résident sur des nœuds défaillants ou inaccessibles. D'autre part, il est logiquement plus rapide de lire une copie proche qu'une copie distante. Cependant, garder la cohérence des données par des opérations d'écriture sur des données dupliquées peut ralentir le traitement, car elle impliquent potentiellement toutes les copies.

Les protocoles de gestion de données dupliquées sont qualifiés d'optimistes ou de pessimistes selon s'ils négligent ou non l'accès conflictuels en cas de partitionnement des copies [9].

Les protocoles pessimistes permettent à tout utilisateur de perçoir la donnée comme si elle n'existait qu'en une seule copie) en forçant l'exclusion mutuelle entre les opérations d'écriture et de lecture. Le protocole le plus simple s'appelle « lire une, écrire toutes » : un processus utilisateur peut lire n'importe quelle copie, mais il doit effectuer toute écriture sur l'ensemble des copies. Cette technique permet de fournir une excellente performance en lecture, mais en contre partie elle fournit une mauvaise performance en écriture, et même par fois les écritures sont bloquées si une seule copie de la donnée n'est pas accessible.

Les protocoles optimistes améliorent la disponibilité des données en sacrifient leurs cohérence. Avec ces protocoles l'écriture sur les copies présente dans des composants distincts est autorisée. Par exemple, le protocole à copies disponibles est une variante optimiste du protocole « lire une, écrire toutes » par laquelle seules les copies accessibles sont modifiées lors d'une opération d'écriture. Mais tout conflit résultant d'écritures effectuées dans des composants différents doit être détecté et résolu. cette résolution de conflit ne peut être automatique que dans des cas très particuliers parceque elle dépend directement de la sémantique des données.

1.4.3.4 Processus répliqués (redondance logicielle)

Les méthodes de tolérance aux fautes matérielles diffèrent selon l'origine et le type de fautes pris en compte [9]. Un service tolérant aux fautes peut être mis en œuvre en coordonnant un ensemble de processus répliqués sur deux ou plusieurs processeurs.

Avec la redondance logicielle, la nécessité d'avoir une architecture distribuée est obligatoire, pour pouvoir attribuer les répliques aux différents processeurs. C'est la technique de tolérance aux fautes matérielles la plus adaptée aux systèmes embarqués car dans ces systèmes le nombre de composants matériels ne peut pas être facilement augmenté [20] [20] [21]. L'ensemble des processus (tâches) doit être géré de façon à présenter le service correct comme étant fournit par un seul processus, malgré la défaillance d'un sous ensemble des membres de l'ensemble.

On distingue généralement trois classes d'algorithmes d'ordonnancement temps réel et tolérants aux fautes (voir chapiter **??**) :

1. Les algorithmes basés sur la redondance active,
2. Les algorithmes basés sur la redondance passive et
3. Les algorithmes basés sur la redondance hybride.

Ces algorithmes s'appliquent à deux types de composants matériels : les processeurs et les médias de communication.

Redondance passive : Avec la Redondance passive, une seule copie dite primaire traite tous les messages reçus, met à jour son état interne et effectue l'envoi de messages de sortie. La copie primaire met à jour une copie de son état interne, pour créer un point de reprise sur une mémoire stable accessible par les copies de secours, ces dernières ne font rien tant que la copie primaire fonctionne correctement (Redondance passive froide), soit, par les copies de secours elles mêmes (redondance passive tiède). Lorsque la copie primaire est cible de faute, une des copies de secours est élue pour la remplacer [22] [23] [24] [25] [26].

Redondance Active : La réplication active est basée sur le même traitement équitable de toutes les copies de tâches. Chacune traite tous les messages reçus, met à jour son état interne de la même manière, et génère les messages de sortie. Le choix des messages de sortie effectifs est fait par le moyen d'une fonction de décision qui dépend des hypothèses de défaillance. Par exemple et pour des arrêts simples, la fonction de décision peut être de choisir le premier message disponible, alors que pour des défaillances arbitraires, le choix peut être fait au moyen d'une fonction de décision à vote majoritaire [11] [27] [28] [29].

Redondance hybride Dans la redondance hybride [30] [31] [1] tout comme dans la réplication active toutes les copies reçoivent les messages d'entrée et peuvent ainsi les traiter. Cependant, comme pour la réplication passive, le traitement n'est pas le même car seulement la copie privilégiée (primaire) assume la responsabilité de certaines décisions (par exemple, sur l'acceptation des messages, ou sur la préemption du traitement en cours).

