THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)

Présentée et soutenue par :
Florian MANY

le 18 Février 2013

Titre :

Combinaison des aspects temps réel et sûreté de fonctionnement pour la conception des plateformes avioniques

---

École doctorale et discipline ou spécialité :

ED MITT : Sureté de logiciel et calcul de haute performance

Unité de recherche :

ISAE ONERA-MOIS

Directeur(s) de Thèse :

Frédéric BONIOL et David DOOSE

Jury :

Mr Frédéric BONIOL - Directeur de thèse
Mr Jean-Christophe BONNET - Examinateur
Mme Eva CRUCK - Examinateur
Mr David DOOSE - Co-directeur de thèse
Mr Jean-Charles FABRE - Examinateur
Mr Pierre-Emmanuel HLADIK - Examinateur
Mme Françoise SIMONOT-LION - Rapporteur
Mr Franck SINGHOFF - Rapporteur

Remerciements

Tant attendue, cette page de « liberté » du doctorant s'offre enfin à moi. A l'heure de remercier tout ceux qui, techniquement, socialement ou familialement, ont contribué à l'accomplissement de cette thèse, je n'oublie pas le hasard qui m'a conduit à faire de la recherche. Par l'intermédiaire initial d'un double diplôme qui devait être validé par un projet de fin d'études orienté recherche, j'y ai découvert un monde passionnant, autre versant de la technique que m'avait apporté l'école d'ingénieur. Bien qu'à l'heure d'écrire ces remerciements ma voie professionnelle ne soit plus celle de la recherche, je souhaite ne jamais trop m'en éloigner, et qui sait, y revenir un jour.

Ainsi je ne remercierai jamais assez David, tuteur avant-gardiste de mon projet de fin d'études et qui m'a dirigé tout au long de cette thèse, sous l'oeil bienveillant de Frédéric. Il a toujours su m'encourager, me montrer le chemin à parcourir et dans certains moments de flottement (ou d'oisiveté) me faire les piqûres de rappel nécessaires à l'accomplissement de ces travaux. En dehors du contexte de ma thèse, il m'a permis d'accéder à de nombreuses connaissances techniques sur des sujets de recherche variés. Au delà de David et Frédéric, c'est l'ensemble de l'ONERA/DTIM que je remercie, tous ces chercheurs qui ne m'ont jamais fermé leur porte de bureau pour répondre à mes interrogations, et échanger sur des problématiques de recherche. Et compte tenu du nombre de mes questions, ils ont été très patients ! Une pensée particulière pour Josette, l'ange gardien du DTIM, depuis partie pour une retraite bien méritée, qui m'a poussé à venir au café du matin avec les « anciens ». J'en garderai un souvenir mémorable !

Je n'oublie pas non plus dans quel contexte particulier cette thèse a été initiée. Ingénieur militaire sorti d'école, cette place en recherche a été le fruit des efforts faits par DGA Techniques aéronautiques (ex Centre d'Essais Aéronautiques de Toulouse). A ce titre, je remercie Jean-Christophe, chef de la division Systèmes Informatiques Embarqués, d'avoir porté ce projet de décaler mon arrivée au sein de son équipe de trois années pour me former à la recherche. De même, je remercie la sous-direction technique de DGA Techniques aéronautiques d'avoir soutenu ce projet. Mes passages réguliers au sein de ma division ont été pour moi l'occasion de ne jamais perdre de vue les problématiques des programmes d'armement. Et au même titre que les personnels de l'ONERA, je remercie ceux qui allaient devenir mes futurs collègues d'avoir donné de leur temps pour répondre à mes questions.

Etre doctorant, c'est aussi côtoyer au quotidien une farandole de doctorants, de ceux qui deviennent docteurs et partent, à ceux de sa propre promotion jusqu'aux nouveaux entrants. Alors, pêle-mêle, je remercie Julien, Stéphanie, Cédric, Pierre et Sophie, pour les heures passées à coincher et qui représentent mes « anciens ». Je n'oublie pas non plus Vincent, Mikel et Jean-Baptiste, ma « promotion », qui sont devenus docteurs avant moi. Et puis, à tous ceux qui suivent et que j'ai côtoyés, je leur souhaite d'atteindre ce qui constitue pendant trois années, ou plus, un graal. Je n'oublie pas non plus mes

camarades thésards de promotion ENSICA, avec en tête François, avec qui nous avons souvent échangé sur nos destinées respectives depuis la fin de l'école.

J'ai aussi profité de ces années passées à l'ONERA pour découvrir le tennis entreprise au sein de l'équipe du DTIM. Alors, malgré deux finales perdues, je remercie Jean-Loup, Julien, Boris, David et Frédéric pour les heures passées sur les courts. Il est à noter qu'il est sans doute peu courant de jouer en double avec son directeur de thèse ! J'en profite aussi pour remercier Virginie, dont aucun entraîneur sain d'esprit ne réussira jamais à comprendre le revers.

Particularité de ma formation militaire, je possède une licence de corps technique militaire, qui me permet de voler sur des avions TB 20. Je remercie Jean-François, Gonzague et Yves, mes compagnons de vol et de la promotion 2008, pour ces heures de vol passées sur l'ensemble du territoire français. Ces moments ont constitué des à côté salvateurs pour mon travail de thèse.

In fine, ces remerciements ne seraient pas complets sans mes soutiens quotidiens. Je remercie ma femme, Anne-Sophie, qui en plus de m'avoir accompagné au cours de ces années, m'a donné en 2010 l'objet de recherche le plus curieux et le moins déterministe qui soit, mon fils Loën. Chaque soir en franchissant le palier, une nouvelle journée commençait, une journée qui me permettait de déconnecter mon cerveau de la recherche. Ces remerciements se concluent sur ceux que j'adresse à mes parents et à mon frère Cédric, qui fondent les racines d'une stabilité et d'une confiance qui me poussent toujours aller plus loin et à me dépasser.

Anima Sana in Corpore Sano

Table des matières

Table des figures

Introduction générale

La conception moderne de plateformes avioniques, qu'elles soient destinées aux avions ou aux drones, tend à donner une importance croissante aux systèmes informatiques embarqués. Ces systèmes informatiques, dit temps réel, assurent des fonctions critiques telles que les lois de commande pour les commandes de vol électriques, la gestion du kérosène ou l'anticollision en vol. La défaillance d'un ou plusieurs de ces systèmes pouvant avoir des conséquences catastrophiques, leur conception et leur validation sont des points cruciaux de tout développement. Cette validation se concrétise par l'emploi de méthodes et de techniques permettant de prouver la fiabilité de tels systèmes.

Un processus complet de validation d'un systèmes temps réel dépend des différents points de vue considérés durant la conception. Chaque point de vue doit faire l'objet d'une validation qui lui est propre, l'aggrégation de l'ensemble de ces validations aboutissant à la conclusion sur la fiabilité du système. Sont ainsi concernées les questions d'architectures fonctionnelles, de contraintes temporelles, de comportement en présence d'une ou plusieurs défaillances, ou encore d'alimentation électrique. Aujourd'hui, le processus de validation prend en compte chaque validation de point de vue comme une entité indépendante. L'étude des interactions entre les contraintes temporelles et la gestion des défaillances, et la méthode de validation qui peut lui être associée, motive ces travaux de thèse.

De plus, par essence, les systèmes embarqués décrits agissent et réagissent à leur d'environnement de fonctionnement ou leur environnement d'installation. En particulier, ces environnements peuvent perturber le système au point de le faire défaillir, souvent de manière transitoire, rendant son comportement potentiellement complexe à prévoir. Ainsi, pour l'environnement extérieur, les perturbations dues aux radiations en haute altitude, aux radars ou aux agressions volontaires - cas de la guerre électronique - sont à envisager dans le cadre de l'exploitation du système. De même, pour l'environnement d'installation , les questions afférantes à la compatibilité électromagnétique, aux différents régimes vibratoires ou thermiques sont à considérer. L'intégration de ces considérations dans le processus de validation est une autre motivation pour ces travaux.

Problématique de l'étude

L'objectif du présent manuscrit est l'étude de la combinaison des aspects sûreté de fonctionnement et temps réel dans la conception de plateformes avioniques. L'approche retenue pour remplir cet objectif est de diviser l'étude en deux questions réciproques :

1. Quel est l'impact de la sûreté de fonctionnement sur les aspects temps réel ?
2. Quel est l'impact des aspects temps réel sur la sûreté de fonctionnement ?

La première question est étudiée au niveau des ordonnancements des systèmes temps réel. Dans le contexte retenu, *id est* un environnement agressif, la ligne directrice suivie est d'évaluer et d'analyser l'impact temporel des mécanismes de tolérance aux fautes au sein des analyses d'ordonnançabilité. Du point de vue théorique, ces analyses concernent les ordonnancements dit tolérants aux fautes.

Toute réponse à la seconde question impose quant à elle de dépasser la question du respect des échéances temporelles pour s'intéresser à l'impact réel de l'ordonnanceur dans la sûreté de fonctionnement. Pour mettre en évidence cet impact, définir le comportement du matériel en présence d'un environnement agressif, étudier les échanges de données et enfin analyser le comportement de l'ordonnanceur sont autant d'étapes à franchir. Les deux questions sont abordées dans la suite du manuscrit au travers de la structure décrite ci-après.

Plan de lecture du document

Ce document est découpé en trois parties, la première est consacrée à la présentation du contexte théorique de l'étude, les deux suivantes aux différentes contributions de la thèse :

Introduction aux systèmes temps réel

Cette partie se compose d'un chapitre unique consacré à l'introduction aux systèmes temps réel et à leurs problématiques :

Chapitre 1 : Des systèmes temps réel et de leurs problématiques

La première partie de ce chapitre rappelle la définition d'un système temps réel, ses principales caractéristiques et les problématiques qui lui sont associées. La seconde partie de ce chapitre met l'accent sur la théorie de l'ordonnancement temps réel. Cette partie traite des principales architectures matérielles des sytèmes temps réel, de la définition et de la modélisation des tâches informatiques ainsi que principales contraintes des ensembles de tâches. Sur ces bases, les méthodes de validation des ensembles de tâches sont présentées.

Ordonnancement tolérant aux fautes

Cette deuxième partie aborde la question des ordonnancements tolérants aux fautes. Elle se compose de quatres chapitres, détaillés ci-après :

Chapitre 2 : Etat de l'art

Ce chapitre présente les principaux résultats des travaux antérieurs à cette étude sur les ordonnancements temps réel tolérants aux fautes. La première partie de ce chapitre

décrit les principales distributions temporelles de fautes existantes, et leurs applications. La deuxième partie se consacre à la description des méthodes de détection et de correction d'erreurs dans les programmes informatiques. La dernière partie aborde les méthodes de validation des ordonnancements tolérants aux fautes. Ces méthodes dépendent de la distribution temporelle de fautes et des méthodes de correction et détection d'erreurs.

Chapitre 3 : Rafales de fautes et taxonomie de la gestion des erreurs

Ce chapitre définit un nouveau modèle de distribution temporelle appelé rafales de fautes. Ce modèle sert à décrire des phénomènes perturbant un système de façon durable dans le temps. Cette proposition de modélisation permet de considérer des phénomènes non pris en compte par les travaux antérieurs. L'autre contribution de ce chapitre est la définition d'une taxonomie autour des notions de stratégie de recouvrement d'erreurs et de tactique de recouvrement d'erreurs.

Cette taxonomie remplit trois objectifs. Elle autorise d'abord le classement des travaux antérieurs suivant les méthodes de détection et de correction étudiées. La taxonomie met ensuite en lumière le comportement de l'ordonnanceur vis à vis des tâches préemptées lorsqu'une erreur est détectée. Enfin, elle permet d'évaluer le coût temporel de chaque stratégie. Les deux aspects abordés dans ce chapitre permettent donc la définition du cadre pour l'étude d'ordonnançabilité.

Chapitre 4 : Ordonnnançabilité sous rafales de fautes

Ce chapitre propose une méthode de validation des ordonnancements tolérants aux rafales de fautes. La première partie de ce chapitre décrit cette méthode basée sur l'évaluation du pire temps de réponse des tâches temps réel. Ce pire temps de réponse tient compte de la durée de la rafale de fautes, ainsi que de la stratégie de recouvrement d'erreur de l'ordonnanceur. La deuxième partie de ce chapitre aborde l'évaluation temporelle des stratégies de recouvrement d'erreurs. Les méthodes analytiques proposées permettent de quantifier le surcoût temporel de ces stratégies et de les comparer.

Enfin, la troisième partie compare les différentes stratégies de recouvrement d'erreurs sur des ensembles de tâches générés aléatoirement. Pour chaque ensemble de tâches, la durée de la rafale de fautes varie, tout comme la stratégie retenue. L'agrégation de l'ensemble des résultats obtenus permet d'argumenter le choix d'une stratégie par rapport à une autre. De plus, cette simulation met en évidence l'importance de la réactivité des systèmes temps réel en présence de rafales de fautes, notamment dans l'anticipation de la présence d'erreurs non encore détectées.

Chapitre 5 : Application aux ordonnancements partitionnés

Ce chapitre s'intéresse aux ordonnancements partitionnés, et décline la méthode de validation des ordonnancements tolérants aux rafales de fautes à ce contexte particulier. La méthode de validation proposée se base sur un algorithme de validation. Ce chapitre clos la première partie contributive consacrée à l'étude des ordonnancements tolérants

aux fautes.

Robustesse des ordonnancements

Cette troisième partie traite de l'impact des ordonnancements tolérant aux fautes dans les analyses de sûreté de fonctionnement au travers de la notion de robustesse des ordonnancements. Deux chapitres, décrits dans la suite, composent cette contribution :

Chapitre 6 : Introduction à la robustesse des ordonnancements

Ce chapitre introduit la problématique de la robustesse des ordonnancements. En particulier, il présente le besoin de disposer d'une métrique pour évaluer les limites de la tolérance aux fautes des ordonnancements, notamment en citant l'état de l'art du domaine.

Chapitre 7 : Robustesse des ordonnancements

Ce chapitre présente les travaux réalisés sur l'évaluation de la robustesse des ordonnancements soumis à des rafales de fautes. Dans ce cadre, cette robustesse, qualifiée de résilience aux rafales de fautes, s'évalue par une méthode analytique directe basée sur l'équation du temps de réponse. En complément, une méthode d'évaluation pour le cas des fautes pseuso-périodiques est aussi présentée.

Vers une analyse de niveau système

Cette quatrième partie présente les travaux réalisés pour ouvrir la réflexion de l'étude des systèmes temps réel soumis à des perturbations vers le niveau système. Deux chapitres composent cette partie :

Chapitre 8 : Modéliser un système soumis à des rafales de fautes

Ce chapitre propose une modélisation de systèmes temps réel mettant en relief les effets de l'ordonnancement sur la sûreté de fonctionnement. Cette modélisation tient compte à la fois du comportement individuel de l'élément modélisé soumis à une perturbation et du comportement dysfonctionnel des architectures matérielles et logicielles. L'étude du système s'effectue par l'application de diverses fonctions aux éléments du modèle dans le but de propager les flots de données et les effets de la perturbation au travers du système. Les aspects modélisation et simulation sont décrits au travers d'un exemple introductif.

Première partie

Introduction aux systèmes temps réel

Chapitre 1

Des systèmes temps réel et de leurs problématiques

Sommaire

L'accroissement de la présence des systèmes temps réel au quotidien est concommitant avec les progrès de l'informatique depuis le début des années 1970. L'aéronautique, dans les commandes de vol électrique numériques, l'automobile, au travers de l'*antiblockiersystem* (ABS), le transport ferroviaire, notamment dans l'automatisation de la ligne 14 (*meteor*) du métropolitain parisien ou le contrôle des centrales nucléaires sont autant de domaines mettant en œuvre ces systèmes.

Le point commun de ces différentes applications est le caractère catastrophique de toute défaillance majeure. En effet, la perte physique d'un système temps réel autant que la perte de contrôle conduit irrémédiablement à des pertes humaines. De ce fait, la conception de tels systèmes met en œuvre des problématiques spécifiques qui impactent directement la spécification technique de ces systèmes.

Ainsi, de tels systèmes doivent être fiables, avec un comportement, notamment temporel, connu et démontré *a priori*. L'objectif de ce chapitre est donc de proposer au lecteur les différentes problématiques afférantes aux systèmes temps réel. Ce chapitre présente les définitions et les résultats essentiels pour appréhender les travaux menés dans cette thèse.

Ce chapitre se structure comme suit. La section 1.1 introduit les systèmes temps réel et leurs principales caractéristiques. La section 1.2 présente la théorie de l'ordonnancement temps réel et ses principaux résultats. Enfin, la section 1.3 conclut sur les travaux décrits en les replaçant dans le cadre de ces travaux de thèse.

1.1 Introduction aux systèmes temps réel

Les applications intégrant des sytèmes temps réel sont interfacées avec un environnement par nature dynamique, appelé **procédé**. Un système est dit **réactif** si son système informatique interne voit son fonctionnement conditionné par l'évolution du procédé qu'il doit contrôler. Ainsi, le système suit et contrôle le procédé suivant des contraintes temporelles issues de la conception de l'application. Par extension, la dénomination de systèmes **temps réel** couvre la capacité du système à fournir un résultat exact à un moment précis. La nature de la contrainte du système temps réel s'articule donc autour du dyptique valeur et temps.

A ce titre la validité du système n'est pas uniquement le résultat de la performance temporelle du logiciel. le procédé ayant une dynamique d'évolution qui lui est propre, de quelques secondes à plusieurs heures, les échéances temporelles à respecter sont dépendantes de ses grandeurs physiques mesurables. Ces considérations font partie intégrante des spécifications du système à implanter.

La figure 1.1 présente un modèle de système temps réel. Des capteurs interfacent le système informatique avec le procédé. Suivant les entrées provenant du procédé, dénomées *stimuli*, ce système effectue des opérations qui produisent, par l'intermédiaire des actionneurs, des sorties applicables au procédé. Ces sorties sont qualifiées de *réactions* ou *commandes*. Ce système réactif est un système dit de *contrôle-commande*.

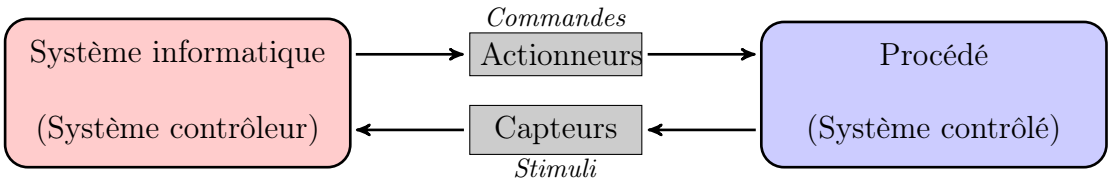


FIGURE 1.1 – Exemple de système temps réel

1.1.1 Classification des systèmes temps réel

Les spécifications techniques définissant les contraintes temporelles des systèmes temps réel tiennent compte de plusieurs facteurs. En premier lieu, la dynamique du procédé conduit à considérer la pérennité des données issues des capteurs, et du comportement

du système informatique sous-jacent. Une première contrainte temporelle est de fait liée au temps de validité des données et à leur fraîcheur.

En second lieu, les spécifications définissent les échéances temporelles des différents calculs produits par le système informatique. L'instant de délivrance des résultats, et les actions qui en résultent, assurent le contrôle du procédé. Ainsi, le non respect de ces échéances peut rendre le système temps réel inapte à remplir sa mission, entraînant des conséquences pouvant s'avérer catastrophique. La rigueur avec laquelle doivent être respectées les échéances temporelles permet de définir une classification des systèmes temps réel, suivant trois catégories :

- **Systèmes temps réel à contraintes strictes** : Aussi qualifiés de systèmes temps réel *durs*, leurs contraintes temporelles doivent impérativement respectées, sous peine de provoquer des évènements pouvant être catastrophiques (perte du système entraînant des pertes humaines). Les commandes de vols électriques numériques des avions modernes, les sytèmes de signalisation de voie ferrée et les systèmes contrôlant les fissions des réacteurs nucléaires en sont autant d'exemples.
- **Systèmes temps réel à contraintes relatives** : qualifiés aussi de systèmes temps réel *souples*, ils tolèrent un nombre spécifié de dépassement d'échéances temporelles. Ces systèmes sont typiquement présents dans les application multimédia, où la perte d'images est acceptable dans une diffusion de vidéo. Derrière ces systèmes se dessine la notion de qualité de service.
- **Systèmes temps réel à contraintes mixtes** : systèmes hybrides mettant en œuvre les deux aspects précédents, leur existence est liée à la mutualisation des ressources de calcul des systèmes, permettant de faire cohabiter des applications critiques et non critiques. Les évolutions des systèmes de bord avioniques permettent d'associer sur un même système des fonctions critiques avion et celles liées au multimédia des passagers.

Les travaux de cette thèse s'intéressent exclusivement aux **systèmes temps réel à contraintes strictes**. Ces systèmes sont **critiques** car le non respect d'échéance peut conduire à une catastrophe. Les propriétés principales qui seront abordées ont trait à la prédictibilité, le déterminisme et la fiabilité de tels systèmes. Plus avant, ces propriétés peuvent être définies comme suit :

- **Prédictibilité :** quelque soit le contexte, les performances du système sont connues. Ces performances sont bornées par la définition d'un pire cas (cf. section 1.2).
- **Déterminisme :** quelque soit le contexte, l'état du système est connu. L'incertitude sur le comportement du système est exclue.
- **Fiabilité :** en conditions nominales, les contraintes temps réel sont respectées. En conditions dégradées (pannes), le système temps réel reste fiable, sans conduire à une catastrophe.

1.1.2 Systèmes temps réel embarqués critiques

A l'ensemble des contraintes précédemment décrites, ces travaux de thèse ajoutent la **contrainte d'embarquabilité**. Un système embarqué (*embedded system*, système enfoui en anglais) est un système informatique intégré à un système plus large. Le procédé auquel il est interfacé est alors, par exemple, l'attitude d'un avion (roulis, lacet, tangage), la vitesse d'une automobile, ou la trajectoire d'un drone.

Ces contraintes d'embarquabilité se traduisent par la limitation des ressources disponibles. Ainsi, la puissance de calcul, la mémoire sont limitées par des problématiques de masse, de volume, consommation d'énergie, mais aussi par des limitations pécuniaires. La prise en compte de l'ensemble de ces contraintes dicte les spécifications techniques et la conception de ces systèmes.

1.2 Théorie de l'ordonnancement temps réel

La démonstration du respect des échéances temporelles des systèmes décrits précédemmet est l'objectif majeur de la théorie de l'ordonnancement temps réel. Les spécifications techniques portant sur le contrôle de la dynamique du procédé se traduisent au niveau du système informatique par des contraintes temps réel sur la fourniture de résultats. La problématique de la démonstration *a priori* de ces contraintes est abordée depuis le début des années 1970. L'état de l'art présenté dans cette section contient les principaux résultats de la théorie de l'ordonnancement temps réel.

Cette section ce structure comme suit. La sous-section 1.2.1 présente les principales architectures temps réel. La sous-section 1.2.2 décrit les principales caractéristiques des tâches temps réel, et la sous-section 1.2.3 aborde la question des algorithmes d'ordonnancement. La sous-section 1.2.4 présente la problématique du partage de ressources entre tâches. Enfin, la sous-section 1.2.5 donne un ensemble de techniques permettant de valider des ensembles de tâches.

1.2.1 Architecture

La conception des systèmes temps réel, et en particulier de leur système informatique, dépend des choix de l'architecture matérielle. Cette architecture matérielle concerne l'ensemble des ressources matérielles qui seront mises à la disposition du système pour remplir sa mission : ressources de calcul (processeurs), ressources de stockage (mémoire), ressources de communication (réseaux), périphériques d'entrée/sortie, etc. Dans le cadre de la théorie de l'ordonnancement, l'architecture des ressources de calcul est un paramètre discriminant pour les études. Ainsi, trois catégories d'architecture sont mises en avant :

– **Architecture monoprocesseur** la ressource de calcul du système temps réel est unique pour l'ensemble de ses fonctions.

- **Architecture multiprocesseur** l'ensemble des fonctions du système temps réel sont réparties sur une architecture parallèle de n processeurs possédant chacun une mémoire propre, Plus avant, les **architectures multicœurs** mutualisent leur mémoire suivant leur niveau de cache.
- **Architecture distribuée** Le système temps réel dispose de plusieurs processeurs, mais répartis sur différents sites, connectés par l'intermédiaire d'un réseau (*Controller Area Network* - CAN [Sta03], ARINC-429 [AEE04], *Avionics Full DupleX* - AFDX [AEE09]).

Dans la suite de ce manuscrit, l'accent est mis sur les systèmes temps réel à contraintes strictes dans le cadre d'architecture monoprocesseur. Ces systèmes informatiques possèdent une architecture logicielle à deux niveaux, l'un représentant l'ensemble des fonctions que doit réaliser le système temps réel pour contrôler le procédé, représentant la couche de haut niveau. La couche bas niveau contient le système d'exploitation temps réel (*Real-Time Operating System* (RTOS)).

Le contrôle du procédé est généralement réalisé pour plusieurs processus informatiques, appelés **tâches temps réel**. Chaque tâche remplit une ou plusieurs fonctions, et accède aux différents périphériques physiques (actionneurs, capteurs, communication) par l'intermédiaire des interfaces (pilotes) fournies par le système d'exploitation temps réel.

Le système d'exploitation temps réel, ou *exécutif temps réel*, gère l'accès aux périphériques physiques, et contrôle l'exécution des tâches temps réel par l'intermédiaire de l'ordonnanceur. Ce programme informatique est responsable de l'attribution des différentes ressources (processeur, mémoire, etc) aux tâches du système temps réel. Pour le processeur, cette attribution s'opère suivant une politique de choix pouvant se définir de plusieurs manières : affectation d'un crédit temporel (*Round Robin*) d'exécution pour chaque tâche, politique *First In-First Out*, ou par l'affectation de priorité aux tâches via des algorithmes d'ordonnancement (cf. sous-section 1.2.3).

En outre les attributions de l'exécutif temps réel peuvent être complétées par une capacité dite de préemption. En résulte deux modes d'exécutions des tâches temps réel, décrits ci-après :

- **non-préemptif :** Une tâche affectée sur le processeur s'exécute jusqu'à sa terminaison. Cette terminaison libère le processeur, et autorise l'accès d'une nouvelle tâche au processeur.
- **préemptif :** Suivant la politique de choix, l'ordonnanceur peut stopper l'exécution d'une tâche en cours au profit d'une autre. Cet arrêt d'exécution entraîne une sauvegarde de l'état courant de la tâche, afin de pouvoir la reprendre plus tard. La possiblité de préemption de l'ordonnanceur peut-être restreinte, notamment si la tâche en cours effectue une opération qui ne doit pas être interrompue (par exemple, lors d'un accès en écriture d'une zone mémoire).

Par ailleurs, le rôle de l'ordonnanceur peut s'étendre à la gestion de ressources partagées (cf. sous-section 1.2.4), ainsi qu'aux mécanismes de gestion d'erreurs (cf. chapitre

2). Les capacités offertes par l'ordonnanceur dépendent du système d'exploitation temps réel retenu lors de la conception, ou de son développement spécifique. Parmi les exécutifs temps réel existants, VxWorks, RTEMS, OSEK/VDX sont les plus usités par l'industrie.

1.2.2 Caractéristiques des tâches temps réel

Les contraintes temporelles issues de la spécification technique du système temps réel se déclinent au niveau des tâches temps réel. A ces contraintes s'ajoutent celles fonctionnelles, au travers des relations avec les autres tâches du système, mais aussi celles liées au partage de ressources. Cette sous-section aborde l'ensemble de ces points.

La modélisation d'une tâche informatique est une composition de plusieurs points de vue. Les attributs de chaque point de vue permettent d'en caractériser un aspect. Ainsi, le modèle comportemental définit l'état de la tâche à un instant donné de l'exécution et le modèle canonique en caractérise les propriétés temporelles. Les relations entre les tâches, ainsi que l'accès aux ressources, nécessitent des modèles supplémentaires. Cette partie présente ces différents modèles.

Modèle comportemental Le comportement d'une tâche est représenté par l'automate à états fini décrit sur la figure 1.2. Lorsqu'à un instant t une tâche se réveille, elle devient éligible (état E) sur le processeur. Lorsque l'ordonnanceur l'élit pour s'exécuter sur le processeur, son état change pour devenir actif (état A).

Depuis l'état actif, plusieurs changements d'état sont atteignables. Soit la tâche termine son exécution (état T). Soit la tâche a besoin d'une ressource non disponible, et elle passe en état d'attente (état W). La mise à disposition de la ressource la ramène à l'état éligible. Si l'ordonnanceur est préemptif, la tâche peut être stoppée en cours d'exécution, son état courant enregistré, et retourne à l'état éligible.

Modèle canonique Dans ces travaux, une tâche se note τ, et un ensemble de n tâches $\Gamma = \{\tau_1, ..., \tau_n\}$. La figure 1.3 représente le modèle canonique d'une tâche informatique τ_i périodique. Les principales caractéristiques temporelles sont :

- C_i, durée d'exécution maximale sur le processeur (*Worst Case Execution Time*)
- D_i, échéance relative avant laquelle doit être achevée l'exécution (*Deadline*)
- $d_{i,k}$, échéance absolue de la i-ème instance
- T_i, période d'exécution (*Period*)
- A_i, date de réveil de la première instance (*Arrival Time*)
- $a_{i,k}$, date de réveil de la i-ème instance de la tâche

Chaque exécution est appelée **instance de tâche**. Elles possèdent toutes une échéance absolue, liée à l'échéance relative par la relation $d_{i,k} = a_{i,k} + D_i$, avec $a_{i,k} = A_i + kT_i$. Lorsque l'échéance relative est égale à la période, la tâche est dite à **échéance sur requête**. Dans le cadre de ces travaux, l'échéance relative est supposée inférieure ou égale

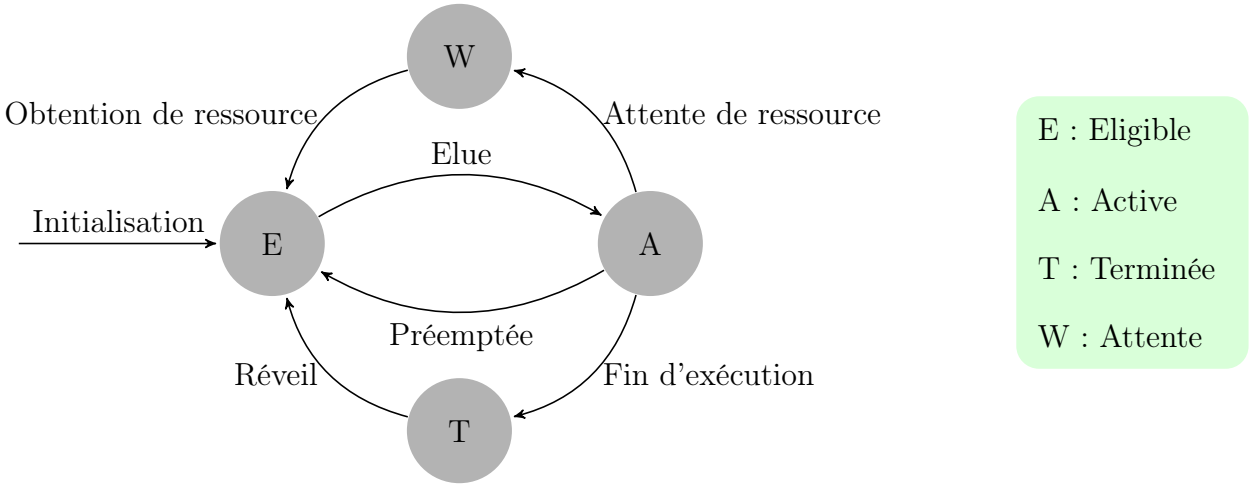


FIGURE 1.2 – Modèle comportemental de tâche préemptable

à la période ($D_i \leqslant T_i$).

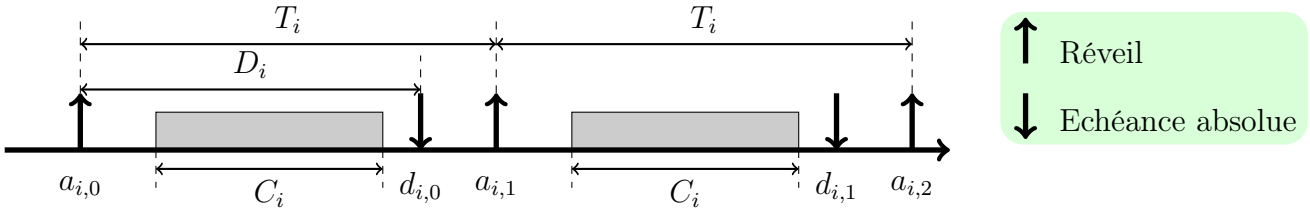


FIGURE 1.3 – Modèle canonique de tâche périodique

D'autres modèles de tâches existent, notamment définis par leur loi d'arrivée. Ainsi, une tâche peut être **sporadique** (ou pseudo-périodique) ou **apériodique**. Dans le cas d'une tâche sporadique, la période d'activation n'est pas connue *a priori*. Cependant, l'intervalle de temps minimal entre deux activations de la tâche est connu, et noté T_i. La tâche périodique est donc un cas particulier de la tâche sporadique.

Une tâche apériodique ne possède généralement ni date de réveil, ni de période, mais doit être traitée avant une échéance temporelle. Le réveil de ce type de tâche est conditionné par les évènements impactant le système et nécessitant un traitement pouvant être urgent. Ainsi, dans un avion, le système d'alarme (*Flight Warning System* (FWS)) peut générer des tâches apériodiques qui seront alors traitées par un serveur (tâche périodique chargée de traiter ces tâches)[DTB93].

Dépendance entre tâches Une tâche peut être dépendante d'autres tâches (dans le cas contraire, la tâche est dite **indépendante**). La communication entre tâches s'opère par mémoire partagée ou par message. L'instant de communication peut être contraint par une synchronisation (ou rendez-vous). Dans ce cas, un ordre partiel prédéterminé apparait. Ces relations sont connues par avance. Ce sont les **relations de précédence**. Si τ_i précède τ_j, la relation est notée $\tau_i \rightarrow \tau_j$.

L'ensemble des relations est représenté par un graphe de précédence (figure 1.4). Dans ce cas, la tâche τ_4 ne peut s'exécuter que si les tâches τ_3 et τ_2 sont terminées. Et leur éligibilité est conditionnée par la terminaison de τ_1.

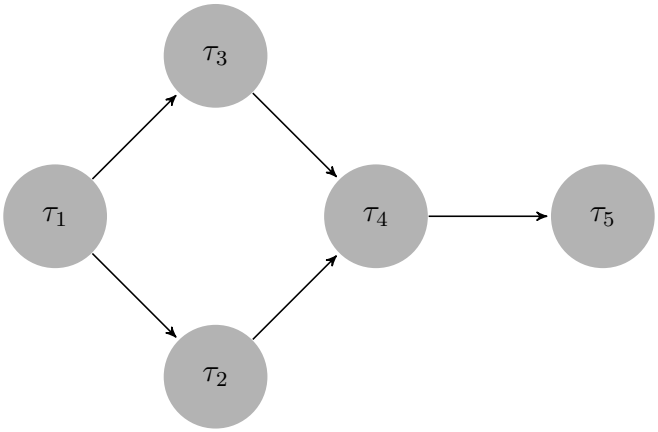


FIGURE 1.4 – Exemple de graphe de précédence

En pratique, les tâches ayant des relations de précédence possèdent des périodes soit identiques, soit multiples entre elles (périodes harmoniques). Cette contrainte supplémentaire permet de faciliter les rendez-vous entre les tâches et les synchronisations. De façon évidente, des tâches, à échéances sur requêtes, avec des périodes premières entre elles, ne pourraient communiquer que par des rendez-vous issues des multiples communs des périodes, qui seraient rapidement très grand. Pour autant, sous certaines conditions, la satisfaction de contraintes de précédences entre tâches de périodes différentes est possible, et ce sans mécanismes de synchronisation [For+10].

1.2.3 Algorithmes d'ordonnancement

Dans ces travaux de thèse, la politique de choix appliquée par l'ordonnanceur est basée sur l'affectation de priorités par l'intermédiaire d'un algorithme d'ordonnancement. Ces algorithmes peuvent être hors ligne ou en ligne, les priorités fixes ou dynamiques.

Les *algorithmes hors ligne* ont une approche statique de l'ordonnancement. A partir de l'ensemble des données temporelles, ils construisent une séquence d'exécution de toutes

les tâches, qui sera répétée indéfiniment. Cette approche est particulièrement rigide, et ne peut s'adapter aux changements de l'environnement.

Dans le cadre des *algorithmes en ligne*, l'ordonnanceur choisit la prochaine tâche à exécuter. Son choix se fonde à la fois sur des paramètres temporels, mais aussi suivant l'occurence d'évènements aléatoires. Cette approche est plus coûteuse en terme de ressources. Mais elle permet de traiter l'arrivée de tâches apériodiques. La séquence d'exécution est donc construite progressivement.

Le choix d'un algorithme d'ordonnancement est guidé par l'objectif de démonstration du respect des échéances temporelles du système temps réel (cf. sous-section 1.2.5). Ce choix est aussi contraint par les algorithmes disponibles au sein du système d'exploitation, les caractéristiques intrinsèques des tâches et le choix du concepteur.

1.2.3.1 Notion d'optimalité

Pour un ensemble de tâches et un ensemble de contraintes donné (architecture, algorithme d'ordonnancement, etc), si une solution d'ordonnancement existe, et qu'un ordonnanceur temps réel est capable de la trouver, alors cet ordonnanceur est dit **optimal**. *A contrario*, s'il ne trouve pas de solution à cet ensemble, alors aucun autre ordonnanceur ne peut en trouver une.

1.2.3.2 Algorithmes à priorité fixe

Un algorithme d'ordonnancement est dit à **priorité fixe** si toutes les instances d'une même tâche s'exécutent avec une priorité unique. Cette priorité est définie hors ligne, soit suivant des critères de criticité, soit des critères temporels.

Rate Monotonic L'algorithme *Rate Monotonic* (RM) est un ordonnanceur en ligne à priorité fixe proposé par [LL73]. La priorité d'une tâche est inversement proportionnelle à sa période. Ainsi, la tâche la plus prioritaire possède la période la plus petite, et la tâche de plus basse priorité la plus longue. Deux tâches ayant la même période obtiennent la même priorité. Dans ce cas, si les deux tâches sont éligibles au processeur, soit la sélection est arbitraire, soit suivant un sous-algorithme d'ordonnancement [MMG06].

Le tableau 1.1 donne un ensemble de trois tâches à ordonnancer sous *Rate Monotonic*, en mode préemptif. L'attribution des priorités est telle que la tâche τ_1, ayant la plus petite période, possède la plus haute priorité, et la tâche τ_3 possède la plus basse priorité.

La figure 1.5 montre l'échéancier obtenu. Le premier point important est que la période d'étude peut se ramener à 24 unités de temps, puisque à l'instant $t = 24$ la configuration d'arrivée des tâches est la même quà $t = 0$. Le second point important est que seule la tâche τ_3 est préemptée, aux instants $t = 3$ (par la tâche τ_1 puis par τ_2) et $t = 18$ (par la tâche τ_1). Seule la seconde instance de la tâche τ_3, à $t = 10$, peut s'exécuter intégralement sans être interrompue.

	A	T	C	D
τ_1	0	3	1	3
τ_2	0	4	1	4
τ_3	0	8	2	7

TABLE 1.1 – Exemple d'ensemble de tâches

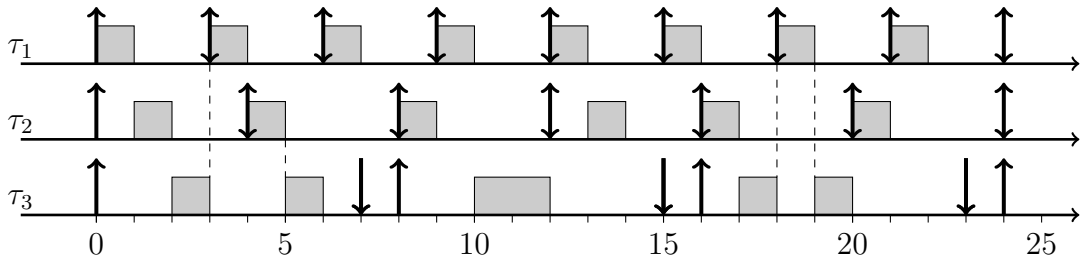


FIGURE 1.5 – Exemple avec Rate Monotonic

Pour un ensemble de tâches périodiques, indépendantes et à échéances sur requêtes, dans un contexte préemptif, cet algorithme est optimal. Du point de vue du concepteur, sous ces conditions, cet algorithme garantit que les tâches dont la fréquence d'activation est la plus élevée ne seront jamais retardées ni préemptées par des tâches de période plus longue.

Deadline Monotonic L'algorithme *Deadline Monotonic*, proposé dans [LW82] permet de traiter les tâches dont les échéances sont inférieures ou égales à leur période ($D_i \leqslant T_i$). Ces travaux étendent ceux de [LL73] en levant la contrainte d'échéance sur requête.

La priorité d'une tâche est inversement proportionnelle à son échéance relative. Ainsi, la tâche de plus haute priorité possède l'échéance relative la plus petite, et la tâche de plus basse priorité la plus grande. Deux tâches ayant la même échéance relative possèdent la même priorité. Dans cette configuration, si les deux tâches sont éligibles au processeur, soit la sélection est arbitraire, soit suivant un sous-algorithme d'ordonnancement.

Le résultat remarquable de ces travaux est l'optimalité du point de vue de l'ordonnançabilité pour des ensembles de tâches périodiques et indépendantes dans un contexte préemptif et de priorités fixes.

1.2.3.3 Algorithmes à priorité dynamique

Un algorithme d'ordonnancement est dit à **priorité dynamique**, si, pour une tâche donnée, la priorité de chaque instance est calculée en ligne. La définition de cette priorité est fonction du contexte d'exécution, et peut être définie suivant des critères temporels ou de criticité.

Earliest Deadline First L'algorithme d'ordonnancement *Earliest Deadline First* (EDF) est issu des travaux de [LL73]. La priorité de chaque instance d'une tâche est inversement proportionnelle à son échéance absolue. Ainsi, lors de l'exécution du système, l'ensemble des tâches éligibles au processeur est trié suivant la valeur de l'échéance absoluse, la tâche la plus prioritaire étant celle dont l'échéance absolue est la plus proche, la moins prioritaire, celle dont l'échéance absolue est la plus tardive. Le réveil de toute instance de tâche impose de recalculer l'ensemble des priorités de l'ensemble de tâches éligibles. L'attribution dynamique de priorités est donc fonction du contexte d'exécution.

Sur la figure 1.6, l'algorithme EDF est appliqué à l'ensemble de tâches du tableau 1.1. La période d'étude est toujours égale à 24. Au départ, la tâche τ_1 a l'échéance la plus proche, et s'exécute donc en premier. Puis l'ordonnanceur élit au processeur la tâche τ_2, puis la tâche τ_3. Les effets d'EDF se voient à partir de l'instant $t = 3$. Une instance de la tâche τ_1 se réveille, et son échéance est la plus proche. La tâche τ_3 est préemptée au profit de τ_1, qui s'exécute. Après sa complétion, une instance de la tâche τ_2 se réveille. L'ordonnanceur peut soit continuer l'exécution de τ_3, soit élire τ_2. Hors comme l'échéance de τ_3 est la plus proche, c'est elle qui est choisie. La tâche τ_2 s'exécutera après la complétion de τ_3. Le reste de l'échéancier est ensuite identique à celui obtenu avec *Rate Monotonic*.

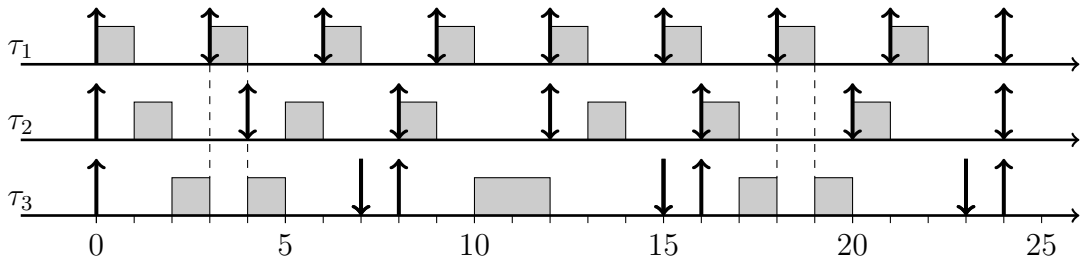


FIGURE 1.6 – Exemple avec Earliest Deadline First

Pour les systèmes monoprocesseur, dans un contexte préemptif, cet algorithme est optimal pour des tâches indépendantes, périodiques et à échéances sur requêtes [Der74]. L'optimalité de l'algorithme a par la suite été démontrée pour des tâches non périodiques. L'avantage de cet algorithme, pour le concepteur, est de garantir que la tâche dont l'échéance est la plus proche sera toujours exécutée, ce qui est un avantage certain dans le traitement des tâches apériodiques correspondant, par exemple, à des alarmes pour le système.