Comparaison entre les trois types de redondance : Le Tableau 1.3 compare les différentes approches de tolérance aux pannes basées sur la redondance active, passive ou hybride des tâches et/ou des communications.

On note ici que avec la redondance active une faute n'augmente pas la latence du système temps réel, ce qui n'est pas le cas de la redondance passive, où une faute de la réplique primaire peut augmenter la latence de manière significative. Cependant, la redondance passive permet une meilleure utilisation des ressources matérielles offertes par l'architecture en réduisent aux maximum le surcharge sur les processeurs et sur le réseau de communication. Le choix d'une stratégie de réplication est guider par les contraintes et les besoins applicatifs. Par exemple, il est plus logique d'utiliser la réplication active en cas de défaillances fréquentes des composants matériels, et la réplication passive lorsque le nombre de communications est élevé.

1.4.4 Techniques matérielle de tolérance aux fautes matérielles

Dans la plupart des systèmes critiques, comme les commandes de vol ou les circuits hydrauliques des avions, certains actionneurs et capteurs sont doublés et

	Approche		
	Redondance active	Redondance passive	Redondance hybride
Temps de réponse	un temps de réponse prévisible, et rapide (architectures avec un taux élevé de parallélisme)	temps de réponse meilleur en cas d'absence de fautes ; la panne de la réplique primaire augmente le temps de réponse	temps de réponse meilleur en cas de présence de fautes
Détection de pannes	pas de mécanisme	mécanisme dédie	mécanisme dédie
Traitement des erreurs	par compensation	par recouvrement	par compensation et par recouvrement
Reprise après panne	immédiate	non immédiate	immédiate (compensation) et non immédiate (recouvrement)

TABLE 1.3 – Comparaison entre les trois approches de redondance.

même triplés par fois. Une défaillance d'un composant peut donc être compensée par les autres copies, et comme la défaillance de chaque composant est un évènement complètement indépendant de celle des deux autres, et que les composants sont supposés de bonne fiabilité, alors il est extrêmement improbable que les trois composants soient fautifs.

Supposons que nous appliquons les mêmes entrées à deux circuits logiques et que nous comparons les sorties. Si nous obtenons le même résultat, nous pouvons conclure que les deux composants sont fonctionnels, ou qu'ils présentent une défaillance tous les deux. Le problème de devient délicat si l'un seulement des deux défaille, et dans ce cas on ne peut pas savoir lequel exactement juste en comparant leurs sorties. C'est la principale limite des systèmes DMR (Dual Modural Redundant). En dupliquant un composant à la redondance la situation s'améliore : avoir les mêmes sorties implique que les trois composants sont fonctionnels (avec une grande probabilité) ou qu'ils défaillent simultanément (cas très peu probable), et si on a deux des trois sorties qui sont identiques, alors il est plus probable que le composant ayant une sortie différente est le composant défaillant. Cette redondance est dite redondance modulaire triple TMR (Triple Modular Redundancy) [32] [33].

Redondance modulaire triple : La Figure 1.7 montre un circuit TMR [34] [35] avec trois circuits logiques identiques A,B et C ayant la même entrée . Les sorties sont comparées à l'aide du composant voteur qui donne un résultat correct à la majorité des votes. Si aucun ou l'un des composant défaille le résultat délivré est correct, mais si les deux ou les trois composants défaillent, le résultat sera erroné. Certains circuits défaillants ont une sortie mise à 1 ou à 0, dans ce cas, cette information sera prise en compte par le voteur.

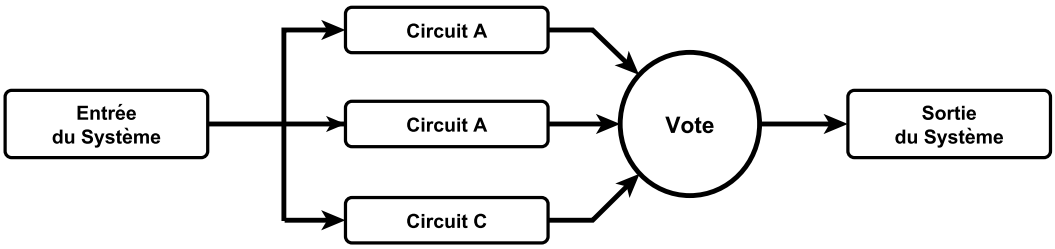

FIGURE 1.7 – Redondance Modulaire Triple

1.5 Mise en œuvre de la tolérance aux fautes

Si on augmente la capacité de traitement fonctionnel d'un composant par un mécanisme de détection d'erreur on se trouve avec un composant auto testable ; le principal intérêt est la possibilité de définir clairement des zones de confinement d'erreur.