Least Laxity First L'algorithme *Least Laxity First* (LLF) [Mok83] est un autre membre des algorithmes à priorité dynamique. Le critère de laxité, différence entre l'échéance et le temps restant de calcul à l'instant t, permet d'attribuer la priorité d'exécution de la tâche. Ainsi, la tâche dont la laxité est la plus faible est la plus prioritaire, et la tâche de

plus basse priorité, celle dont la laxité est la plus importante.

Le principal inconvénient de LLF réside dans son importante consommation en ressources de calcul. En effet, les priorités doivent être recalculées régulièrement pour prendre en compte l'évolution de la laxité des tâches éligibles. De plus, cet algorithme tend à augmenter le nombre de changements de contexte, source supplémentaire de consommation de ressources. Ainsi, si deux tâches ont la même échéance, LLF affectera à chaque unité de temps une tâche, puis l'autre.

La table 1.2 illustre ce cas de figure. L'ordonnancement par LLF de cet ensemble de deux tâches identiques donne l'échéancier présenté par la figure 1.7.

	A	T	C	D
τ_1	0	10	5	10
τ_2	0	10	5	10

TABLE 1.2 – Ensemble de tâches à ordonnancer par LLF

Puisque les deux tâches se réveillent en même temps, l'ordonnanceur choisit arbitrairement d'exécuter la tâche τ_1. Au bout d'une unité de temps, la laxité de la tâche τ_2 est la plus faible, elle devient la plus prioritaire, et l'ordonnanceur préempte τ_1 pour la laisser s'exécuter. La situation se produit en sens inverse à l'instant $t = 2$, et ainsi de suite. Au bilan, l'échéancier montre huit préemptions et neuf changements de contexte, illustrant par la même l'inconvénient majeur de LLF.

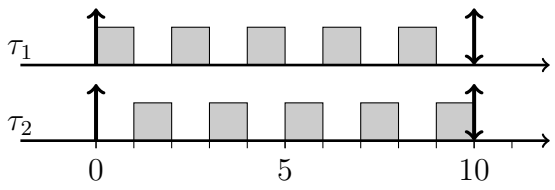


FIGURE 1.7 – Mise en évidence des changements de contexte sous LLF

1.2.4 Gestion des ressources partagées

L'ordonnancement de tâches avec partage de ressources génère deux types de problèmes pour les systèmes temps réel : l'interblocage et l'inversion de priorité. Pour remédier à ces problèmes, des protocoles de partage de ressources sont mis en place.

1.2.4.1 Interblocage

L'accès à plusieurs ressources, partagées entre différentes tâches, peut provoquer une défaillance du système par *interblocage*. Il survient lorsque les différentes requêtes d'accès

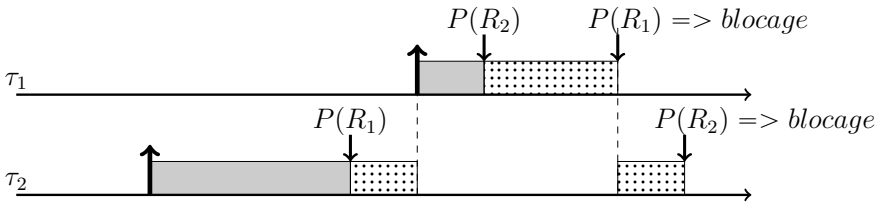


FIGURE 1.8 – Exemple d'interblocage

sont imbriquées.

La figure 1.8 l'illustre. Soit deux tâches τ_1 et τ_2. Elles utilisent deux ressources en exclusion mutuelle R_1 et R_2. τ_1 accède à R_2 puis R_1, à l'inverse de τ_2 qui accède à R_1 puis R_2. La tâche 1 est plus prioritaire, dans un cadre d'ordonnancement en ligne préemptif à priorité fixe. τ_2, alors qu'elle utilise R_1, est préemptée par τ_1. L'exécution de cette tâche est ensuite interrompue, en section critique (ressource R_2), car elle est en attente de R_1. τ_2 reprend son exécution jusqu'à sa demande d'utilisation de R_2, non disponible. Le système se bloque, car les deux tâches sont mises en attente indéfiniment.

1.2.4.2 Inversion de priorité

L'utilisation d'un ordonnancement à priorité fixe et de partage de ressource peut entraîner le phénomène d'*inversion de priorité* [SRL90]. Faute d'accès à une ressource, une tâche est placée en état d'attente par l'ordonnanceur, celui-ci affectant ensuite une tâche de moins haute priorité au processeur.

L'exemple de la figure 1.9 illustre l'inversion de priorité. Soit trois tâches de priorités décroissantes τ_1, τ_2 et τ_3. Les tâches τ_1 et τ_3 partagent une ressource R_1, dont l'accès est en exclusion mutuelle. Soit $P(R_1)$ la requête de ressource et $V(R_1)$ sa libération. La tâche τ_3 commence à s'exécuter. Elle demande l'accès à la ressource, qui lui est accordé, et commence son exécution en section critique. Elle est préemptée par τ_1 qui est éligible. Celle-ci est ensuite bloquée par la demande d'accès à R_1 qui est non disponible. L'ordonnanceur rend alors la main à τ_3. Elle est ensuite préemptée par τ_2, plus prioritaire. La tâche τ_2 s'exécute donc avant τ_1, qui attend la libération de la ressource. Il y a inversion de priorité.

1.2.4.3 Protocole de partages de ressources

Les *protocoles de partage* permettent de pallier ces problèmes par des règles d'allocation de ressources. Sur une tâche donnée, la fraction de C pendant laquelle une ressource est utilisée est appelée **section critique**. Ces protocoles permettent, pour chaque tâche, de calculer les temps de blocage maximaux (notés B_i) induits par l'accès aux sections critiques. Les techniques de validation intègrent ensuite ces termes.

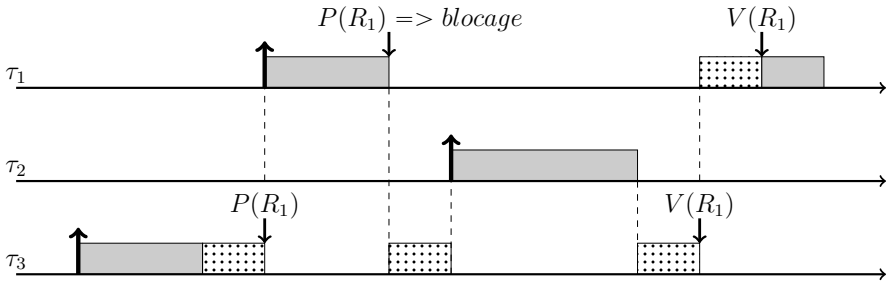


FIGURE 1.9 – Exemple d'inversion de priorité

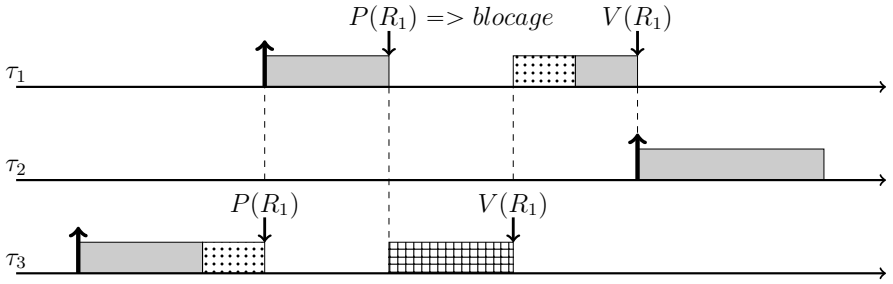


FIGURE 1.10 – Exemple d'utilisation de Priority Inheritance Protocol

Priority Inheritance Protocol Le *Priority Inheritance Protocol* (PIP) est un des principaux protocoles [SRL90]. Si une tâche est en section critique, elle prend la priorité de la tâche la plus prioritaire en attente sur la section critique. On dit qu'elle hérite de la priorité. La figure 1.10 reprend l'exemple décrit pour l'inversion de priorité (figure 1.9). Après avoir préempté la tâche τ_3, la tâche τ_1 est en attente d'accès à la ressource. La tâche τ_3, moins prioritaire, hérite donc de la priorité de τ_1. A la libération de la ressource, τ_1 est élue, et la tâche τ_2 s'exécute ensuite. L'inversion de priorité est levée.

Ce protocole permet donc de borner le temps de blocage des tâches. Il devient possible d'évaluer *a priori* le respect des échéances. Le terme de blocage correspond, pour une tâche donnée, à la section critique maximale parmi toutes les tâches de plus basse priorité avec laquelle elle partage une ressource. Si plusieurs ressources sont partagées, on somme la durée de toutes les sections critiques maximales par ressource.

Autres protocoles Parmi les protocoles, on peut aussi citer *Priority Ceiling Protocol* (PCP) [SRL90]. On alloue à chaque ressource une priorité « seuil » qui correspond à celle de la tâche de plus haute priorité pouvant demander la ressource. Les tâches héritent toujours selon le même mécanisme que PIP. La différence se fait dans le cas de plusieurs ressources. Une tâche ne pourra accéder à une ressource que si sa priorité est supérieure à toutes celles déjà en cours d'utilisation par les autres tâches. On prévient ainsi les risques

d'interblocage. D'autres protocoles, comme *Stack Resources Policy* (SRP) se basent sur des mécanismes de gestion complémentaires [Bak90].

1.2.5 Validation d'un système temps réel

Les techniques de validation issues de la littérature sont variées, et dépendantes du modèle de tâche considéré, de l'ensemble de tâches, du nombre de processeurs et de l'algorithme d'ordonnancement. Ainsi, un ensemble de tâches sous un algorithme d'ordonnancement donné est dit **faisable** s'il existe au moins une technique de validation démontrant *a priori* le respect de toutes les échéances temporelles. Pour autant, une ou plusieurs tâches peuvent être **ordonnançables** sans que l'ensemble soit faisable. La faisabilité d'un ensemble de tâches peut alors se définir comme suit :

Définition 1 *Un ensemble de tâches* Γ *est faisable ssi il existe un ordonnancement capable d'ordonnancer toutes les tâches* τ_i

Les différentes techniques de validation présentent soit une validation **globale** de l'ensemble de tâches, soit une validation **locale**. Ainsi, une validation globale donne une réponse sur la faisabilité d'un ensemble de tâches. Dans le cas d'une validation locale, la réponse porte sur l'ordonnançabilité d'une tâche. La faisabilité de l'ensemble de tâches est alors déduite de la définition 1.

Dans de précédents travaux [MD08], la complémentarité des différentes techniques a été mise en valeur. Ainsi, l'identification d'une technique à même de valider un ensemble de tâches apporte de l'information sur le système temps réel. Si plusieurs techniques concluent à la faisabilité, les informations déduites sont complémentaires.

L'objectif de cette sous-section est de présenter les grandes familles de techniques de validation, ainsi que leurs principes sous-jacents.

1.2.5.1 Instant critique : une notion essentielle

Pour un ensemble de tâches Γ, dans un système temps réel monoprocesseur, l'instant critique se définit comme l'instant conduisant au pire scénario. En d'autres termes, pour un système préemptif, l'instant critique de la tâche τ_i correspond à l'instant t tel que les interférences dues aux tâches de plus haute priorité sont maximales.

L'instant critique correspond donc à un instant tel que toutes les tâches de plus haute priorité et la tâche elle-même sont réveillées [LL73]. Par ailleurs, depuis cet instant, la période d'étude est réduite à la notion d'hyperpériode H, qui correspond au plus grand commun multiple des périodes de l'ensemble de tâches.

1.2.5.2 Borne du taux d'utilisation

Pour un ensemble de tâches Γ, le taux d'utilisation du processeur représente la somme des contributions $\frac{C_i}{T_i}$ de chaque tâche dans l'utilisation du processeur (Equation 1.1). Sur la base de cet indicateur de charge du processeur, plusieurs travaux ont développé des bornes garantissant la faisabilité des ensembles de tâches si elles sont respectées.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{1.1}$$

Ainsi, pour un ensemble de tâches indépendantes, périodiques à échéances sur requêtes et ordonnancé sous RM, les travaux de [LL73] proposent l'existence d'une borne dépendante du nombre n de tâches (Equation 1.2). Pour un nombre élevé de tâches, cette borne converge vers une valeur de taux d'utilisation de 69%. Cette condition est suffisante, si bien que certains ensembles de tâches peuvent être faisables pour des taux supérieurs. Cet état de fait montre que l'évaluation de cette borne est **pessimiste**, au sens où la non satisfaction de la borne ne permet pas de conclure sur l'ordonnançabilité de l'ensemble de tâches. Les travaux de [LW82] étendent cette borne à l'algorithme d'assignation de priorité DM, en substituant dans l'expression du taux d'utilisation D_i à T_i.

$$U \leqslant n \times (\sqrt[n]{2} - 1) \tag{1.2}$$

Par ailleurs, les travaux de [LL73] démontrent, sous les mêmes hypothèses que pour RM (tâches indépendantes et à échéances sur requêtes, prises à l'instant critique), qu'un ensemble Γ de tâches sous EDF est faisable si le taux d'utilisation est inférieur à 100%. De plus, EDF est optimal sous ces hypothèses [Der74], *i.e* s'il existe un ordonnancement tel que Γ soit faisable, alors Γ est aussi faisable sous EDF.

1.2.5.3 Évaluation du temps de réponse

Le temps de réponse d'une tâche représente la durée écoulée entre le réveil de la tâche et sa complétion effective. Dans le cadre de la technique de validation, dite de calcul de pire temps de réponse (*Worst Case Response Time - WCRT*), l'objectif est d'évaluer cette durée dans le pire scénario. Ce pire scénario est propre à chaque tâche. Ainsi, si ce pire temps de réponse est inférieur ou égal à l'échéance relative de la tâche ($\mathcal{R}_i \leqslant D_i$), la tâche est ordonnançable. Sinon, la tâche n'est pas ordonnançable.

Pour une tâche donnée, dans un système temps réel concurrentiel préemptif à priorités fixes, l'équation de temps de réponse s'exprime comme la somme du pire temps d'exécution C_i de la tâche, du terme d'interférences I_i dû aux tâches de plus haute priorité, et d'un éventuel terme B_i si la tâche partage des ressources (Equation 1.3) [Aud+91 ; JP86]. Le terme B_i dépend du protocole de partage de ressource, et représente le temps de blocage maximum de la ressource.

$$\mathcal{R}_i = C_i + I_i + B_i \tag{1.3}$$

Par construction, l'équation 1.3 évalue le pire temps de réponse de la tâche τ_i. En effet, le terme d'interférences I_i représente la durée cumulée des préemptions et des délais dus aux tâches de plus haute priorité. Pour évaluer ce nombre de préemptions, l'opérateur $\lceil\ \rceil$ est utilisé. Celui-ci retourne le plus grand entier naturel supérieur ou égal à la valeur de sa variable. En l'occurence, ici, le nombre de réveils d'une tâche de plus haute priorité au cours du temps de réponse de la tâche τ_i. Pour chacune de ces préemptions, dans le pire cas, le pire temps d'exécution C_j est atteint (Equation 1.4).

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{\mathcal{R}_i}{T_j} \right\rceil C_j \tag{1.4}$$

Dans l'expression de l'équation du pire temps de réponse, le terme $\mathcal{R}_i$ apparaît de chaque côté. L'équation 1.3 se résout donc par itération, suivant le théorème du point fixe 1.5. La convergence est assurée si $U \leqslant 1$, c'est à dire si le système temps réel n'est pas en surcharge.

$$\mathcal{R}_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\mathcal{R}_i^n}{T_j} \right\rceil C_j + B_i \tag{1.5}$$

De la convergence du point fixe ($\mathcal{R}_i^n = \mathcal{R}_i^{n+1}$) se déduit la valeur du pire temps de réponse $\mathcal{R}_i$. Ainsi, si cette valeur est inférieure à l'échéance D_i, la tâche est ordonnançable. Dans le cas contraire, la non ordonnançabilité de la tâche entraîne la non faisabilité de l'ensemble de tâches. Par ailleurs, cette technique de validation s'opérant par ordre décroissant des priorités, si l'ensemble de tâches est non faisable, il est possible d'en extraire un sous-ensemble de tâches faisable.

L'application du pire temps de réponse au cas des ordonnancements à priorité dynamique a été étudiée dans le cadre d'EDF par [Spu96a ; Spu96b]. Outre la méthode d'évaluation de $\mathcal{R}_i$, le résultat remarquable de ces travaux est la démonstration que le pire temps de réponse n'est pas nécessairement atteint depuis l'instant critique, ce qui impose d'étudier plusieurs périodes d'exécution.

1.2.5.4 Demande d'utilisation du processeur

Dans l'approche par demande d'utilisation, une fonction W calcule à l'instant t le temps d'exécution cumulé par les différentes instances de tâches. Pour un ensemble Γ de tâches indépendantes à échéances sur requêtes et un ordonnancement à priorité fixe, la technique de validation proposée par [LSD89] suit le raisonnement suivant : la tâche de plus haute priorité i voit sa première instance complétée avant son échéance s'il existe $t \in [0; T_i]$ telle que $W_i(t)$ soit inférieure ou égale à t (Equation 1.6).

$$W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leqslant t \tag{1.6}$$

Puisque $\forall i, \frac{t}{T_i}$ est strictement croissant, l'opération $\left\lceil \frac{t}{T_i} \right\rceil$ change de valeur lors du réveil de la tâche i. Ainsi, l'ensemble des valeurs de t à tester sont les multiples des dates de réveil $(kT_i, k \in \mathcal{N})$ des différentes tâches de l'ensemble Γ. Les travaux de [NB91] étendent cette méthode aux échéances inférieures ou égales à la période.

L'approche par demande d'utilisation du processeur est aussi utilisée dans le cadre d'EDF. Les travaux [BMR90 ; BRH90], suivant diverses hypothèses sur l'ensemble de tâches Γ, donnent le résultat fondamental suivant. Puisque la demande d'utilisation du processeur dans l'intervalle $[t_1, t_2]$ represente le cumul des demandes $W(t_1, t_2)$ dues aux instances des tâches actives sur $[t_1, t_2]$, ces instances de tâches doivent être complétées sur $[t_1, t_2]$.

Ainsi, un ensemble de tâches Γ est faisable sur tout intervalle de temps si et seulement si la demande totale d'utilisation de processeur sur cet intervalle ne dépasse pas le temps disponible (Equation 1.7).

$$\forall (t_1, t_2),\ W_i(t_{1,t_2}) \leqslant (t_2 - t_1) \tag{1.7}$$

1.2.6 Cheddar : outil de validation et de simulation [Sin+04]

Cheddar est logiciel à visée universitaire, dédié aux analyses d'ordonnançabilité et aux simulations d'ordonnancement. Écrit en ADA, il permet de modéliser des ensembles de tâches indépendantes, à contraintes de précédence, communicant par messages et partageant des ressources via des mémoires tampon. L'architecture du système temps réel peut être monoprocesseur, multiprocesseur ou distribuée. Le cœur du logiciel propose les algorithmes d'ordonnancement classiques décrits dans l'état de l'art de ce manuscrit.

Ce logiciel propose aux utilisateurs une méthode pour implanter de nouveaux ordonnanceurs et de nouveaux tests de faisabilité. Cependant, les comportements liés à la détection de fautes et à leur correction, ainsi que l'implantation des stratégies ne font pas partie des possibilités offertes à l'utilisateur. Dans le cadre de ces travaux, un logiciel dédié a été développé pour permettre d'évaluer la pertinence des résultats qui vont être décrits. L'interface fournie par Cheddar permettant à l'utilisateur de définir son propre ordonnancement s'est révélée insuffisante pour implanter les problématiques décrites.

1.3 Conclusion

L'objectif de cette introduction aux systèmes temps réel est de donner au lecteur les clés de compréhension du manuscrit. La spécificité du domaine qui sera décrit dès la partie suivante - les ordonnancements tolérants aux fautes - impose de détenir un ensemble de connaissances générales sur les ordonnancements, leurs contraintes et leur validation.

Les lecteurs les plus avertis remarqueront que l'ensemble des techniques de validation présentées sont par nature analytiques et statiques. Des techniques basées, par exemple,

sur le *model checking*, ne sont pas décrites. Ce choix de non exhaustivité du domaine se justifie du fait que l'état de l'art central sur lequel se positionne ces travaux de thèse est décrit dans le chapitre 2.

Les techniques de validation qui seront décrites dans le chapitre 2 sont toutes de nature analytiques et statiques, et basées sur les connaissances générales décrites dans ce chapitre. Par la suite, ces travaux de thèse seront aussi de cette nature. Le choix de conception de ce chapitre vers un contenu autoporteur pour la compréhension du chapitre 2 a donc prévalu à une exhaustivité de la validation des ordonnancements temps réel.

Deuxième partie

Ordonnancement tolérant aux fautes

Chapitre 2

Etat de l'art

Sommaire

Cette partie du manuscrit, consacrée aux **ordonnancements temps réel tolérants aux fautes**, aborde le problème de l'impact des mécanismes de tolérance aux fautes sur les ordonnancements temps réel. Cet impact est une déclinaison au niveau système de l'étude de l'impact de la sûreté de fonctionnement sur les aspects temps réel. La présence d'erreurs dans une ou plusieurs tâches temps réel n'est pas acceptable au sens où même si les échéances temporelles sont respectées, la diffusion d'erreurs dans le système peut conduire à des évènements catastrophiques. Ainsi, pour des raisons de fiabilité, la présence de mécanismes de gestion d'erreurs est essentielle.

Pour évaluer l'impact des mécanismes de tolérances aux fautes, la suite d'évènements suivante doit être considérée : occurence d'une faute dans le système temps réel révélée par la présence d'une erreur dans une tâche, puis détection de l'erreur et correction. Une technique à même de valider un ordonnancement tolérant aux fautes doit donc intégrer le surcoût temporel de cette suite d'évènements. De plus, dans le cadre d'une validation hors ligne, la distribution temporelle des fautes est une entrée du problème.

Ce chapitre est dédié à l'état de l'art des ordonnancements tolérants aux fautes, et propose au lecteur les clés de compréhension du domaine. Il se structure comme suit. La

section 2.1 décrit les différents modèles de distribution de fautes qui précèdent ces travaux de thèse. La section 2.2 détaille les principaux mécanismes de détection et de correction d'erreurs applicables aux systèmes temps réel. Puis la section 2.3 présente les travaux majeurs portant sur les ordonnancements tolérants aux fautes, classés suivant le modèle de fautes pris en compte. Enfin la section 2.4 conclut ce chapitre.

2.1 Modèle de distribution de fautes

En dehors des problématiques liées aux fautes permanentes, l'étude de la tolérance aux fautes transitoires des systèmes temps réel repose essentiellement sur trois modèles de distribution temporelle de fautes : le modèle de fautes pseudo-périodiques, le modèle en « motif de fautes » et l'approche stochastique.

Les sous-sections suivantes décrivent ces trois modèles de fautes antérieurs à celui proposé dans ces travaux. Le modèle de fautes pseudo-périodiques traduit des fautes transitoires séparées par une pseudo-période connue. Dans le cadre des motifs de fautes, l'accent est mis sur le caractère discrétisable du nombre de fautes, et sur l'exploration des différentes combinaisons possibles. Enfin, l'approche stochastique considère l'occurrence de fautes comme un évènement aléatoire.

2.1.1 Modèle de fautes pseudo-périodiques

Dans un modèle de fautes pseudo-périodiques, chaque faute transitoire est séparée de la suivante par un intervalle minimum de temps T_F (Figure 2.1). Une faute est vue comme un évènement ponctuel et isolé des fautes suivantes. Ce modèle permet de décrire par construction des phénomènes affectant le système de manière pseudo-périodique, la pseudo-période étant connue.

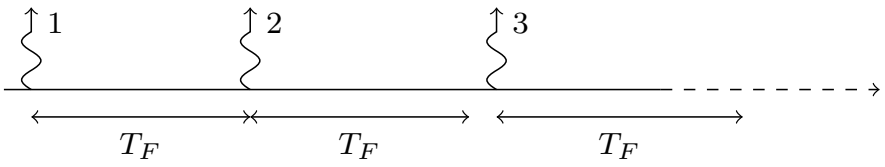


FIGURE 2.1 – Modèle de fautes pseudo-périodiques

Le modèle est applicable aux problématiques de type compatibilité électromagnétique. Ainsi la proximité d'un calculateur et d'une alimentation provoque des fautes sur le calculateur de manière périodique. En effet, le champ généré par la source de courant perturbe le matériel suivant la phase du champ, phase qui est périodique.

.

2.1.2 Approche par motifs de fautes

L'hypothèse initiale dans la définition d'un motif de fautes (*fault pattern*) est la présence de k fautes perturbant le système. Les fautes considérées sont transitoires, et peuvent éventuellement intervenir sur un intervalle de temps borné T_{max} [Ayd07 ; Pat10]. Un motif de fautes représente le placement temporel de ces k fautes. Dans les travaux de [Ayd07 ; LMM00 ; LB05 ; Pat10] l'idée générale est l'étude de l'impact de ces motifs de fautes sur le système temps réel considéré.

Ces différents travaux considèrent soit que le motif est connu ou identifiable, soit que le motif est quelconque. Dans le premier cas, l'étude se réduit à une étude directe de l'impact des fautes sur le système. Le second cas est plus complexe. Généralement, pour k fautes données, l'exploration des différentes combinaisons de fautes conduit à définir plusieurs motifs, et à en étudier les effets. L'effort se porte alors sur la complexité des algorithmes d'exploration, l'objectif étant de trouver les motifs de fautes maximisant les effets néfastes sur le système temps réel.

2.1.3 Approche stochastique

Les deux approches précédemment décrites font l'hypothèse que certaines caractéristiques des fautes sont connues (pseudo-période, nombre de fautes, intervalle d'occurence de fautes). L'approche stochastique diffère en considérant que par essence, l'occurence de fautes au sein d'un système temps réel est aléatoire, et doit donc être modélisée au travers de lois probabilistes.

L'approche stochastique s'appuie sur les processus de poisson généralisés, dont la loi probabiliste est donnée par l'équation (2.1), qui donne le nombre de fautes k sur l'intervalle $[0; t)$. Le paramètre λ représente la densité moyenne d'occurence des fautes par unité de temps t.

$$P_k(t) = e^{-\lambda t}\frac{(\lambda t)^k}{k!} \tag{2.1}$$

Les travaux de [Pun97], pour les études d'ordonnancement tolérant aux fautes, et [NSS00], pour la fiabilité des transmissions sur des bus CAN, appliquent cette approche. L'évaluation du terme λ est faite suivant à la fois la connaissance de l'environnement d'évolution du système temps réel et les défaillances connues des composants du système. Dans ce dernier cas, l'évaluation du temps moyen entre deux défaillances (*Mean Time Between Failures* - MTBF) conduit à la définition du terme $\lambda = \frac{1}{MTBF}$.

2.2 Se protéger des fautes et gérer les erreurs

Ces travaux de thèse s'appuient sur l'hypothèse que le système temps réel considéré fournit des méthodes de détection et de correction d'erreurs. De fait, la question de la

détection et de la correction d'erreurs est une entrée de la problématique, sans en être le cœur. Cette section présente l'état de l'art du domaine, fournissant par là même au lecteur les éléments de compréhension suffisants pour appréhender la problématique.

La section se structure en deux parties. Dans la première, la question de la tolérance aux fautes matérielles est abordée. Puis dans la seconde, la tolérance aux fautes logicielles est présentée.

2.2.1 Tolérance aux fautes matérielles

La tolérance aux fautes matérielles peut être native et implantée directement dans le matériel, ou concerner une couche logicielle du système d'exploitation. D'une manière générale, la tolérance aux fautes matérielles est faite de telle façon que la faute reste masquée. L'incapacité du mécanisme mis en place à isoler et à traiter l'erreur détectée conduit soit à une faute permanente, soit à un redémarrage du matériel concerné. Dans le cas des fautes permanentes, seule la redondance spatiale permet de traiter le problème.

L'architecture de von Neumann (décomposition d'un ordinateur en flot de contrôle, flot de données, unité de calcul et mémoire) facilite la description des mécanismes de tolérance aux fautes matérielles. Ainsi, le graphe de flot de contrôle statique, issu du programme binaire en mémoire, peut être comparé au graphe de flot de contrôle obtenu pendant l'exécution. Toute incohérence conduit à la détection d'erreur [DS91 ; WH90]. De la même façon, la validité des données traversant le graphe de flot de données, reconstruit à l'exécution, peut être comparé à celui statique issu du binaire [MS07]. La correction peut-être effectuée via des codes correcteur d'erreur, moyennant un dépassement temporel fonction de la redondance d'information ajoutée dans la structure de donnée.

Au niveau de l'unité de calcul, des méthodes utilisent une transformation initiale du programme en mémoire (multiplication des variables par un facteur de diversité k par exemple) et comparent l'écart entre les valeurs initiales et les valeurs produites (moyennant le facteur k) [OSM02]. L'ensemble des principes décrits ci-dessus sont par exemple intégrés au sein d'ARGUS, qui est une proposition d'implantation des divers mécanismes de détection d'erreurs pour les processeurs à simple cœur. Cette approche s'est concrétisée par un premier prototype nommé ARGUS-1 [MBS07].

Enfin, au niveau de la mémoire, des mécanismes à base de bits de controle, pour la détection (et la correction) d'erreurs sont utilisés, ainsi que la comparaison d'information redondées [Lee+11]. La duplication en mémoire des informations est aussi utilisée nativement dans le matériel moderne.

2.2.2 Tolérance aux fautes logicielles

La notion de fautes et d'erreurs logicielles est à considérer dans le contexte d'absence de réalité physique du logiciel. Ainsi, les erreurs logicielles sont perçues comme des erreurs de

conception, introduites durant le développement ([RE92 ; RE11 ; Tor00]). La gestion de ces erreurs peut suivre plusieurs approches, décrites ci-après. La question de la méthodologie de détection d'erreurs est abordée à la lumière des mécanismes de recouvrement considérés.

Recovery Block L'approche par *Recovery Block* [Hor+74] suppose que le programme est découpé en module primaire (*Primary Block*) et alternatifs (*Alternate Blocks*). Ainsi, le bloc primaire contient l'algorithme préférentiel à exécuter. A la fin du bloc, un test d'acceptance est exécuté. Si le test n'est pas satisfait, alors un bloc alternatif est exécuté, puis de nouveau le test d'acceptance, et ainsi de suite. Si tous les blocs alternatifs échouent, alors une défaillance du système est signalée.

Dans cette approche, le mécanisme de détection n'est pas à proprement parlé explicité, *a contrario* des types d'erreurs : échec du test d'acceptance, *timeout* et erreur internes (division par zéro, etc...). De fait, certains mécanismes d'exception peuvent être associés à ces types d'erreurs. Ainsi, l'ajout d'assertions dans le code, décrivant par exemple les exigences sur une variable, permet de détecter une erreur lorsque la sortie calculée est hors de son intervalle [EAM85]. Les exigences temporelles, pouvant conduire à un *timeout*, sont implantables par l'utilisation de *watchdog timer* [Mir+92]. Tout dépassement du budget temporel entraîne la détection d'une erreur.

L'utilisation de *Recovery Block* permet aussi d'intégrer aux blocs alternatifs des algorithmes plus rapides à calculer, au prix d'une précision moindre. L'idée sous-jacente est d'obtenir, dans la contrainte temporelle donnée, un résultat exploitable par le système temps réel, plutôt que de risquer un dépassement d'échéances et l'absence de données [LNL87].

N-versions d'un même programme La tolérance aux fautes par N-versions d'un même programme appartient à la catégorie des redondances temporelles. Dans les travaux de [Avi85], la ligne directrice est de décliner un même programme en $N \geqslant 2$ versions indépendantes. La condition de réussite est le développement indépendant des différentes versions. La notion d'indépendance évoquée concerne autant les équipes de programmeurs, que les langages et algorithmes utilisés.

Lorsque les N-versions d'un même programme sont exécutées parallèlement, les sorties peuvent être comparées. L'inconvénient réside initialement dans les différents temps de calcul des versions. Ces différences imposent la définion d'un point de rendez-vous déterministe pour la comparaison. L'autre remarque porte sur le nombre de versions de programme nécessaires pour, par exemple, une comparaison par voteur. Si $N = 2$, la détection d'erreur conduit nécessairement à des actions correctrices sur les deux versions, car dans ce cas il est imposssible de connaître quelle version est fautive. Par ailleurs, l'application des N-versions peut se faire séquentiellement, mais au prix d'un coût temporel très important.

Mécanismes d'exception A un certain niveau d'abstraction, les mécanismes d'exception ont le même comportement que les *recovery blocks* [Goo75 ; MR77]. Lors de la détection d'une erreur, une exception est levée et conduit à une action correctrice implantée par le développeur. La différence s'opère principalement dans le ciblage de l'action correctrice.

Ainsi, là où un test d'acceptance est utilisé dans le cadre de l'approche par *recovery blocks*, l'exception est typée et propre à l'erreur [MR77]. L'avantage de ce typage est donc de particulariser l'action correctrice à entreprendre. Si au départ les mécanismes d'exception sont plutôt liées aux langages objets, des travaux existent pour l'implanter dans les langages comme le C [BVDT06].

Checkpointing Le mécanisme de checkpointing répond au besoin de réactivité du système en présence de fautes. Ainsi, une tâche informatique τ_i est subdivisée en m_i secteurs de longueur δ_i. Le passage entre chaque secteur est soumis à un test d'acceptance [Bro79]. Le temps de d'exécution de ce test est noté O_A. Lorsque le test réussit, l'état courant de la tâche (variables) est sauvegardé dans une zone mémoire non corruptible, définissant ainsi un *checkpoint*. Le temps de sauvegarde est noté O_S. Ces deux actions ajoutent donc au temps d'exécution de la tâche un surcoût temporel noté $O = O_A + O_S$, modifiant ainsi l'expression du pire temps d'exécution, noté $C_i^{m_i}$ (équation (2.2)). Pour des raisons de simplicité, quelle que soit la tâche, la valeur du surcoût temporel est considérée comme unique.

$$C_i^{m_i} = C_i + m_i \times O \tag{2.2}$$

Le temps d'exécution est augmenté de $m_i \times O$. En effet, $m_i - 1$ checkpoints existent entre les m_i secteurs (figure 2.2). De plus, une sauvegarde est effectuée avant le début d'exécution du secteur δ_1 et un test d'acceptance est exécuté à la fin du secteur δ_m, formant ainsi temporellement un surcoût de O. Ce dernier surcoût n'apparaît pas sur la figure 2.2 pour des raisons de lisibilité.

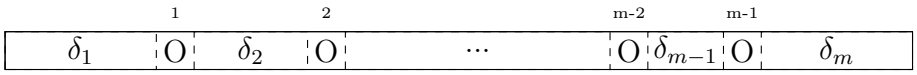


FIGURE 2.2 – Illustration du mécanisme de checkpointing

Lorsque le test échoue, le secteur qui a échoué est réexécuté depuis le dernier checkpoint validé [PBD01]. Pour cela, l'état correspondant au dernier checkpoint est restauré, le coût de la restauration étant habituellement considéré comme négligeable [Pun97]. Ainsi, en considérant la figure 2.2, si le test d'acceptance du secteur $m-1$ échoue, la réexécution s'opérera depuis le checkpoint $m-2$. Le coût de cette réexécution sera de $\delta_{m-1} + O_A$.

Deux problématiques sont inhérentes au checkpointing : le nombre et le placement des checkpoints. Si l'augmentation du nombre de checkpoint améliore d'une part la réactivité

du système, elle dégrade d'autre part le temps d'exécution de la tâche, en multipliant les sauvegardes d'état. De nombreux travaux s'intéressent à cette problématique [Dud83 ; Gel79].

Par ailleurs, un placement judicieux des checkpoints est nécessaire. Les divers travaux proposent de rechercher les placements minimisant le dépassement temporel dû aux sauvegardes d'état [CR72 ; LF90 ; TB84 ; ZB97]. La question de la transformation de programme permettant l'insertion des checkpoints dans la tâche est aussi abordée [AFG06].

2.3 Analyse d'ordonnançabilité et tolérance aux fautes

Les travaux portant sur l'intégration de la tolérance aux fautes dans les analyses d'ordonnançabilité sont nombreux. D'une manière générale, ces travaux étendent les techniques de validation issues de la théorie de l'ordonnancement et présentées dans la section 1.2. Ces extensions intègrent le modèle distribution de fautes étudié et un ou plusieurs mécanismes de gestion d'erreurs au sein des tâches.

L'état de l'art des principaux travaux et résultats du domaine dépeint ci-après est trié suivant le modèle de distribution de fautes, puis affiné en fonction des techniques de validation. Ainsi, cette section se structure comme suit. La sous-section 2.3.1 donne les principes généraux du traitement des fautes permanentes. Les sous-sections 2.3.2 et 2.3.3 fournissent les résultats obtenus autour des modèles de fautes pseudo-périodiques et en motif de fautes. Enfin, la sous-section 2.3.4 présente les principaux travaux sur les ordonnancements tolérants aux fautes « par construction », *i.e* sans considération de distribution de fautes.

2.3.1 Tolérer les fautes permanentes

Les fautes permanentes se traduisent généralement par la défaillance d'un ou de plusieurs processeurs. Les approches traditionnelles se fondent sur le mécanisme de copie primaire (*Primary*) et de sauvegarde (*Backup*). Ainsi, pour chaque tâche, une copie primaire et une copie secondaire sont considérées, chaque copie étant affectée sur un processeur différent [BMR99 ; KS86 ; PM98].

S'il est nécessaire de prouver que pour chaque processeur considéré, les copies primaires et de sauvegarde allouées sont ordonnançables, les techniques de validation associées n'intègrent pas directement la notion de tolérance aux fautes. Seules les redondances spatiales permettent de gérer efficacement les fautes permanentes, dans le cadre de systèmes à multiprocesseurs ou distribués. Dès lors, bien que faisant partie de la théorie des ordonnancements tolérants aux fautes, ces travaux sortent du périmètre d'étude de ces travaux de thèse.

2.3.2 Tolérer les fautes pseudo-périodiques

Les analyses d'ordonnançabilité en présence de fautes pseudo-périodiques sont nombreuses et fondées pour la plupart sur l'extension des techniques de validation présentées dans la section 1.1. Cette sous-section présente les principaux travaux précédents et leurs résultats, suivant la technique de validation utilisée.

2.3.2.1 Approche par file d'attente

Initialement, les travaux de [GMM95] prennent en compte des fautes de pseudo-période $T_F \geqslant 2 \times \max(C_i)$. Chaque tâche échoue au plus une fois. L'ordonnancement est une file d'attente de n tâches apériodiques indépendantes dont les dates de réveil et les échéances sont connues. Un algorithme insère entre les tâches une sauvegarde telle que toute tâche erronée puisse être complètement réexécutée avant son échéance. La détection d'erreur intervient à la fin de la tâche. La complexité de l'algorithme, initialement en $O(n^2)$, a été amélioré en $O(n)$ [CHS09]. L'algorithme de [GMM95] est implanté dans le système d'exploitation FT-RT-Mach [MMG03].

2.3.2.2 Borne du taux d'utilisation du processeur

Les travaux de [PM98] considèrent un ensemble de tâches périodiques indépendantes ordonnancé par RM. La technique de validation est dérivée de la borne de [LL73] pour prendre en compte des fautes de pseudo-période $T_F \geqslant \max(T_i)$. Ces travaux démontrent qu'aucune tâche ne dépassera son échéance si le taux d'utilisation du processeur reste inférieur ou égal à 50%.

Ce résultat est remarquable car contre-intuitif. La borne de [LL73] converge vers 69%. Dans le pire cas toutes les tâches sont erronées. Chaque tâche s'exécute donc pendant C_i unités de temps, et nécessite C_i unités de temps supplémentaires pour être corrigée. En conséquence, l'intuition conduit à diviser le taux d'utilisation sans fautes de l'ensemble de tâches par deux, soit $34,5\%$, ce qui est contredit par les travaux de [PM98].

La borne représentée par les 50% de taux d'utilisation du processeur est une condition suffisante. Comme le montreront les résultats de l'étude quantitative présentés dans la section 4.3, ces travaux donnent une technique de validation permettant de conclure à la faisabilité d'ensemble de tâches dont le taux d'utilisation est supérieur à cette borne.

2.3.2.3 Evaluation du pire temps de réponse

Les précédents travaux contraignent la valeur de la pseudo-période T_F. Dans l'approche suivie par [BDP96], cette pseudo-période devient une donnée d'entrée du problème. Cette approche dérive l'équation du pire temps de réponse sans fautes (Equation 1.3) par l'ajout d'un terme F_i, correspondant au surcoût temporel dû au mécanisme de gestion d'erreurs

(équation (2.3)).

$$\mathcal{R}_i^{T_F} = C_i + B_i + I_i + F_i \tag{2.3}$$

Cas d'une détection à la fin de la tâche Dans les travaux de [BDP96], la détection d'erreurs est supposée intervenir à la fin de la tâche, et la méthode de correction d'erreurs peut être au choix une réexécution complète, un mécanisme d'exception ou un *recovery block*. L'équation (2.4) donne l'expression générale du terme F_i.

$$F_i = \left\lceil \frac{\mathcal{R}_i^{T_F}}{T_F} \right\rceil \max_{j \in hp(i) \cup \{i\}} (C_j^F) \tag{2.4}$$

Ainsi, le terme $\max(C_j^F)$ représente le surcoût temporel dû à la plus longue exécution d'une méthode de correction d'erreur parmi $j \in hp(i) \cup \{i\}$. La valeur de C_j^F dépend de la méthode de correction d'erreurs. Enfin, le terme $\left\lceil \frac{\mathcal{R}_i^{T_F}}{T_F} \right\rceil$ évalue le nombre d'erreurs à prendre en compte dans le calcul du pire temps de réponse.

Les résultats de ces travaux ont par la suite été améliorés par [LB03] par l'introduction de priorités variables pour les méthodes de correction d'erreurs. L'effort est alors porté sur la définition d'une configuration de priorités garantissant à la fois la correction des tâches erronées, et le non dépassement d'échéances temporelles.

Cas du checkpointing Cette sous-section serait incomplète sans mentionner l'évaluation du pire temps de réponse dans le cadre d'une détection et correction d'erreurs par un mécanisme de checkpointing. Ces travaux, menés par [Pun97], aboutissent à une formule générale présentée dans l'équation (2.5).

$$\mathcal{R}_i^{T_F} = C_i^{m_i} + B_i + \sum_{j \in hp(i)} \left\lceil \frac{\mathcal{R}_i^{T_F}}{T_j} \right\rceil C_j^{m_j} + \left\lceil \frac{\mathcal{R}_i^{T_F}}{T_F} \right\rceil (\widehat{C_j} + O) \tag{2.5}$$

Le raisonnement de construction de l'équation est similaire à celui suivi dans l'établissement de l'équation (2.3). Ici, le terme $\widehat{C_j}$ correspond, parmi la tâche τ_i et les tâches de plus haute priorité, au plus long temps de réexécution d'un secteur, soit $\widehat{C_j} = \max(\delta_{j,k})$.

2.3.3 Tolérer les motifs de fautes

L'intégration de motifs de fautes au sein des analyses d'ordonnançabilité commence avec les travaux de [LMM00] sur des ensembles de tâches apériodiques ordonnancés suivant EDF. La faisabilité des ensembles de n tâches est prouvée par la dérivation des techniques de validation basées sur la demande du processeur, dans le cadre de détection d'erreurs à la fin de l'exécution de la tâche et de méthodes de correction d'erreurs basées sur les

recovery blocks. Ces travaux proposent à la fois, si le motif est déjà connu, une condition nécessaire et suffisante, et s'il est inconnu, une méthode d'exploration des combinaisons de fautes de complexité $O(n^2)$.

Ces travaux initiaux sont ensuite étendus par [Ayd07] pour des ensembles de tâches périodiques. Dans le cas d'un motif de fautes quelconque, un algorithme d'exploration de scénarios, basé sur l'intégration progressive de nouvelles tâches dans un ensemble de tâches déjà validées, est proposé. La complexité de cet algorithme est $O(n^2k^2)$ où n est le nombre de tâches et k le nombre de fautes du motif. Cette approche est dérivée dans [Pat10] pour prendre en compte *Rate Monotonic*. L'algorithme de résolution proposé possède une complexité en $O(nNk^2)$ où N est le nombre maximal d'instances de n tâches réveillées au cours de T_{max} , k étant le nombre de fautes sur T_{max}.

Les travaux de [LB05] déclinent la méthodologie suivie dans [LB03] dans le cadre des ordonnancements à priorités fixes soumis à des motifs de fautes. Ainsi, les méthodes de correction possèdent une priorité pouvant être plus élevée que les instances de tâches. La technique de validation, basée sur l'évaluation du pire temps de réponse, est combinée avec un algorithme qui recherche la combinaison de fautes la plus défavorable pour le système. Le nombre de fautes, noté N_E est considéré comme une entrée du problème.