Éviter que la propagation d'erreurs n'affecte le travail des composants non défaillants est la mission principale de la coordination des activités de composants multiples. Cet aspect devient particulièrement important lorsqu'un composant donné doit intéragir et communiquer avec d'autres composants (les informations communiquées peuvent êtres des données locales de capteurs ou la valeur d'une horloge locale) [9]..

La tolérance aux fautes est aussi récursive : on doit aussi protéger les mécanismes destinés à mettre en œuvre la tolérance aux fautes puisque eux aussi doivent être protégés contre les fautes susceptibles de les affecter. Des exemples sont fournis dans [17], ils utilisent la rédondance des voteurs, par les contrôleurs auto-testables, où la notion de mémoire stable est utilisée pour sauvegarder les programmes et les données de la reprise .

1.6 Conclusion

Dans ce chapitre nous avons présenté les terminologies et les principes de la sûreté de fonctionnement. En s'appuyant sur la notion de tolérance aux fautes matérielles. Les principales méthodes ont été détaillées, d'une part pour la redondance matérielle et surtout pour la redondance logicielle.

Enfin, les principales tendances et les défis posés à l'informatique, en ce qui concerne la tolérance aux fautes, ont été brièvement exposées. Le lecteur est encouragé à approfondir l'ensemble de ces aspects à partir de la bibliographie.

Bibliographie

[1] P. Chevochot and I. Puaut, "Scheduling fault-tolerant distributed hard real-time tasks independently of the replication strategies," in *Real-Time Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference on*, pp. 356–363, IEEE, 1999. (Cité en pages 1, 14 et 18.)

[2] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 389–402, 2009. (Cité en pages 1 et 14.)

[3] K. Chaaban, M. Shawky, and P. Crubille, "A distributed framework for real-time in-vehicle applications," in *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pp. 925–929, IEEE, 2005. (Cité en pages 1 et 14.)

[4] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems," in *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*, pp. 864–869, IEEE Computer Society, 2005. (Cité en pages 1 et 14.)

[5] J.-C. Laprie, *Guide de la sûreté de fonctionnement*. Cépadués, 1995. (Cité en page 1.)

[6] J. Arlat, Y. Crouzet, Y. Deswarte, J.-C. Laprie, D. Powell, P. David, J. Dega, C. Rabéjac, H. Schindler, and J.-F. Soucailles, "Fault tolerant computing," *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999. (Cité en page 1.)

[7] J. Arlat, A. Costes, J.-P. Blanquart, and J.-C. Laprie, *Guide de la sûrté de fonctionnement*. Cépaduès-éditions, 1996. (Cité en page 2.)

[8] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004. (Cité en page 3.)

[9] J. Arlat, Y. Crouzet, Y. Deswarte, J.-C. Fabre, J.-C. Laprie, and D. Powell, "Tolérance aux fautes," *Encyclopédie de l'informatique et des systèmes d'information. Vuibert, Paris, France*, p. 92, 2006. (Cité en pages 3, 5, 6, 10, 11, 13, 15, 17 et 20.)

[10] J.-C. Laprie, "Sûreté de fonctionnement des systèmes : concepts de base et terminologie," *Revue de l'électricité et de l'électronique (REE),*, vol. 11, pp. 95–105, 2004. (Cité en pages 3 et 5.)

[11] J.-C. Laprie, *Dependability : Basic Concepts and Terminology*. Springer-Verlag, 1992. (Cité en pages 6, 8 et 18.)

[12] M. Marouf, *Ordonnancement temps réel dur multiprocesseur tolérant aux fautes appliqué à la robotique mobile*. PhD thesis, école nationale supérieure des mines de Paris, 2012. (Cité en page 10.)

[13] A. Avižienis, "Design of fault-tolerant computers," in *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, AFIPS '67 (Fall), (New York, NY, USA), pp. 733–743, ACM, 1967. (Cité en page 10.)

[14] A. Burns and A. J. Wellings, *Real-Time Systems and Programming Languages : ADA 95, Real-Time Java, and Real-Time POSIX*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 3rd ed., 2001. (Cité en pages 10 et 12.)

[15] J. Wakerly, *Error detecting codes, self-checking circuits and applications*. Elsevier, 1978. (Cité en page 12.)

[16] P. A. Lee and T. Anderson, *Fault tolerance*. Springer, 1990. (Cité en page 13.)