2.3.4 Tolérer des fautes par construction

La tolérance aux fautes « par construction »est une approche visant à rendre tolérant aux fautes un ensemble de tâches sans considérer de modèle de fautes particulier. La problématique est alors l'optimisation de la laxité disponible dans le système pour autoriser un certain nombre de corrections de tâches erronées.

Ainsi dans [Gho+98], les travaux proposés dérivent la borne de [LL73]. Cette borne sert à valider les schémas de recouvrement et d'utilisation de laxité définis dans [GMM95]. La méthode de recouvrement mise en place est une réexécution complète de la tâche erronée. Le schéma d'utilisation de la laxité couplé à la dérivation de borne permet d'assurer à chaque tâche une réexécution, en garantissant le non dépassement d'échéances, et indépendamment d'un modèle de fautes prédéfini.

Les travaux de [HSW03] proposent de rendre un ordonnancement tolérant aux fautes suivant la technique de la dernière chance développée par [CC89]. Chaque tâche possède une copie primaire, et une copie secondaire plus rapide en terme de temps d'exécution, mais moins précise en terme de résultats. La ligne directrice est, sachant que la séquence des copies primaires sur l'hyperpériode est connue, de déterminer l'instant au plus tard de démarrage des copies secondaires pour garantir qu'aucune tâche ne ratera son échéance. Ce démarrage est effectué soit si la copie primaire est erronée, soit si elle dépasse le crédit de temps qui lui est allouée. La validation est effectuée par la borne de [LL73]. Ces travaux ont été améliorés par [DG09a], par l'insertion d'un algorithme favorisant la réexécution d'une copie primaire au détriment de la copie secondaire.

D'autres approches existent. Une première vise à considérer non pas le temps qui s'écoule, mais le nombre d'intervalles d'exécution disponibles (*Slotted Time*) [San+04]. La longueur de chacun de ses intervalles donne le temps d'exécution disponible, exprimé en *slots*, valant chacun une unité de temps. Un nombre s de slots disponibles correspond à une durée d'exécution de s unités de temps. Ainsi, pour chaque tâche d'un ensemble donné, s'il existe k slots disponibles tels que $\mathcal{R}_i + k \leqslant D_i$, l'ensemble de tâches est dit k-faisable. Les travaux [SSO05] proposent d'utiliser ces k slots disponibles pour permettre la réexécution de tâches erronées. L'inégalité proposée permet de faire varier les schémas de réexécution, notamment en fonction de la criticité des tâches.

Une autre approche consiste à considérer des ensembles mixtes de tâches critiques et non critiques [DAP08]. Dans ces travaux, le but est de garantir que les tâches critiques peuvent tolérer des fautes, via une méthode de recouvrement. Ce recouvrement se fait au détriment des tâches non critiques. Les auteurs proposent alors une méthode permettant de garantir une qualité de service aux tâches non critiques. Même si ces travaux sortent du périmètre d'étude de cette thèse, l'approche par qualité de service est très pertinente lorsque les tâches sont considérées du point de vue de leur criticité.

2.4 Conclusion

Ce chapitre a présenté l'état de l'art du domaine des ordonnancements temps réel tolérants aux fautes. De part sa construction, il montre que les analyses d'ordonnançabilité à conduire dans ce domaine sont fonction pour l'essentiel du modèle de distribution de fautes et des mécanismes de gestion des erreurs utilisés. Les ordonnancements tolérants aux fautes « par construction » constituent une exception à cette approche, l'objectif étant d'être tolérant à des fautes dont le modèle n'est pas défini.

La construction de la suite de ce manuscrit suit l'approche déduite de l'état de l'art. Le chapitre 3 présente le modèle de distribution de fautes en rafales ainsi que la taxonomie appliquée au mécanismes de gestion des erreurs. Le chapitre 4 montrer comment produire des analyses d'ordonnançabilités pour des ordonnancements devant être tolérants aux rafales de fautes, et présente les résultats de l'étude quantitative correspondante. Enfin, le chapitre 5 s'intéresse au cas particulier des analyses d'ordonnançabilité pour les ordonnancements partitionnés..

Chapitre 3

Rafales de fautes et taxonomie de la gestion des erreurs

Sommaire

Dans l'optique de proposer une analyse d'ordonnançablité pour les ordonnancements tolérants aux fautes, la définition d'un modèle de distribution de fautes, et le choix des mécanismes de tolérance aux fautes sont centraux. Le parti pris de ces travaux de thèse, concernant le modèle de distribution de fautes, est de considérer l'environnement comme agressif, et donc de caractériser l'agression. En ce qui concerne la tolérance aux fautes, l'objectif est de traduire en surcoût temporel, du point de vue de la tâche, le temps de mise en œuvre de la détection et de la correction d'erreurs. Ainsi, ce chapitre aborde la question de la modélisation de l'environnement retenu pour ces travaux, et le choix des mécanismes de gestion des erreurs.

Ce chapitre se structure comme suit. La section 3.1 décrit le modèle en rafale de fautes, modèle de distribution développé au cours de ces travaux de thèse. La section 3.2 présente quant à elle la taxonomie de la gestion des erreurs, issue d'une réflexion portant sur le classement et la comparaison des mécanismes de tolérance aux fautes dépeints dans le chapitre précédent. Enfin la section 3.3 conclut ce chapitre.

3.1 Rafales de fautes

Ces travaux de thèse s'intéressent aux phénomènes transitoires perturbant durablement un système temps réel. L'origine de ces perturbations s'avère multiple, et souvent liée à des effets physiques. Les conséquences vibratoires, thermiques ou électromagnétiques sur les systèmes temps réels éprouvent la robustesse des systèmes temps réel, à la fois au niveau matériel et logiciel. Au sein de la présente section, la question de la modélisation de ces perturbations est centrale, et doublée d'une volonté d'approcher la réalité temporelle des perturbations prises en compte. Le modèle en rafales de fautes traduit cette volonté.

3.1.1 Illustration de la problématique

La figure 3.1 montre un exemple de phénomène transitoire affectant un système temps réel. Le cas décrit est issu des documents [RE05] et [SR01] afférant à la certification des aéronefs civils. Un avion, en phase d'approche, est soumis aux perturbations générées par un radar rotatif. Ces perturbations provoquent des effets sur les équipements (capteurs, calaculateurs) et les données échangées, effets pouvant s'avérer néfastes pour le bon fonctionnement de l'avion.

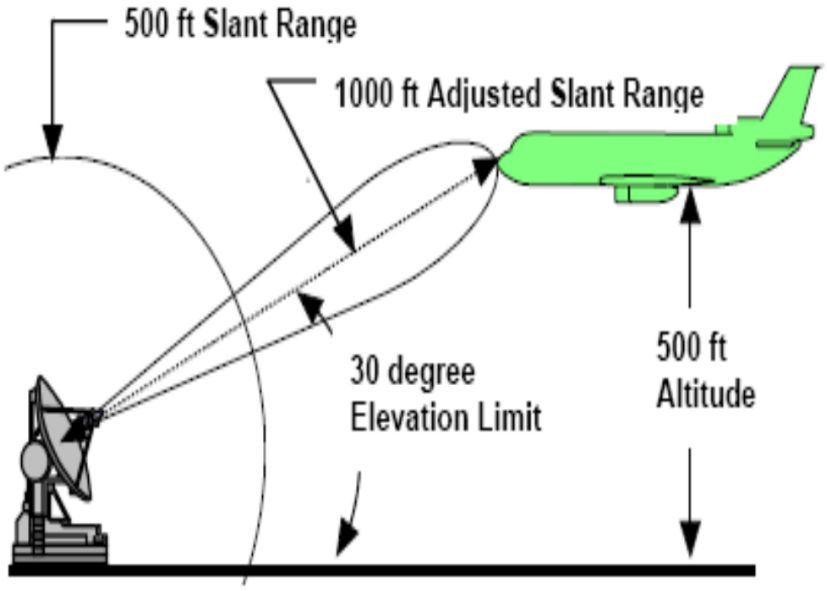


FIGURE 3.1 – Exemple de système temps réel soumis à une perturbation

Ces perturbations prennent la forme de champs électromagnétiques appelés HIRF (*High-Intensity Radiated Fields*). Les documents [RE05] et [SR01] précisent à la fois l'intensité et la forme de ces champs que doit être capable de subir l'aéronef afin d'obtenir

sa certification. Ces documents précisent aussi les durées d'exposition et les périodes de balayage moyen et pire cas des radars d'approche.

Ainsi, le cas moyen constaté pour les aéronefs est un temps d'exposition n'excédant pas quelques dixièmes de seconde avec une période de balayage de quelques secondes. Le pire cas décrit concerne les avions les plus lents. Compte tenu de leur faible vitesse, leur phase d'approche dure plus longtemps, augmentant de fait leur exposition aux effets indésirables dus au radar rotatif. L'évaluation chiffrée de cette situation conduit à considérer que ces aéronefs sont soumis à une exposition de $100ms$ toutes les $2s$. Le nombre total de balayages est égal à 15, correspondant à un temps d'approche de $30s$.

L'objectif de certification, associé à cette phase de vol, est donc de prouver qu'un aéronef, soumis durablement à des perturbations électromagnétiques affectant son comportement, est à même de fonctionner de façon sûre et déterministe. Cet exemple met en évidence que dans un environnement agressif, la validation d'un système temps réel perturbé durablement est nécessaire. Cette forme d'agression est prise en compte dans le modèle en **rafale de fautes**, décrit dans la sous-section suivante. Chaque balayage du radar est alors considéré comme la source d'une rafale de fautes.

3.1.2 Définition

Le modèle dit en rafales de fautes représente une distribution temporelle de fautes. Ce modèle est défini comme un intervalle de temps borné de longueur Δ_F, au sein duquel la distribution exacte des fautes est inconnue. Chaque début de rafale de fautes est séparé du suivant par un minimum de temps T_F (définition 2).

Définition 2 *Rafale de fautes*

Une rafale de fautes est un couple (Δ_F, T_F) *où* Δ_F *la durée de la rafale, et* T_F *sa pseudo-période d'occurence.*

Les fautes sont contenues dans Δ_F si bien qu'aucune faute ne peut exister en dehors de cet intervalle. La figure 3.2 décrit ce modèle.

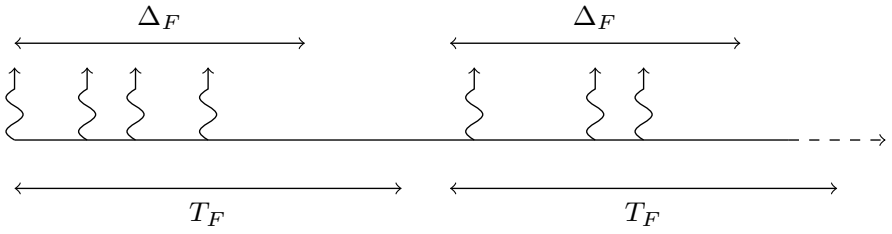


FIGURE 3.2 – Modèle en rafales de fautes

Les travaux menés considèrent que l'intervalle de temps minimum séparant deux fautes n'est pas connu. De fait, la distribution interne à la rafale de fautes est une inconnue de

la modélisation. Cette inconnue se justifie de la manière suivante : le modèle en rafales de fautes définit la réponse du système à une perturbation externe. L'obtention de la distribution interne est subordonnée à la connaissance effective des effets de la perturbation sur chaque élément du système temps réel, et à la manière de propager ces fautes. Compte tenu de la complexité de cette étude, et de l'explosion combinatoire qui lui est liée, l'approche retenue est de considérer la rafale comme un délai pendant laquelle le système est perturbé. Cette approche ne remet donc pas en cause les études temps réel du système.

De par sa construction, le modèle de fautes en rafales permet la modélisation temporelle de phénomènes physiques. Ainsi, tout phénomène se traduisant temporellement par une pseudo-périodicité d'apparition, et d'effets étendus dans le temps, est pris en compte par cette approche.

En reprenant l'exemple de la sous-section 3.1.1, la rafale de fautes $(100ms, 2s)$ modélise l'effet du champ électromagnétique généré par le radar rotatif pour un aéronef lent.

3.1.3 Lien avec les autres modèles de distributions de fautes

Les modèles de fautes pseudo-périodique et en rafales ne sont pas à opposer, leur capacité de modélisation étant différente. Là où le modèle pseudo-périodique se concentre sur la fréquence d'un évènement ponctuel provoquant une faute, le modèle en rafales propose une modélisation d'un phénomène durable dans le temps. De même l'approche par motif de fautes se révèle appropriée si les évènements affectant le sytème sont discrétisables, et si une séquence de fautes peut-être déterminée. En l'état, la définition du modèle en rafales de fautes contourne ce problème en reconnaissant le caractère aléatoire possible des fautes, sous forme d'une inconnue non bloquante pour l'étude des systèmes. Ce modèle est donc complémentaire des deux autres préexistants.

De plus, les problématiques liées aux rafales de fautes ou d'erreurs ne sont pas nouvelles en soi. Le problème a déjà été abordé à deux reprises. Initiallement dans les travaux [NSS00], ce problème est abordé pour modéliser les erreurs de transmission sur les réseaux de type CAN. Dans cette approche, un modèle probabiliste de fautes est proposé et traduit la notion de gravité vis à vis des transmissions. Cette gravité se traduit par le nombre d'erreurs à un instant t, ce qui définit la taille de la rafale. Dans ces travaux, la taille de la rafale est traduite en terme de durée.

Les travaux de [Ayd07] présentent un groupement de k fautes sur un intervalle de temps borné comme une rafale de fautes. La précondition d'utilisation de ce modèle de distribution de fautes est donc de pouvoir discrétiser le nombre de fautes affectant un système temps réel. L'approche retenue par ces travaux propose de s'affranchir de cette précondition en considérant que la distribution interne des fautes est inconnue. Ainsi, le modèle en rafales de fautes généralise celui en motif de fautes.

3.2 Taxonomie de la gestion des erreurs

En préambule de cette section, deux hypothèses sont faites. En premier lieu, les moyens nécessaires à la détection et à la correction d'erreurs au sein d'une tâche sont supposés disponibles. L'intérêt est porté seulement sur leur temps de mise en œuvre . En second lieu, la couverture des mécanismes de détection d'erreurs est supposée complète, *i.e* toutes les erreurs contenues dans les tâches sont détectables et détectées.

L'état de l'art présenté dans le chapitre précédent met en évidence la quantité de travaux déjà effectués sur le domaine des ordonnancements tolérants aux fautes. Au cours de ces travaux de thèse, la nécessité de classer et de comparer les différents résultats, notamment pour évaluer les axes de recherche prometteurs, a conduit à la définition d'une taxonomie. Cette taxonomie de la gestion des erreurs prend en compte deux aspects importants : le couple détection-correction d'erreurs, et le comportement de l'ordonnanceur lors de la détection d'erreurs au sein d'une tâche.

Ces deux aspects sont présentés au travers des notions de stratégie de recouvrement d'erreurs et de tactique de recouvrement d'erreurs, décrites dans les sous-sections suivantes. Cette section présente aussi les stratégies et tactiques qui illustreront les méthodes d'analyse d'ordonnançabilité proposées au chapitre 4.

3.2.1 Stratégies de recouvrement d'erreurs

Une stratégie de recouvrement d'erreurs définit le comportement de l'ordonnanceur vis à vis de la tâche erronée et des tâches préemptées à l'instant de la détection d'une erreur. S'il est évident d'appliquer une méthode de correction à la tâche erronée, la question est plus complexe quant aux tâches préemptées. En effet, la faute qui est à l'origine de l'erreur détectée affecte potentiellement une ressource commune à plusieurs tâches. Sous cette hypothèse, un comportement possible de l'ordonnanceur est d'appliquer préventivement une méthode de correction d'erreurs aux tâches préemptées.

Dans ces travaux de thèse, deux grandes catégories de stratégies de recouvrement sont proposées : les stratégies *simple* et *multiple*. Une stratégie simple signifie que l'ordonnanceur applique une méthode de correction d'erreurs uniquement sur la tâche erronée. De fait, si des erreurs sont présentes dans les tâches préemptées, l'ordonnanceur attendra la détection effective de l'erreur pour lui appliquer la méthode de correction correspondante.

Dans le cadre d'une stratégie multiple, si la détection d'une erreur impose que la tâche erronée soit corrigée, elle impose aussi l'application préventive des méthodes de correction à l'ensemble des tâches préemptées. Ce comportement permet de limiter le nombre de détections ultérieures en purgeant le système, au détriment de corrections de tâches potentiellement saines.

La figure 3.3 illustre la logique suivie par les différentes stratégies. Ainsi, pour un ensemble de deux tâches τ_1 et τ_2, avec τ_1 tâche de plus haute priorité, la détection de

l'erreur latente présente dans τ_1 permet d'envisager deux comportements. Soit la détection d'erreurs entraîne la correction de τ_1, et la présence d'une erreur latente dans τ_2 sera détectée et corrigée ultérieurement (stratégie simple). Soit la détection d'erreurs dans τ_1 entraîne la correction de τ_1 et la correction préventive de τ_2 (stratégie multiple).

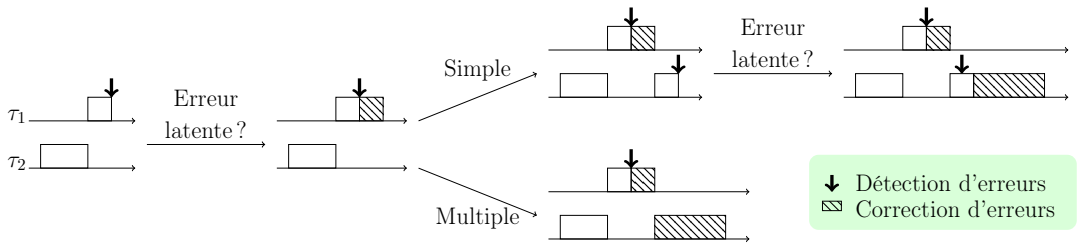


FIGURE 3.3 – Stratégie simple et stratégie multiple

L'application de ce taxon à l'état de l'art du domaine montre que ces différents travaux concernent des stratégies simples. Les travaux de [PM98] représentant l'exception qui confirme la règle. D'une manière générale, le choix d'étudier uniquement cette stratégie est dicté par le modèle de distribution de fautes. Le traitement d'erreurs dues à des fautes ponctuelles (pseudo-périodiques par exemple) rend peu vraisemblable la possibilité que plusieurs tâches soient affectées par une faute unique. Le choix du modèle en rafale de fautes remet en cause cette analyse, comme montré plus haut.

Pour conclure, le choix d'une stratégie de recouvrement d'erreurs affère au concepteur du système temps réel. Néanmoins, ce choix peut être contraint à la fois par la nature des erreurs considérées et leur origine, mais aussi par les caractéristiques de la plateforme d'accueil des tâches. Ainsi, dans le cadre d'un *Commercial Off the Shelf* (COTS), la sélection d'une stratégie dépend des services offerts par la plateforme.

3.2.2 Tactiques de recouvrement d'erreurs

La notion de tactique de recouvrement concerne la détection des erreurs au sein d'une tâche et leur correction. Pour chaque tâche, les moyens de détection et de correction d'erreurs peuvent varier suivant leur criticité, l'algorithme qui y est implanté et leur accès à différentes ressources (mémoire, périphériques), . Du point de vue de la détection d'erreurs, ces travaux se concentrent sur l'instant de détection, le mécanisme étant supposé disponible.

Une tactique de recouvrement d'erreurs se définit alors comme le couple instant de détection et mécanisme de détection d'erreurs, notée *détection/correction*. L'application de ce taxon à l'état de l'art du chapitre précédent met en évidence plusieurs éléments. Les instants de détection considérés sont soit à la fin de la tâche (*End Detection - ED*) soit à la fin d'une section de tâches (*Checkpoint - CP* [Bro79]). Le cas décrit par [PM98], dans lequel l'instant de détection intervient avant toute préemption (*Before Preemption -*

BP) est considéré comme un cas particulier d'ED. En effet, positionner un mécanisme de détection avant toute préemption suppose un contrôle de préemption difficile à mettre en oeuvre. Ces travaux considèrent donc que dans le pire cas, la préemption intervient entre la fin de la tâche et la détection d'erreur.

Concernant la méthode de correction d'erreurs, quatre principales méthodes sont utilisées : réexécution complète (*Full Re-execution- FR*), le mécanisme d'exception (*Exception Handler - EH*), le bloc de recouvrement (*Recovery Block - RB*) et la réexécution partielle de la tâche (*Checkpoint - CP*). Ainsi, les principales tactiques de recouvrement d'erreurs décrites dans l'état de l'art sont :

- ED/*, où * représente soit FR, soit EH, soit RB
- CP/CP

Outre la facilité de classification, la notion de tactique de recouvrement permet d'évaluer aisément le coût temporel dû à chaque action, et de fait, d'en déduire le surcoût temporel de chaque tactique. Dans le cadre d'une validation de système temps réel, ce surcoût temporel pour chaque tactique doit être pris dans la pire configuration. Pour une tâche τ_i, le temps de détection pire cas est noté C_i^D et le coût de la méthode de correction d'erreurs est notée $\overline{C_i}$. Appliquée aux tactiques précédemment décrites, l'évaluation des différents surcoûts est :

- ED/FR : $C_i^D = C_i$ et $\overline{C_i} = C_i$
- ED/EH : $C_i^D = C_i$ et $\overline{C_i} = \max(C_i^{EH_j})$
- ED/RB : $C_i^D = C_i$ et $\overline{C_i} = \max(C_i^{RB_j})$
- CP/CP : $C_i^D = \max(\delta_i) + O_A$ et $\overline{C_i} = \max(\delta_i) + O_A$

La figure 3.4 illustre la tactique CP/CP. La tâche τ est découpée en trois secteurs δ_1, δ_2 et δ_3 de longueur respective 1, 1, 5 et 1 unités de temps. Ainsi, du point de vue la tâche τ, le secteur le plus pénalisant dans lequel peut résider une erreur est le secteur δ_2, qui a le plus long temps d'exécution. Dans ce cas là, C^D et $\overline{C}$ valent $\delta_2 + O_A$. Pour des raisons de simplicité, la valeur de O_A possède une valeur unique quelque soit le secteur de la tâche concerné.

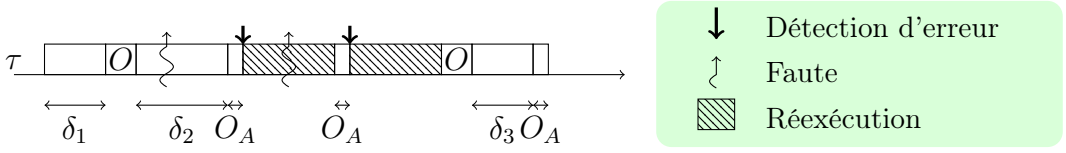


FIGURE 3.4 – Illustration d'une tactique de recouvrement CP/CP

Du point de vue des travaux proposés, ce taxon met l'accent donc sur le comportement temporel de la détection et de la correction d'erreurs, permettant ainsi de réaliser les analyses d'ordonnançabilité présentées dans le chapitre 4.

3.2.3 Exemples de stratégie

Dans la suite de ce manuscrit, les principaux résultats sur l'analyse d'ordonnançabilité seront illustrés par l'application des deux stratégies suivantes : ED/FR/S (*End Detection/Full Reexecution/Simple*) et ED/FR/M (*End Detection/Full Reexecution/Multiple*).

Stratégie ED/FR/S Dans le cadre de la stratégie ED/FR/S, la tactique appliquée aux tâches est unique : la détection d'erreur intervient à la fin de la tâche, et la méthode de correction est effectuée par une réexécution complète de l'instance erronée ($C_i^D = C_i$ et $\overline{C_i} = C_i$). La stratégie est simple, une détection d'erreur entraîne la correction de la tâche fautive.

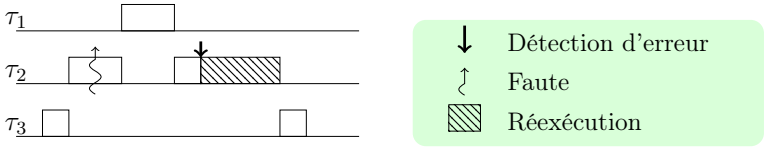


FIGURE 3.5 – Stratégie ED/FR/S

Stratégie ED/FR/M La tactique de cette stratégie est identique à ED/FR/S. La stratégie est multiple, la détection d'erreur dans une tâche entraînant à la fois la correction de la tâche erronée, et celles de toutes les tâches préemptées.

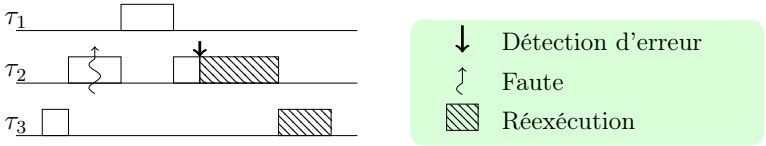


FIGURE 3.6 – Stratégie ED/FR/M

3.3 Conclusion

Ce chapitre a décrit le modèle de distribution de fautes en rafales. Ce modèle élaboré par ces travaux de thèse a pour objectif la description de perturbations affectant un système temps réel de manière durable. Les perturbations électromagnétiques, par exemple générées par un radar, font partie des perturbations prises en compte par le modèle en rafales de fautes. Vis à vis des modèles préexistants, le modèle en rafales de fautes est complémentaire au modèle de fautes pseudo-périodiques et à celui en motifs de fautes.

De plus, ce chapitre a établi une taxonomie de la gestion des erreurs. Cette taxonomie définit les notions de stratégie de recouvrement d'erreurs et de tactique de recouvrement

d'erreurs. La stratégie définit le comportement de l'ordonnanceur vis à vis des tâches préemptées lorsqu'une erreur est détectée dans la tâche en cours d'exécution. La tactique permet quant à elle de gérer la correction d'erreur sur cette même tâche. Cette taxonomie fournit aussi un moyen de classement des travaux précédents, et permet d'évaluer la couverture des résultats préexistants en regard des mécanismes de gestion des erreurs considérés.

Le chapitre suivant intègre ces deux résultats dans le but de conduire des analyses d'ordonnançabilité dans le cas d'ordonnancements tolérants aux rafales de fautes. Ce chapitre décrit notamment la construction de l'équation de pire temps de réponse sous rafales de fautes, montre comment évaluer temporellement l'impact des stratégies de recouvrement, et donne les résultats de l'étude quantitative menée.

Chapitre 4

Ordonnançabilité sous rafales de fautes

Sommaire

Ce chapitre présente les analyses d'ordonnnançabilité developpées au cours de cette thèse, sur la base du modèle en rafale de fautes. Ces analyses, basées sur l'évaluation du pire temps de réponse, intègrent la taxonomie définie au chapitre précédent. Ce chapitre démontre la faisabilité d'ensemble de tâches soumises à des perturbations durables dans le temps. Le sous-jacent important est la démonstration qu'un système, dont le comportement dysfonctionnel peut être difficile à évaluer, garantit le respect de ses échéances temporelles avec un temps de reprise pire cas déterministe.

Ce chapitre se structure comme suit. La section 4.1 donne la formule générale de l'équation du pire temps de réponse d'une tâche soumise à une rafale de fautes. La section 4.2 montre comment sont intégrés les surcoûts temporels dus aux stratégies de recouvrement d'erreurs. La section 4.3 présente les résultats obtenus par l'application des différentes stratégies à des ensembles de tâches aléatoires. Enfin la section 4.4 conclut ce chapitre.

4.1 Équation de temps de réponse sous rafales de fautes

Les sections 1.2 et 2.3 donnent un aperçu au lecteur du panel de techniques de validation utilisables dans le cadre de la théorie de l'ordonnancement, sous fautes ou non. L'intérêt porté par ces travaux au calcul de pire temps de réponse est multiple.

Tout d'abord cette technique de validation fournit un résultat global (faisabilité de l'ensemble de tâches) et un résultat local (ordonnançabilité de la tâche). Même si un ensemble n'est pas faisable, il est possible d'extraire un sous-ensemble faisable. En second lieu, l'information donnée au concepteur permet de caractériser le comportement temporel de la tâche. Enfin, cette approche permet de construire la succession d'évènements qui conduit le système à être perturbé, puis restauré. Ce dernier point facilite la compréhension du concepteur du comportement temporel du système en présence de rafales de fautes.

Dans cette section, le modèle de tâches considéré par ces travaux de thèse, les hypothèses sur les rafales de fautes et sur le système temps réel sont précisés. Dans le cadre de travail ainsi défini, l'équation de pire temps de réponse sous rafales de fautes est établie, puis déclinée suivant la stratégie de recouvrement d'erreurs considérée.

4.1.1 Modèle de tâche

Soit Γ un ensemble de n tâches $\{\tau_1, ..., \tau_n\}$ tel que Γ soit faisable en l'absence de fautes. Chaque tâche τ_i est représentée par un 5-uplet $(C_i, T_i, D_i, C_i^D, \overline{C_i})$ où C_i est son pire temps d'exécution, T_i sa période, D_i son échéance, C_i^D le pire temps de détection et $\overline{C_i}$ le pire temps de correction d'erreurs. L'échéance des tâches est inférieure ou égale à leur requête ($D_i \leqslant T_i$). Les tâches peuvent soit avoir un comportement strictement périodique, ou sporadique. Dans le cas d'une tâche sporadique, la période T_i correspond à la période minimale existante entre deux instances de la tâche. Enfin, les tâches sont indépendantes, *i.e* sans ressources partagées et sans relation de précédence.

Le système est supposé monoprocesseur. Les tâches sont assignées au processeur suivant des priorités allouées par un algorithme à priorité fixe. A tout instant t, la tâche qui s'exécute sur le processeur est toujours celle de plus haute priorité parmi les tâches éligibles. Ces priorités allouées sont distinctes pour chaque tâche $\tau_i \in \Gamma$, tel que 1 soit la plus haute priorité et n la plus basse. La tâche de plus haute priorité est donc τ_1, telle que $P(\tau_1)$=1. Ainsi, pour toute tâche τ_i, $P(\tau_i) = i$. L'ordonnanceur est préemptif, et les coûts temporels dus aux préemptions sont négligeables. L'exécution du mécanisme de recouvrement d'erreurs est faite suivant la priorité initiale de la tâche erronée.

4.1.2 Hypothèses générales

L'équation de pire temps de réponse est établie suivant l'hypothèse classique de l'ordonnancement tolérant aux fautes suivante :

Hypothèse 1 *La période T_F séparant deux débuts de rafale de fautes est supérieure ou égale à la plus grande des échéances de l'ensemble de tâches.*

$$T_F \geqslant max_{i \in n}(D_i) \tag{4.1}$$

L'hypothèse 1 est cohérente avec les paramètres temporels décrits dans la section 3.1.1, sur les bases d'une exposition à une perturbation de quelques dixièmes de seconde avec une périodicité de quelques secondes.

Cette hypothèse impose que chaque instance de tâche sera perturbée au plus par une rafale de fautes. La propriété 1 en est déduite :

Propriété 1 *Pour chaque tâche, le calcul de pire temps de réponse intègre au plus une rafale de fautes.*

Si une tâche τ_i est non ordonnançable, alors l'algorithme de point fixe s'arrêtera avant le prochain début de rafales de fautes puisque $\mathcal{R}_i > D_i$. Sinon, la tâche est ordonnançable et le pire temps de réponse calculé sera inférieur ou égal à l'échéance, et de fait le calcul s'arrêtera avant le début de la rafale de fautes suivante. Ainsi, la propriété 1 garantit la convergence de l'algorithme de point fixe.

Par ailleurs le calcul du pire temps de réponse d'une tâche est soumis à l'hypothèse 2, qui concerne le comportement des équipements affectés par une rafale de fautes :

Hypothèse 2 *A la fin de la rafale de fautes, les équipements affectés par ces perturbations recouvrent leur comportement nominal.*

Sous l'hypothèse 2, le temps de recouvrement des équipements depuis un état perturbé est considéré comme nul. Le retour à un comportement nominal ne nécessite pas une manipulation telle qu'un *reset* ou un redémarrage. Cependant, les erreurs latentes contenues dans la tâche courante sur le processeur, et sur les tâches préemptées, restent présentes jusqu'à leur détection effective.

4.1.3 Une équation conservative

Proposer une équation de temps de réponse combinant les caractéristiques temporelles de la perturbation et des mécanismes de tolérance aux fautes pose le problème de l'évaluation du pire cas. Plus finement, cette évaluation du pire cas impose de trouver le ou les instants d'arrivée de la rafale de fautes qui maximisent le temps de réponse de la tâche considérée. Une solution brutale serait de calculer et d'explorer l'ensemble des traces d'exécution possibles. Cependant, cette solution se heurte à la combinatoire élevée du problème.

Ces travaux de thèse proposent de contourner ce problème par l'établissement d'une équation conservative. La solution de cette équation représentera une **borne supérieure**

du pire temps de réponse réel. Ainsi, le temps de réponse sous rafales de fautes, noté $\mathcal{R}_i^{\Delta_F}$, est exprimé comme la somme du pire temps de réponse sans fautes $\mathcal{R}_i$, de la durée de la rafale de fautes Δ_F, des interférences $I_i^{\Delta_F}$ dues aux préemptions par les tâches de plus haute priorité après la fin de la perturbation, et d'un terme F_i représentant le coût temporel du mécanisme de tolérance aux fautes. L'expression correspondante est donnée par l'équation 4.2 :

$$\mathcal{R}_i^{\Delta_F} = \mathcal{R}_i + \Delta_F + (I_i^{\Delta_F} + F_i) \tag{4.2}$$

L'équation 4.2 définit trois parties ($\mathcal{R}_i$, Δ_F, $I_i^{\Delta_F} + F_i$). L'évaluation du pire temps de réponse $\mathcal{R}_i^{\Delta_F}$ correspond à la somme des valeurs maximales prises par ces différentes parties dans leur pire scénario. Puisque toutes ces valeurs sont positives, et ajoutées les unes aux autres, la somme des maxima est supérieure ou égale au maximum de la somme. La valeur évaluée du pire temps de réponse $\mathcal{R}_i^{\Delta_F}$ est donc une borne supérieure de sa valeur exacte. Le calcul proposé introduit de fait un pessimisme dans certains cas, au profit d'une complexité de calcul faible. Le résultat est donc une condition suffisante pour démontrer qu'une tâche est ordonnançable sous une rafale de fautes.

Pour chaque partie, le pire scénario est considéré. Les paragraphes suivant décrivent chacune de ces parties, en mettant en évidence le raisonnement sous-jacent :

Jusqu'au début de la rafale Le contexte d'exécution de la tâche τ_i est tel que son temps de réponse est égal au pire temps de réponse $\mathcal{R}_i$ sans faute. Un instant ε avant la terminaison effective de τ_i, une rafale de fautes débute, une faute impactant la tâche considérée et devant conduire à une détection d'erreur. En réponse à la détection d'erreur, une méthode de recouvrement d'erreur sera alors exécutée.

Cette situation est décrite par la figure 4.1, dans laquelle la tâche considérée est τ_3. Dans le pire cas, l'instant ε est tel que la durée écoulée entre l'occurence de faute dans τ_3 et la détection d'erreur tend vers 0. La valeur de $\mathcal{R}_3$ est donc égale à son pire temps de réponse sans fautes.

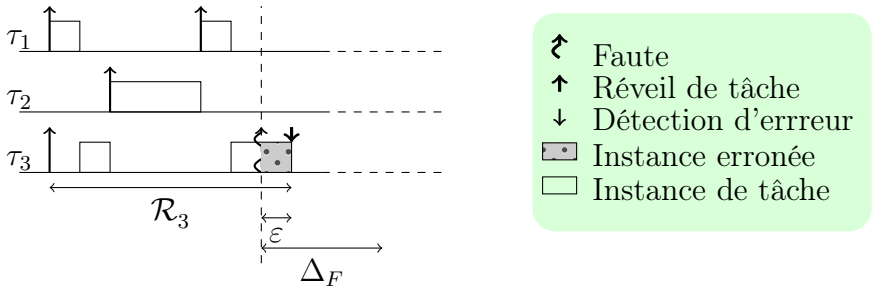


FIGURE 4.1 – Explication de $\mathcal{R}_i^{\Delta_F}$: scenario jusqu'au début de la rafale de fautes

Pendant la rafale La distribution de fautes due à la rafale, ainsi que ses effets sur les tâches sont inconnus. De fait, la séquence d'exécution des instances des tâches, de détection d'erreur et des méthodes de recouvrement d'erreurs est inconnue. Certaines instances de plus hautes priorités peuvent s'exécuter, contenir des erreurs et être corrigées, ou encore contenir des erreurs qui ne seront détectées qu'après la fin de la rafale de fautes. Dans tous les cas de figure, les mécanismes de détections et de corrections d'erreurs sont actifs, mais le resultat de leur application est inconnu durant la durée Δ_F de la rafale de fautes. La seule certitude concernant l'établissement de l'équation 4.2 est que la tâche τ_i considérée ne réussit pas à se terminer pendant la rafale.

La figure 4.2 montre un scénario possible durant la rafale de fautes. Une erreur est détectée dans τ_3 qui exécute une méthode de correction d'erreur. Une instance de la tâche τ_2 préempte cette correction. Cette instance est soumise à des fautes, qui conduisent à une détection d'erreur. Cependant, la tâche τ_1 se réveille, et une de ses instances préempte τ_2. Une erreur est par la suite détectée dans l'instance de τ_1, et la méthode de correction exécutée se termine correctement. Cette instance réussit donc à s'exécuter au sein de la rafale de fautes. L'ordonnanceur rend la main à la tâche τ_2, qui exécute une méthode de correction d'erreur, qui elle-même sera erronée à cause d'une nouvelle faute. La rafale de fautes se termine, mais l'erreur ne sera détectée que plus tard.

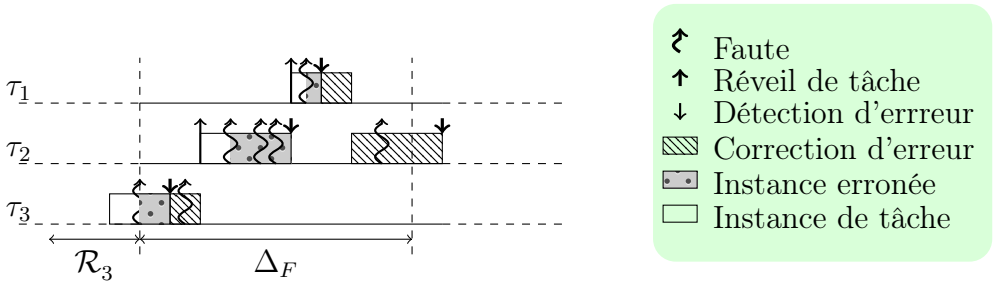


FIGURE 4.2 – Explication de $\mathcal{R}_i^{\Delta_F}$: scenario possible durant la rafale

A la fin de la rafale A partir de l'instant $\mathcal{R}_i + \Delta_F$, les mécanismes de tolérance aux fautes corrigent les erreurs résiduelles dans les tâches. Le pire scénario, dans la troisième partie de l'équation, intègre donc le coût temporel des mécanismes de détection et de correction d'erreurs. Ces mécanismes sont inclus dans le terme F_i, qui dépend de la stratégie de recouvrement choisie. Le terme F_i intègre à la fois le coût de la correction de τ_i, ainsi que les éventuelles corrections des tâches de plus haute priorité (cf section 4.2). Les corrections d'erreurs s'exécutant à la priorité initiale de la tâche τ_i, le pire scénario doit aussi intégrer les préemptions dues aux tâches de plus haute priorité. Les interférences dues aux tâches de plus haute priorité sont notées $I_i^{\Delta_F}$.

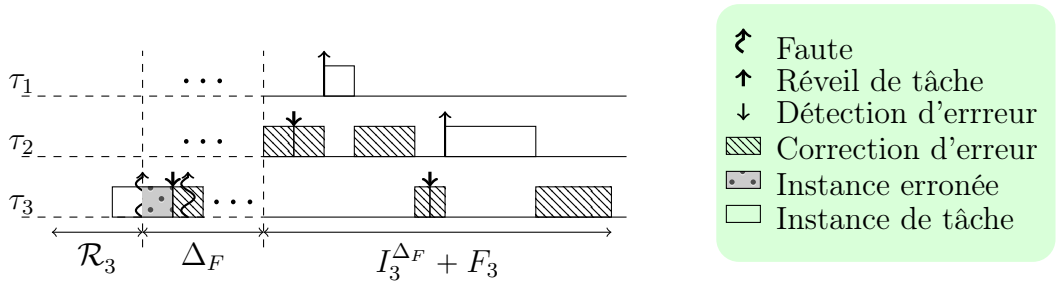


FIGURE 4.3 – Explication de $\mathcal{R}_i^{\Delta_F}$: scenario possible après la rafale

La figure 4.3 présente une séquence possible après la fin de la rafale de fautes. Une correction d'erreur au sein d'une instance de la tâche τ_2 est en cours, et celle-ci contient encore une erreur. Après la détection de cette erreur, le mécanisme de correction d'erreurs est de nouveau appliqué. Cette correction est préemptée par le réveil de la tâche τ_1. Suite à la terminaison de l'instance de cette tâche, l'ordonnanceur élit de nouveau la tâche τ_2 au processeur, qui termine donc sa correction. A l'issue, la tâche τ_3 reprend sa correction, mais une erreur est détectée, ce qui déclenche une nouvelle correction. Cette action est préemptée par une instance de τ_2, et la terminaison effective est obtenue au bout du temps $I_3^{\Delta_F} + F_3$.

Les interférences dues aux tâches de plus hautes priorités qui interviennent à partir de l'instant $\mathcal{R}_i + \Delta_F$ sont définies suivant le raisonnement suivant. Pour une tâche τ_j telle que $j \in hp(i)$, deux cas sont possibles :

1. Il n'y a pas d'instance de τ_j éligible pendant la rafale, et la prochaine instance se réveillera au plus tôt à l'instant $\mathcal{R}_i + \Delta_F$.
2. Une instance de τ_j est éligible, s'exécute, et dans le pire scénario, est affectée et corrigée après la fin de la rafale de fautes. Dans ce cas, la prochaine instance se réveillera au plus tôt après la correction de cette instance. Sinon cette tâche n'est pas ordonnançable.

L'évaluation exacte du nombre de préemptions dues aux tâches de plus hautes priorités représente la même problématique que celle du placement de la rafale de fautes. Ces travaux proposent d'adopter de nouveau une approche conservative en considérant l'instant $\mathcal{R}_i + \Delta_F$ comme **un nouvel instant critique**. Depuis cet instant, le terme $I_i^{\Delta_F}$ intègre toutes les préemptions dues aux tâches de plus haute priorité. Cette approche évalue un nombre de préemptions supérieur ou égal à la réalité, introduisant dans certains cas un pessimisme déjà mentionné plus haut. L'équation 4.3 donne l'expression du terme d'interférences $I_i^{\Delta_F}$:

$$I_i^{\Delta_F} = \sum_{j \in hp(i)} \left\lceil \frac{\mathcal{R}_i^{\Delta_F} - (\mathcal{R}_i + \Delta_F)}{T_j} \right\rceil C_j \tag{4.3}$$

La valeur des termes $\mathcal{R}_i$, Δ_F et F_i se calcule indépendamment. Le terme d'interférences $I_i^{\Delta_F}$ dépendant du terme $\mathcal{R}_i^{\Delta_F}$, la résolution de l'équation 4.2 s'effectue par l'application d'un théorème du point fixe. La convergence de l'algorithme est assurée par les hypothèses 1 et 2. L'ordonnançabilité de la tâche τ_i s'obtient alors par comparaison de $\mathcal{R}_i^{\Delta_F}$ avec D_i . Si chaque tâche est ordonnançable, alors l'ensemble de tâches est faisable suivant la stratégie de recouvrement d'erreurs et la rafale de fautes considérés.

4.2 Evaluation de l'impact des stratégies de recouvrement

Pour une tâche τ_i, la valeur du terme F_i dépend du coût temporel de sa propre correction, des corrections des tâches de plus haute priorité, et de l'impact des stratégies de recouvrement au travers de l'instant de détection des erreurs. Le scénario d'évaluation est celui du pire cas. Le caractère conservatif de l'approche de ces travaux permet d'évaluer indépendamment le terme F_i. Quelque soit la durée de la rafale de fautes, dans le pire cas, la séquence de correction est toujours la même.

4.2.1 Cas particulier du terme de recouvrement F_1

L'évaluation du terme F_i de la tâche de plus haute priorité ($P = 1$) ne dépend pas de la stratégie de recouvrement. Par définition, cette tâche ne peut-être préemptée. Ainsi, l'occurence d'une rafale de fautes, juste avant sa complétion effective conduit au pire cas suivant : la fin de la dernière exécution du mécanisme de correction d'erreurs coïncide avec la fin de la rafale. Cette dernière exécution est erronée. Á l'issue de la fin de la rafale de fautes, une détection d'erreur et une correction d'erreurs sont donc nécessaires. Suivant ce scenario, l'équation 4.4 donne l'expression du terme F_1 :

$$F_1 = C_1^D + \overline{C_1} \tag{4.4}$$

Ainsi, quelque soit la stratégie de recouvrement considérée, l'expression 4.5 donne l'équation du pire temps de réponse $\mathcal{R}_1^{\Delta_F}$:

$$\mathcal{R}_1^{\Delta_F} = C_1 + C_1^D + \overline{C_1} + \Delta_F \tag{4.5}$$

4.2.2 Cas d'une stratégie simple

Au niveau de l'ordonnanceur, la stratégie de recouvrement d'erreurs est simple. Cette sous-section présente successivement la forme générale du terme de recouvrement F_i, puis sa déclinaison dans le cadre d'une stratégie ED/FR/S.