[17] G. Muller, M. Banâtre, N. Peyrouze, and B. Rochat, "Lessons from ftm : an experiment in design and implementation of a low-cost fault tolerant system," *Reliability, IEEE Transactions on*, vol. 45, no. 2, pp. 332–340, 1996. (Cité en pages 13 et 20.)

[18] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Morgan Kaufmann, 2010. (Cité en page 14.)

[19] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 375–408, 2002. (Cité en page 15.)

[20] R. Guerraoui and A. Schiper, "Fault-tolerance by replication in distributed systems," in *Reliable Software Technologies-Ada-Europe'96*, pp. 38–57, Springer, 1996. (Cité en page 17.)

[21] R. Guerraoui and A. Schiper, "Software-based replication for fault tolerance," *Computer*, vol. 30, no. 4, pp. 68–74, 1997. (Cité en page 17.)

[22] X. Qin, Z. Han, H. Jin, L. Pang, and S. Li, "Real-time fault-tolerant scheduling in heterogeneous distributed systems," in *Proceeding of the International Workshop on Cluster Computing-Technologies, Environments, and Applications (CC-TEA'2000)*, 2000. (Cité en page 18.)

[23] C. Dima, A. Girault, C. Lavarenne, and Y. Sorel, "Off-line real-time fault-tolerant scheduling," in *Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on*, pp. 410–417, IEEE, 2001. (Cité en page 18.)

[24] Y. Oh and S. H. Son, "Scheduling real-time tasks for dependability," *Journal of the Operational Research Society*, pp. 629–639, 1997. (Cité en page 18.)

[25] K. Ahn, J. Kim, and S. Hong, "Fault-tolerant real-time scheduling using passive replicas," in *Fault-Tolerant Systems, 1997. Proceedings., Pacific Rim International Symposium on*, pp. 98–103, IEEE, 1997. (Cité en page 18.)

[26] R. Al-Omari, A. K. Somani, and G. Manimaran, "A new fault-tolerant technique for improving schedulability in multiprocessor real-time systems," in *Parallel*

and Distributed Processing Symposium., Proceedings 15th International, pp. 8–pp, IEEE, 2001. (Cité en page 18.)

[27] P. Ramanathan and K. G. Shin, "Delivery of time-critical messages using a multiple copy approach," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 2, pp. 144–166, 1992. (Cité en page 18.)

[28] A. Girault, M. Sighireanu, Y. Sorel, *et al.*, "An algorithm for automatically obtaining distributed and fault-tolerant static schedules," in *Proceeding of International Conference on Dependable Systems and Networks*, pp. 165–190, 2003. (Cité en page 18.)

[29] P. Felber and A. Schiper, "Optimistic active replication," in *Distributed Computing Systems, 2001. 21st International Conference on.*, pp. 333–341, IEEE, 2001. (Cité en page 18.)

[30] P. Felber, X. Defago, P. Eugster, and A. Schiper, "Replicating corba objects : a marriage between active and passive replication," in *Distributed Applications and Interoperable Systems II*, pp. 375–387, Springer, 1999. (Cité en page 18.)

[31] K. Hashimoto, T. Tsuchiya, and T. Kikuno, "Effective scheduling of duplicated tasks for fault tolerance in multiprocessor systems," *IEICE Transactions on Information and Systems*, vol. 85, no. 3, pp. 525–534, 2002. (Cité en page 18.)

[32] V. Bobin, S. Whitaker, and G. Maki, "Links between n-modular redundancy and the theory of error-correcting codes," in *Idaho Univ, The 1992 4 th NASA SERC Symposium on VLSI Design 11 p(SEE N 94-21694 05-33)*, 1992. (Cité en page 19.)

[33] T. Smith and J. N. Yelverton, "Processor architectures for fault tolerant avionic systems," in *Digital Avionics Systems Conference, 1991. Proceedings., IEEE/AIAA 10th*, pp. 213–219, IEEE, 1991. (Cité en page 19.)

[34] J.-x. Zou and H.-b. Xu, "Design and reliability analysis of emergency trip system with triple modular redundancy," in *Communications, Circuits and Systems, 2009. ICCCAS 2009. International Conference on*, pp. 1006–1009, IEEE, 2009. (Cité en page 19.)

[35] T. J. Dysart and P. M. Kogge, "System reliabilities when using triple modular redundancy in quantum-dot cellular automata," in *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pp. 72–80, IEEE, 2008. (Cité en page 19.)