4.2.2.1 Forme générale du terme de recouvrement F_i

Sous l'hypothèse de rafale de fautes, au plus une instance de chaque tâche de plus haute priorité peut être affectée. Dans le pire scénario, à la fin de la rafale, toutes les instances de plus haute priorité contiennent une erreur. De plus, la détection de ces erreurs sera effective après un temps écoulé égal à C_i^D. En conséquence, le recouvrement d'erreurs, dans la stratégie considérée, nécessite le temps d'exécution $\overline{C_i}$. Ce scénario maximise donc la valeur du terme de recouvrement F_i. L'équation correspondante s'exprime donc comme la somme, sur les tâches de plus haute priorité et la tâche concernée, de $\overline{C_j}$ et C_j^D (équation (4.6)).

$$\forall(i > 1)\ F_i = \sum_{j \in \{i\} \cup hp(i)} (C_j^D + \overline{C_j}) \tag{4.6}$$

Intuitivement, l'équation (4.6) correspond à l'application de la tactique de recouvrement d'erreurs sur chaque tâche de plus haute priorité et sur la tâche considérée. Une détection et une correction d'erreurs sont donc nécessaires pour garantir la tolérance aux fautes de l'ensemble de tâches dans le pire cas.

4.2.2.2 Stratégie ED/FR/S

Dans le cadre d'une stratégie ED/FR/S, la détection d'erreur intervient à la fin de la tâche. et la méthode de recouvrement d'erreur est une réexécution complète de la tâche erronée ($C_i^D = C_i$ et $\overline{C_i} = C_i$). L'expression du terme de recouvrement F_i est donnée par l'équation (4.7).

$$\forall(i > 1)\ F_i = 2 \times \sum_{j \in \{i\} \cup hp(i)} C_i \tag{4.7}$$

Echéancier commenté La figure 4.4 décrit l'évaluation du pire temps de réponse $\mathcal{R}_3^{\Delta_F}$. La représentation du pire cas sur cet échéancier montre qu'à la fin de la rafale de fautes Δ_F, les tâches τ_1, τ_2, τ_3 sont erronées. La stratégie ED/FR/S s'applique telle que la tâche τ_1 est corrigée après l'application de la tactique de recouvrement. Enuite, l'erreur de la tâche τ_2 est détectée, et la méthode de recouvrement d'erreur s'exécute. La priorité de cette exécution étant égale à la priorité initiale de τ_2, elle est préemptable par toute nouvelle instance de la tâche τ_1. Cet événèment arrive, et la méthode de recouvrement reprend son exécution après la complétion de la tâche τ_1. Enfin, la complétion de la tâche τ_3 est effective après l'application de la tactique de recouvrement.

Exemple La table 4.1 décrit les résultats obtenus sur un exemple de trois tâches. L'ordonnanceur assigne les tâches sur le processeur suivant l'algorithme Rate Monotonic. Une rafale de fautes d'une durée Δ_F égale à 50 unités de temps perturbe l'exécution de cet ensemble de tâches.

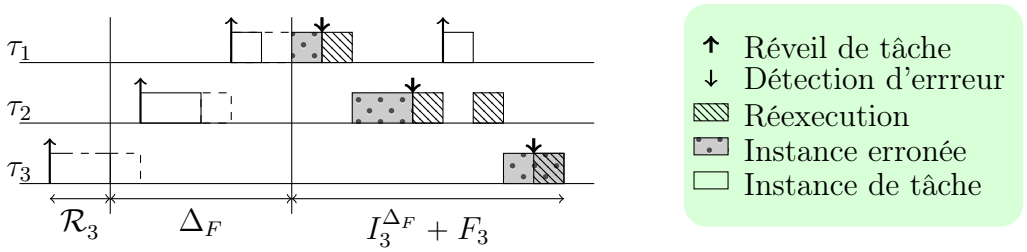


FIGURE 4.4 – Evaluation de $\mathcal{R}_3^{\Delta_F}$ sous la stratégie ED/FR/S

$\Delta_F = 50$						
P	T	C	D	$\mathcal{R}$	F	$\mathcal{R}^{\Delta_F}$
1	300	10	300	10	20	80
2	500	50	500	60	120	240
3	800	150	800	210	420	750

TABLE 4.1 – Exemple dans le cadre d'une stratégie ED/FR/S

Sous les hypothèses considérées, l'ensemble de tâches est faisable. Une étude plus avant des résultats montre que l'impact de la rafale de fautes est plus important sur les tâches de plus faible priorité. Dans le pire cas, le surcoût temporel nécessaire à la complétion de la tâche τ_3 est de 540 unités de temps, alors qu'il est de seulement 70 unités de temps pour la tâche τ_1.

4.2.3 Cas d'une stratégie multiple

Au niveau de l'ordonnanceur, la stratégie de recouvrement d'erreurs est multiple. Cette sous-section présente successivement la forme générale du terme de recouvrement F_i, puis sa déclinaison dans le cadre d'une stratégie ED/FR/M.

4.2.3.1 Forme générale du terme de recouvrement F_i

Sous l'hypothèse de rafales de fautes, au plus une instance de chaque tâche de plus haute priorité peut être affectée. L'établissement du pire cas conduit à considérer les évènements se produisant à la fin de la rafale de fautes. La détection d'une erreur conditionne l'application de la stratégie multiple. Cette détection intervient au plus tard après un temps écoulé égal au plus long C_i^D parmi les tâches de plus haute priorité. A partir de cet instant, la séquence de correction est maximale si toutes les tâches de plus haute priorité et la tâche considérée appliquent leur méthode de recouvrement d'erreurs. L'équation

(4.8) donne la forme générale du terme de recouvrement F_i.

$$\forall(i>1)\ F_i = \max_{j\in hp(i)}(C_j^D) + \sum_{j\in\{i\}\cup hp(i)} \overline{C_j} \tag{4.8}$$

Une approche intuitive de l'équation (4.8) revient à considérer que pour le sous-ensemble de tâches composé de la tâche considérée et de ses tâches de plus haute priorité, une détection d'erreur provoque la réexécution de toutes les tâches du sous-ensemble.

4.2.3.2 Cas de la stratégie ED/FR/M

Dans le cadre d'une stratégie ED/FR/M, la détection d'erreur intervient à la fin de la tâche. et la méthode de recouvrement d'erreur est une réexécution complète de la tâche erronée ($C_i^D = C_i$ et $\overline{C_i} = C_i$). L'expression du terme de recouvrement F_i est donnée par l'équation (4.9).

$$\forall(i>1)\ F_i = \max_{j\in hp(i)}(C_j) + \sum_{j\in\{i\}\cup hp(i)} C_j \tag{4.9}$$

Echéancier commenté La figure 4.5 décrit le scénario de l'évaluation du temps de réponse $\mathcal{R}_i^{\Delta_F}$. Sur cet échéancier, à la fin de la rafale des fautes Δ_F, la tâche τ_3 s'exécute et contient une erreur. Suite à la détection de cette erreur, la stratégie ED/FR/M s'applique. La séquence détection et correction d'erreurs de la tâche τ_1 représente une tactique de recouvrement. Par la suite, les tâches τ_2 et τ_3 sont complètement réexécutées à leur priorité initiale. Ainsi, la tâche τ_3 voit sa réexécution préemptée par l'exécution d'une instance de la tâche τ_1.

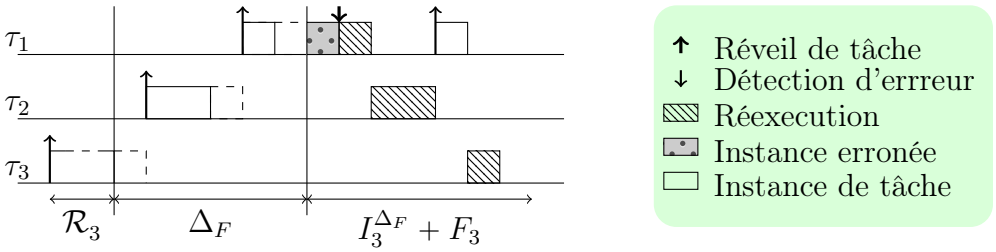


FIGURE 4.5 – Evaluation de $\mathcal{R}_3^{\Delta_F}$ sous la stratégie ED/FR/M

Exemple La table 4.2 contient les résultats de l'application de la stratégie ED/FR/M à l'ensemble de tâches décrit dans la Table 4.1. L'ensemble de tâches est toujours faisable. Au premier abord, ces résultats montrent que l'impact de la rafale de fautes est moins important sur les tâches τ_2 et τ_3.

$\Delta_F = 50$						
P	T	C	D	$\mathcal{R}$	F	$\mathcal{R}^{\Delta_F}$
1	300	10	300	10	20	80
2	500	50	500	60	70	190
3	800	150	800	210	260	590

TABLE 4.2 – Exemple dans le cadre d'une stratégie ED/FR/M

Les résultats obtenus par la stratégie ED/FR/M sont meilleurs en terme de performances que dans le cadre d'une stratégie ED/FR/S. Ainsi, la valeur de $\mathcal{R}_3^{\Delta_F}$ calculée est réduite de 33% grâce à ce changement de stratégie.

Cette comparaison montre une différence d'efficacité entre les deux stratégies. Le gain de temps dû à la réduction du nombre de détections d'erreurs, dans le pire cas, est non négligeable. Cependant, l'évaluation du terme de recouvrement F_i présentée dans l'équation (4.7) correspond à un ensemble de tâches dans lequel la tâche de plus haute priorité aurait la valeur de C_i la plus grande. Si l'ensemble de tâches ne correspond pas à ce cas de figure, le terme F_i évalué sera surapproximé.

4.2.4 Raffinement dans le cas d'une stratégie multiple

Les expressions du terme de recouvrement F_i précédemment proposées se basent sur une approche sommant le maximum de temps dû aux détections d'erreurs et le maximum de temps dû aux réexécutions. Dans le cadre d'une stratégie multiple, ce raisonnement peut-être affiné. En effet, à la fin de la rafale de fautes, les tâches vont s'éxécuter suivant leur priorité initiale, la tâche de plus haute priorité éligible à l'instant considéré étant assignée au processeur. Ainsi, si une erreur est détectée dans la tâche en cours d'exécution, alors les réexécutions ne concernent que la tâche erronée, et celles de moins haute priorité.

4.2.4.1 Forme générale du terme de recouvrement F_i

Pour une tâche τ_i, la valeur du terme F_i dépend de la tâche τ_j de plus haute priorité dans laquelle l'erreur est détectée, du coût temporel de réexécution de cette tâche, et des tâches τ_k de plus basse priorité concernées ($j \leqslant P(\tau_k) \leqslant i$). Ainsi, pour chaque tâche τ_j de plus haute priorité que τ_i, il existe donc une séquence de détection (dont le coût temporel est égal à C_j^D) et de réexécutions associées.

Le raffinement proposé consiste donc à rechercher, parmi ces séquences, quelle est celle qui maximise le terme de recouvrement F_i. Le scénario ainsi défini est donc celui du pire cas. La distribution interne de fautes au sein de la rafale étant inconnue, la tâche de plus haute priorité éligible à la fin de la rafale est aussi inconnue. Le calcul de toutes les séquences possibles est donc nécessaire. L'équation (4.10) donne l'expression du terme F_i raffiné.

$$\forall (i > 1)\ F_i = \max_{j<i} \left(C_j^D + \sum_{k=j}^{i-1} \overline{C_k} \right) + \overline{C_i} \tag{4.10}$$

Si de la séquence maximale évaluée sont exclues une ou plusieurs tâches, deux interprétations sont possibles. Soit aucune instance des tâches concernées n'était éligible durant la rafale de fautes. Soit toute instance dont l'état était erroné à cause de la rafale de fautes a réussi à se terminer avec succès.

4.2.4.2 Cas de la stratégie ED/FR/M

Sachant que $C_i^D = C_i$ et $\overline{C_i} = C_i$ pour chacune des tâches, l'expression du terme F_i dans le cadre d'une stratégie ED/FR/M est donnée par l'équation (4.11)

$$\forall (i > 1)\ F_i = \max_{j<i} \left(C_j + \sum_{k=i-1}^{j} C_k \right) + C_i \tag{4.11}$$

Echéancier commenté La figure 4.6 décrit l'évaluation de $\mathcal{R}_3^{\Delta_F}$. Sur cet échéancier, le pire cas arrive lors de l'absence d'instance éligible de τ_1 à la fin de la rafale de fautes. Cela signifie que l'instance précédente s'est exécutée avec succès. La tâche τ_2 possède le plus long temps de réexécution $\overline{C_2}$. Á la fin de la rafale, une instance de cette tâche est erronée, et dans le pire cas, la détection d'erreur arrive au bout de $\overline{C_2}$ unités de temps. Depuis cette détection, les réexécutions complètes des tâches τ_2 et τ_3 s'effectuent. La conservation des priorités initiales autorise les préemptions des réexécutions par les instances de la tâche τ_1.

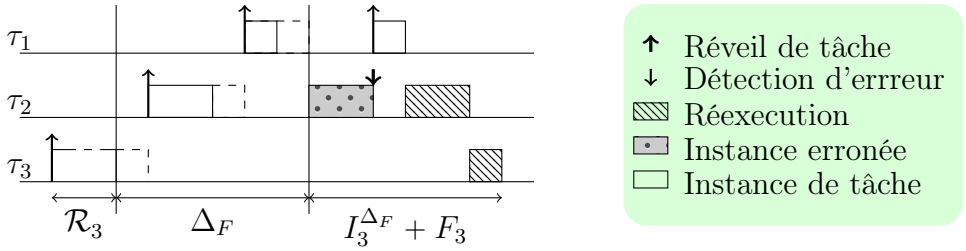


FIGURE 4.6 – Evaluation de $\mathcal{R}_3^{\Delta_F}$ sous la stratégie ED/FR/M raffinée

Exemple La table 4.3 contient les résultats de l'évaluation raffinée du terme F_i sur l'exemple de la table 4.2. La faisabilité de l'ensemble de tâches n'est pas remise en cause.

Le gain est observé sur la tâche τ_3. Ce gain signifie que la tâche τ_1 n'est pas prise en compte dans l'évaluation du terme de recouvrement F_3. Le résultat obtenu, compte tenu

$\Delta_F = 50$						
P	T	C	D	$\mathcal{R}$	F	$\mathcal{R}^{\Delta_F}$
1	300	10	300	10	20	80
2	500	50	500	60	70	190
3	800	150	800	210	250	580

TABLE 4.3 – Exemple dans le cadre d'une stratégie ED/FR/M raffinée

de la surapproximation induite par l'équation (4.9) exprimée plus haut, est donc plus précis.

4.3 Etude quantitative

L'étude quantitative décrite dans cette section a plusieurs objectifs. Dans un premier temps, il s'agit d'appliquer les diverses stratégies décrites a de nombreuses configurations de tâches afin d'évaluer la tolérance aux rafales de fautes des ordonnancements. Ensuite, il s'agit d'obtenir des résultats à même de comparer les stratégies entre elles.

La section se structure comme suit. La sous-section 4.3.1 décrit le protocole de simulation retenu. Les résultats obtenus sont ensuite décrits dans la sous-section 4.3.2, et commentés dans la sous-section 4.3.3.

4.3.1 Protocole

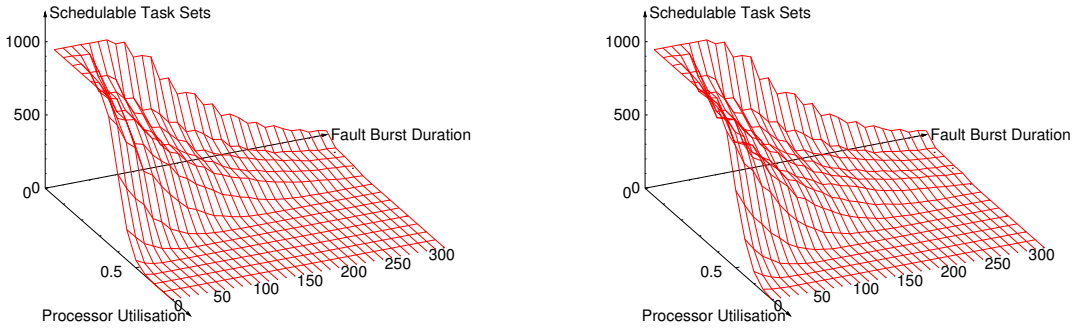
Le protocole de simulation se base sur la génération aléatoire et l'analyse de plusieurs milliers d'ensembles de tâches. Plus précisément, pour une variation de 5% du taux d'utilisation du processeur, 1000 ensembles de 10 tâches sont générés. L'hypothèse de base est leur faisabilité en l'absence de fautes. Les échéances des tâches sont inférieures ou égales à la période. Ce protocole de simulation est classique et couramment utilisé.

Le taux d'utilisation du processeur est compris entre 5% et 95%. L'assignation d'une priorité aux tâches est faite en utilisant Rate Monotonic. Les rafales de fautes varient depuis 10% de la plus longue période de tâche, jusqu'à 35%. L'accroissement de la durée de la rafale est faite suivant un pas de 5%. Les durées de ces rafales de fautes sont cohérentes avec les recommandations [SR01]. Pour chaque durée de rafales de fautes, les trois stratégies précédemment décrites sont appzliquées.

4.3.2 Résultats

Les résultats obtenus sont présentées sous la forme d'une courbe 3D. L'axe X représente le taux d'utilisation, l'axe Y la durée de la rafale de fautes et l'axe Z le nombre

d'ensembles de tâches faisables. La figure 4.7(a) décrit les résultats obtenus avec l'application de la stratégie ED/FR/S, et la figure 4.7(b) ceux obtenus avec la stratégie ED/FR/M et l'évaluation raffinée du terme de recouvrement F_i.



(a) Stratégie ED/FR/S (b) Stratégie ED/FR/M

FIGURE 4.7 – Courbe 3D obtenues par simulation

Ces deux figures montrent en premier lieu que le nombre de configurations faisables décroit avec l'augmentation de la durée des rafales pour un taux d'utilisation fixé. Le constat est identique pour une durée de rafales fixée et une augmentation du taux d'utilisation.

L'exploitation de ces résultats est faite autour de quelques points particuliers. Pour une durée de rafales nulle, le cas est équivalent à celui de fautes pseudo-périodiques tel que $T_F > \max(D_i)$. Une rafale de fautes de durée nulle est vue comme une faute instantanée, un dirac qui impacte l'ensemble de tâches. Les résultats de cette simulation montrent que l'application de la stratégie ED/FR/S permet de prouver la faisabilité de certains ensembles de tâches jusqu'à un taux d'utilisation de 55%. Dans le cadre d'une stratégie ED/FR/M, ce résultat est amélioré et borne le taux d'utilisation à 65%.

La considération d'un taux d'utilisation égal à 50% est un autre point intéressant. Les travaux de [PM98] prouvent que cette valeur est une condition suffisante pour valider un ensemble de tâches sous l'hypothèse d'une faute pseudo-périodique telle que $T_F > \max(D_i)$. Puisque la stratégie de recouvrement considérée est ED/FR/M, la comparaison entre ces travaux de thèse et ce résultat est naturelle. Premièrement, la simulation montre que des ensembles de tâches sont validés pour des taux d'utilisation supérieur à 50 % pour une durée de rafale nulle, *id est* une seule faute ponctuelle. Dans ces cas là, la condition suffisante ne permet pas de conclure. L'approche proposée dans ces travaux permet de dépasser cette limitation.

Ensuite, l'application de la stratégie ED/FR/M permet de valider, pour un taux d'utilisation de 50%, des ensembles de tâches subissant une rafale de fautes allant jusqu'à 14% de la plus longue des périodes. La méthode de validation décrite permet donc de prouver la

faisabilité d'un ensemble de tâches pour des durées significatives. A titre de comparaison, dans cette simulation, la rafale de fautes ne dépasse pas 3% de la plus longue des périodes via l'application d'une stratégie ED/FR/S. Cette dernière est donc moins efficace qu'une stratégie ED/FR/M en présence d'une rafale de fautes.

4.3.3 Commentaires

La simulation montre le gain d'efficacité obtenu par l'emploi de la stratégie ED/FR/M par rapport à la stratégie ED/FR/S. Le caractère de dominance de la stratégie ED/FR/M permet donc une réelle amélioration de la tolérance aux fautes en présence de rafales. Pour tout couple taux d'utilisation et durée de la rafale, le nombre de configurations faisable sous la stratégie ED/FR/M est égal ou supérieur à celui sous la stratégie ED/FR/S.

La figure 4.8 représente la projection des résultats obtenus par les deux stratégies pour une durée de rafale de fautes égale à 10% de la plus longue des périodes de tâches. L'axe X est le taux d'utilisation et l'axe Y le nombre d'ensemble de tâches faisables. Les deux courbes décroissent continûment lorsque le taux d'utilisation du processeur augmente. Dans le cadre des ensembles aléatoires générés pour cette étude quantitative, l'efficacité des deux stratégies est équivalente jusqu'à un taux d'utilisation de 30 %. L'efficacité de la stratégie ED/FR/M s'affirme à partir de cette valeur.

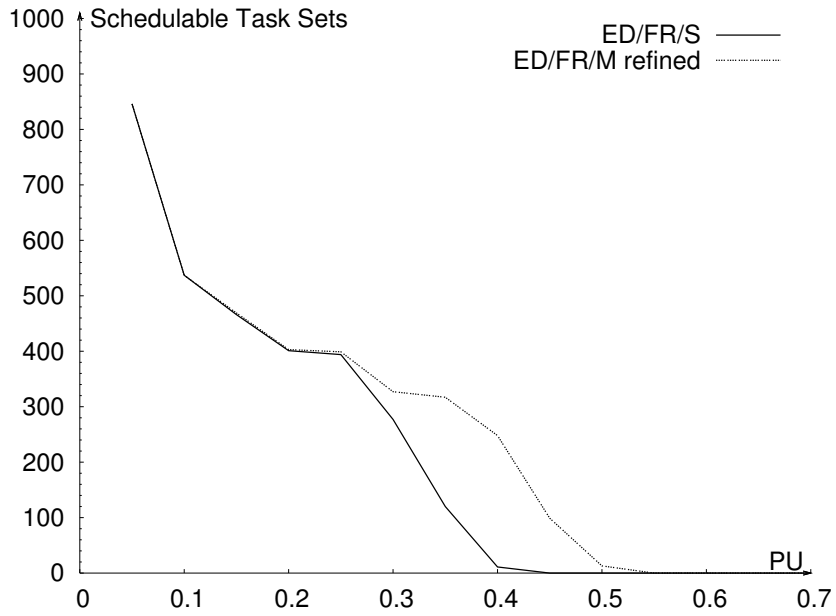
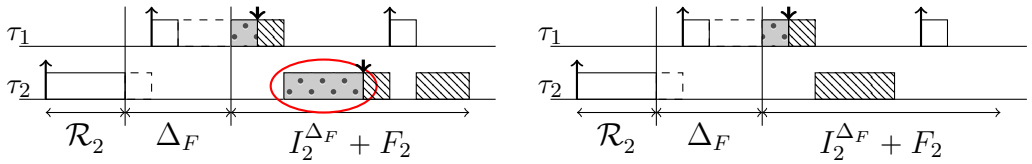


FIGURE 4.8 – Projection des résultats obtenus pour une rafales de fautes de 10%

Du point de vue qualitatif, l'origine de ce gain d'efficacité est décrit par les schémas de la figure 4.9. La détection d'erreurs pour la tâche τ_2 mise en évidence sur la figure

4.9(a) est absente sur la figure 4.9(b). Le principe de la stratégie ED/FR/S se base sur n détections et n corrections. Seules les tâches erronées sont réexécutées. Pour la stratégie ED/FR/M, ce principe devient 1 détection et n corrections.



(a) Stratégie ED/FR/S (b) Stratégie ED/FR/M

FIGURE 4.9 – Explication qualitative de l'efficacité de la stratégie ED/FR/M

Lors de l'exécution des tâches, les réexécutions préventives peuvent concerner des tâches qui ne contiennent pas d'erreur. L'évaluation du temps de réponse $\mathcal{R}^{\Delta_F}$ se basant sur le pire scénario, ces corrections anticipées améliorent l'approximation de $\mathcal{R}^{\Delta_F}$ par la limitation du nombre de détection d'erreurs. Ainsi, du point de vue temporel, une augmentation du nombre de réexécutions inutiles est introduite. Pourtant, du point de vue de la validation, cette approximation du $\mathcal{R}_i^{\Delta_F}$ est plus précise, et le nombre de configurations ordonnançables augmente. Le temps de réponse moyen est donc dégradé au profit du pire temps de réponse.

4.4 Conclusion

Ce chapitre a établi une équation conservatrice du temps de réponse sous rafales de fautes. Cette équation se construit comme la somme des contributions « pire cas » des temps de réponse sans fautes, des interférences avec les autres tâches après la fin de la rafale de fautes et de l'impact des stratégies de recouvrement. Cette approche permet d'évaluer indépendamment ces différentes contributions.

A ce titre, le terme F_i évalue le surcoût temporel dû au déclenchement de la stratégie de recouvrement d'erreurs. Ce terme F_i dépend de la stratégie considérée, de la tactique de recouvrement d'erreurs et du nombre de tâches. Suivant le contexte, les équations générales (*id est* indépendantes de la tactique condidérée) ont été décrites pour la stratégie simple, la stratégie multiple et celle d'une stratégie multiple plus précise. Puis des exemples d'équations dépendantes de la tactique de recouvrement d'erreurs ont été montrés. Les analyses d'ordonnançabilité peuvent alors être menées.

Pour évaluer l'efficacité des différentes stratégies, une étude quantitative a été conduite. De cette étude, il ressort que les stratégies sont équivalentes jusqu'à une charge du processeur égale à 30%. A partir de cette valeur, les stratégies multiples deviennent plus efficaces, malgré la possibilité de réexécuter des tâches qui ne contiennent pas d'erreurs.

L'autre résultat majeur concerne la démonstration de faisabilité de certains ensembles de tâches dont la charge processeur correspondante est supérieure à 50% (suivant la condition suffisante de [PM98]).

In fine, ce chapitre a donc montré comment mener des analyses d'ordonnançabilité pour des ensembles de tâches soumis à des rafales de fautes, suivant plusieurs formes de stratégies de recouvrement d'erreurs. Ces résultats peuvent être étendus au cas particulier des ordonnancements partitionnés, ce qui est l'objet du chapitre suivant.

Chapitre 5

Application aux ordonnancements partitionnés

Sommaire

La problématique des ordonnancements partitionnés est liée à l'émergence dans le monde aéronautique des architectures modulaires intégrées (*Integrated Modular Avionics* - IMA) [AEE06]. Une architecture modulaire intégrée se définit comme un ensemble de ressources matérielles et logicielles partagées. Ces ressources forment une plateforme fournissant des services aux applications qu'elles hébergent.

La mutualisation des ressources de calcul et de communication pour l'ensemble des fonctions intégrées impose des contraintes de sûreté de fonctionnement et de temps réel. Ainsi, le comportement défectueux d'une fonction ne doit ni affecter le comportement des autres fonctions hébergées (non interférence) ni propager une faute (isolation). Les propriétés de non interférence et d'isolation sont garanties par la définition de *partition*, qui héberge une fonction. Ce partitionnement est à la fois spatial et temporel [AEE06].

Le partitionnement spatial assure que l'exécution des différentes tâches représentant les fonctions s'effectue sur des zones mémoires prédéterminées. Les accès en mémoire hors d'une partition sont interdits par le *Memory Management Unit* (MMU)[AEE06].

Le partitionnement temporel est fait via l'APEX (*Application/Executive*) [AEE06], qui définit les interfaces entre les tâches allouées aux partitions et le système d'exploitation. L'APEX exige deux niveaux d'ordonnancement : un pour les partitions (niveau système d'exploitation), et un autre pour les applications (niveau partition). L'ordonnancement

des partitions doit être fait suivant un ordonnanceur cyclique déterministe. L'ordonancement des applications suit un ordonnancement à priorité fixe. Les applications ne peuvent s'exécuter que pendant leur partition.

La validation des ensembles de tâches hébergées sur les partitions constitue donc la **théorie des ordonnancements partitionnés**. Dans les études d'ordonnançabilité correspondantes, le partitionnement spatial est supposé correct.

Ce chapitre se structure comme suit. La section 5.1 présente les principaux résultats sur la validation d'ordonnancements partitionnés. Puis la section 5.2 décrit la prise en compte du modèle en rafales de fautes dans ces analyses d'ordonnançabilité. Enfin la section 5.3 conclut ce chapitre.

5.1 État de l'art

L'état de l'art de la théorie des ordonnancements partitionnés propose des techniques de validation pour les partitions à fenêtre d'exécution unique et les partitions à fenêtres d'exécution multiples. Cette section se structure donc autour de ces deux types de partitions et des techniques de validation qui leur sont consacrées.

5.1.1 Fenêtre d'exécution unique

Les travaux de [AW96] définissent l'ensemble des contraintes à prendre en compte dans la validation d'ordonnancements partitionnés à deux niveaux. Chaque partition, notée π, possède une criticité (suivant la fonction hébergée), une période d'activation et une durée d'exécution composée d'une fenêtre temporelle unique. L'APEX exigeant un ordonnancement cyclique déterministe pour les partitions, leur ordre d'exécution est donc défini hors ligne et représente une *Major Time Frame* (MTF). La MTF est ensuite répétée indéfiniment.

Chaque partition π héberge un groupe de tâches, noté $TG(\pi)$. Ces travaux proposent de valider chacun de ces groupes en évaluant le pire temps de réponse $\mathcal{R}_i^\pi$ de chaque tâche τ_i ($\tau_i \in TG(\pi)$). L'équation de $\mathcal{R}_i^\pi$ s'obtient en ajoutant à l'équation du pire temps de réponse sans fautes (équation (1.3)) les termes suivants :

- G_i, temps de blocage dû à la gigue des partitions
- $OS_i(0, \mathcal{R}_i)$, coût des préemptions
- $max\left(0, \left\lceil \frac{\mathcal{R}^\pi - O_0}{T_0} \right\rceil C_0\right)$, indisponibilité du processeur dûe à l'exécution des autres partitions

L'équation (5.1) donne l'expression du pire temps de réponse $\mathcal{R}_i^\pi$:

$$\mathcal{R}_i^\pi = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\mathcal{R}_i}{T_j} \right\rceil C_j + B_i + G_i + OS_i(0, \mathcal{R}_i^\pi) + max\left(0, \left\lceil \frac{\mathcal{R}_i^\pi - O_0}{T_0} \right\rceil C_0\right) \quad (5.1)$$

Dans ces travaux, les caractéristiques temporelles des partitions (période d'activation et une durée d'exécution) sont supposées connues. L'étude menée par [Lee+98] complète ces précédents travaux en proposant de déterminer les caractéristiques temporelles des partitions. La méthode proposée définit les exigences d'ordonnançabilité des partitions, puis la construction du cycle (approche cycle unique, ou harmonique). L'ordonnançabilité des tâches est faite via la méthode de la demande de processeur. Comme chaque partition n'accède qu'à une fraction de la demande de processeur, la méthode employée est celle du processeur lent, déjà abordée par [DL97], dans le cadre d'ordonnancement partitionné en environnements dits ouverts (contraintes moins importantes que dans le cas de l'IMA). Ces travaux sont complétés par la définition d'un outil de validation [Lee+00].

5.1.2 Fenêtres d'exécution multiples

Les travaux précédemment décrits prennent un compte des partitions avec une fenêtre unique. L'étude menée par [MFC01] permet la prise en compte de partitions contenant des fenêtres d'exécutions multiples. Une partition π est alors définie comme un couple (Γ, P) tel que :

- $\Gamma = \{(S_1, E_1), ..., (S_N, E_N)\}$ où les couples (S_i, E_i) sont les fenêtres d'exécution.
- P est la période d'activation de la partition π.

A chaque activation de la partition π, chaque fenêtre d'exécution possède un début d'exécution S_i, représentant son décalage relatif dans le temps depuis le début de la partition, et une fin d'exécution E_i. Ainsi, les différentes tâches du groupe hébergé s'exécutent sur ces fenêtres. Ces travaux permettent de valider des groupes de tâches ordonnancés suivant des priorités fixes (cas décrit ci-après) ou dynamiques [MFC01].

La faisabilité d'un groupe de tâches d'une partition π nécessite la connaissance de la fraction de processeur allouée à la partition, ou l'évaluation à un instant donné du temps d'éxécution cumulé. Les partitions considérées étant à fenêtre d'exécution multiple, la première solution fournit un résultat global insuffisant. En effet, la fraction du processeur ne tient pas compte de la répartition temporelle des fenêtres d'exécution sur la partition. Pour prendre en compte ce problème, une fonction de support S (*Supply Function*) est définie :

Définition 3 *Fonction de support S [MFC01]*
La fonction de support S d'une partition π évalue pour chaque instant t le temps d'exécution cumulé de la partition π depuis l'instant $t = 0$.

Par construction, la fonction de support S est une fonction croissante monotone. D'autres propriétés sont démontrées :

Propriété 2 *Propriétés de la fonction S [MFC01]*

- $S(0) = 0$

- $S(u) - S(v) \leqslant u - v$, *pour* $u > v \geqslant 0$
- $S(t + P) - S(t) = S(P)$ *pour* $t \geqslant 0$
- $S(t) = \left\lfloor \frac{t}{P} \right\rfloor \times S(P) + S(t - P \left\lfloor \frac{t}{P} \right\rfloor)$ *pour* $t > 0$

Les travaux de [MFC01] montrent que la notion d'instant critique, telle que définie par [LL73], doit être reconsidérée pour évaluer le pire scénario d'exécution d'une tâche τ_i. En effet, l'intuition serait de considérer que l'instant critique correspond à l'instant t tel que toutes les tâches du groupe sont relachées simultanément et que le plus long intervalle d'indisponibilité du processeur débute. Cette intuition s'avère fausse, ce qui conduit à la nécéssité de considérer plusieurs instants critiques, et à vérifier que le groupe de tâches est faisable sur chacun d'entre eux. Ces instants critiques sont définis comme suit :

Définition 4 *Instant critique basé sur un intervalle [MFC01]*

Pour une partition $\pi = (\{(S_1, E_1), ..., (S_N, E_N)\}, P)$, *les instants* E_i *représentent des points critiques basés sur un intervalle. Si une tâche est réveillée simultanément avec toutes les tâches de plus haute priorité, alors* E_i *est un instant critique basé sur un intervalle.*

Pour un ordonnancement à priorités fixes, la démonstration d'ordonnançabilité de chaque tâche doit être faite sur chacun des instants critiques E_i de la partition π (théorème 1).

Théorème 1 *[MFC01]*

Une tâche τ_i $(D_i \leqslant T_i)$ *appartenant à un groupe de tâches est ordonnançable sur la partition* π *si et seulement si elle est ordonnançable sur tous les instants critiques basés sur un intervalle de* π.

Pour chaque instant critique E_i, le pire temps de réponse $\mathcal{R}_i^\pi$ de la tâche τ_i est évalué. Ce pire temps de réponse dépend du temps d'exécution cumulé par la partition π depuis E_i. Ainsi, pour une valeur $\mathcal{R}_i^\pi$, la valeur de t telle que $\mathcal{R}_i^\pi = S(E_i+t)-S(E_i)$ est recherchée. Si t est supérieur à l'échéance D_i, alors la tâche n'est pas ordonnançable. En effet, cela signifie que pour obtenir un temps d'exécution cumulé égal à $\mathcal{R}_i^\pi$, le temps écoulé est supérieur à l'échéance relative. Dans le cas contraire, la tâche τ_i est ordonnançable.

5.2 Partitions soumises à des rafales de fautes

Ces travaux proposent une méthode de validation d'un groupe de tâches alloué à une partition π, dans le cadre d'un système temps réel soumis à une rafale de fautes. Cette méthode s'appuie sur une adaptation de l'équation (4.2) au contexte des ordonnancements partitionnés, sur la fonction de support S et sur le théorème 1.

5.2.1 Equation de pire temps de réponse $\mathcal{R}^{\pi,\Delta_F}$

L'évaluation du pire temps de réponse d'une tâche τ_i sous rafales de fautes couple les problèmes déjà décrit pour l'ordonnancement tolérant aux fautes avec ceux liés aux multiples fenêtres d'exécution d'une partition π. Ainsi, l'établissement d'une équation conservative est nécessaire pour fournir une méthode de validation adaptée au contexte.

Le théorème 1 implique que N calculs de pire temps de réponse sont nécessaires afin de valider une tâche pour chaque instant critique basé sur un intervalle de π. En conséquence, il existe donc N temps de réponse calculés, notés $\mathcal{R}_i^{E_k}$. Suivant la logique qui a conduit à l'établissement de l'équation (4.2), l'hypothèse que la rafale de fautes apparaît à la fin du pire temps de réponse $\mathcal{R}$ est conservée. Ici, ce pire temps de réponse, noté $\mathcal{R}_i^{\pi}$, représente le maximum des temps de réponse évalués sur les N instants critiques de π (équation (5.2)).

$$\mathcal{R}_i^{\pi} = \max_{k \in [\![1;N]\!]} (\mathcal{R}_i^{E_k}) \tag{5.2}$$

Le reste de l'expression est inchangé, puisque le comportement du système temps réel est toujours supposé inconnu pendant la durée Δ_F de la rafale de fautes, qu'il est toujours nécessaire de borner le nombre d'interférences dues aux tâches de plus haute priorité après la fin de la rafale, et enfin, d'évaluer le terme F_i. L'expression est donnée par l'équation

$$\mathcal{R}_i^{\pi,\Delta_F} = \mathcal{R}_i^{\pi} + \Delta_F + I_i^{\Delta_F} + F_i \tag{5.3}$$

Dans l'équation (4.2), un nouvel instant critique depuis l'instant $\mathcal{R}_i + \Delta_F$ était défini. Hors, les travaux de [MFC01] montrant qu'il existe N instants critiques, le théorème 1 est toujours valide. De fait, depuis l'instant $\mathcal{R}_i^{\pi} + \Delta_F$, afin d'assurer la propriété de conservatisme de l'équation (5.3), il est nécessaire d'évaluer pour chaque E_k la valeur de $I_i^{\Delta_F} + F_i$.

5.2.2 Algorithme de validation

L'algorithme 1 montre une méthode de validation d'une tâche τ_i allouée à une partition π_i. Cet algorithme évalue, pour chaque tâche, la laxité $L_i = D_i - (\mathcal{R}_i^{\pi} + \Delta_F)$. Puisque chaque terme de cette expression est calculé indépendamment, la laxité L_i est constante, et représente le temps de préemptions dûes aux tâches de plus haute priorité du groupe TG et au surcoût temporel dû à la stratégie de recouvrement.

Pour chaque instant critique E_k basé sur un intervalle de la partition π, ce temps supplémentaire de calcul, noté R', correspond à un temps processeur évalué par la fonction S. Si ce temps processeur est supérieur à la laxité, alors la tâche n'est pas ordonnançable sur E_k. Il n'existe alors pas de temps processeur, dans le temps donné L_i, capable de gérer le surcoût temporel R'.

En conclusion, l'évaluation du temps R' permet d'évaluer le pire temps de réponse $\mathcal{R}_i^{E_k}$ suivant l'équation (5.3) pour chaque E_k. Le maximum de ces valeurs représentent

Algorithme 1 Validation d'une tâche τ_i sur la partition π sous rafales de fautes Δ_F

$L_i = D_i - (\mathcal{R}_i^{\pi} + \Delta_F)$
pour tout $E_k,\ k \in [\![1; N]\!]$ **faire**
  $R \longleftarrow F_i$
  loop
    $R' \longleftarrow \sum_{hp(i)} \left\lceil \frac{R}{T_j} \right\rceil C_j + F_i$
    Evaluer x tel que $R' = S(E_k + x) - S(E_k)$
    si $x > L_i$ **alors**
      retour Echec
    si $R' = x$ **alors**
      Sortie de boucle % *Valide sur* E_k
    sinon
      $R \longleftarrow x$
  retour Succes

donc le pire temps de réponse $\mathcal{R}_i^{\pi,\Delta_F}$. De la même façon que l'équation (4.2) est potentiellement pessimiste, la méthode de validation fournit seulement une condition suffisante pour valider un groupe de tâches alloué à une partition et soumis à une rafale de fautes de durée Δ_F.

5.2.3 Exemple illustratif

Dans l'exemple illustratif suivant, un groupe de trois tâches est alloué à la partition $\pi = (\Gamma, 100)$ tel que $\Gamma = \{(0, 20), (40, 70), (90, 95)\}$. Ce groupe de tâches est soumis à une rafale de fautes $\Delta_F = 20$ unités de temps. La table 5.1 en décrit les principales caractéristiques temporelles, et donne les temps de réponses obtenus en l'absence de fautes suivant les différents instants critiques basés sur un intervalle E_k. Suivant l'algorithme d'ordonnancement *Rate Monotonic*, chaque tâche est ordonnançable et le groupe de tâches est faisable.

				$\mathcal{R}_i^{E_k}$		
P	T	C	D	E_1	E_2	E_3
1	95	5	95	35	25	10
2	210	20	210	45	50	50
3	480	50	480	145	150	160

TABLE 5.1 – Calculs de $\mathcal{R}_i^{E_k}$ pour un ordonnancement partitionné

De la table 5.1 sont déduits les temps de réponse maximum suivant : $\mathcal{R}_1^{\pi} = 35$, $\mathcal{R}_2^{\pi} = 50$ et $\mathcal{R}_3^{\pi} = 160$. L'algorithme de validation d'une tâche τ_i sur la partition π sous une rafale

de fautes Δ_F est ensuite appliqué. Deux stratégies sont étudiées, la stratégie ED/FR/S (sous-table 4.5(a)) et la stratégie ED/FR/M raffinée (sous-table 4.5(b)). Pour chaque tâche, la laxité L_i est évaluée, et puisqu'elle dépend de $\mathcal{R}_i^\pi$, de l'échéance D_i et de la durée de la rafale de fautes, elle est constante dans les deux applications des stratégies.

(a) Calculs pour une stratégie ED/FR/S

			L suivant E_k			
P	L	F	E_1	E_2	E_3	Rés.
1	40	10	30	35	15	O
2	140	50	125	125	110	O
3	300	150	395	395	370	¬O

(b) Calculs pour une stratégie ED/FR/Mr

			L suivant E_k			
P	L	F	E_1	E_2	E_3	Rés.
1	40	10	30	35	15	O
2	140	45	95	95	75	O
3	300	90	275	280	250	O

TABLE 5.2 – Résultats pour un ordonnancement partitionné soumis à une rafale de fautes

Une première analyse montre que dans cet exemple, le groupe de tâches est faisable avec la stratégie ED/FR/M raffinée et non faisable avec la stratégie ED/FR/S. En effet, la tâche τ_3 ne peut être corrigée dans le temps de laxité L_3 dans le pire cas, en tenant compte des préemptions dues aux tâches de plus haute priorité et de son terme de recouvrement F_3. En revanche, cette tâche devient ordonnançable sous la stratégie ED/FR/M raffinée (Gain temporel de 30%).

Une seconde analyse, plus qualitative, montre que la dominance de la stratégie ED/FR/M raffinée sur la stratégie ED/FR/S se renforce dans le contexte des ordonnancements partitionnés. En effet, du point de vue du groupe de tâches alloué à la partition π, le processeur n'est accessible que 55% du temps. De fait, cet accès limité à la ressource de calcul est pénalisant dans le cas d'une stratégie simple, puisque dans le pire cas, toutes les tâches du groupe nécessitent deux fois leur temps d'exécution C_i pour être corrigée. Qualitativement, la limitation du nombre de détection est importante pour améliorer les temps de réponse des tâches au sein d'un groupe de tâches alloué à une partition.

5.3 Conclusion

Ce chapitre s'est intéressé à l'étude des ordonnancements partitionnés soumis à des rafales de fautes. Une équation de pire temps de réponse, dérivée de celle présentée au chapitre 4, est proposée. La particularité de cette équation est d'intégrer un pire temps de réponse sans fautes qui est en réalité le maximum de tous les pires temps de réponse évaluables. En effet, dans le cadre d'un ordonnancement partitionné, le pire temps de réponse dépend de la fenêtre d'exécution.

L'analyse d'ordonnançabilité proposée se fonde sur un algorithme de validation. Cet algorithme de validation est dérivé de celui proposé par [MFC01], le critère d'arrêt étant

soit la convergence du calcul de pire temps de réponse, soit un dépassement du temps restant disponible de la tâche, basée sur la laxité L_i.

Ce chapitre a donc permis l'étude des ordonnancements partitionnés soumis à des rafales de fautes. Cette étude trouve sa pertinence dans l'émergence des technologies basées sur les architectures modulaires intégrées. Ces technologies permettent d'héberger des fonctions « avioniques » initialement allouées à des architectures dédiées sur un ensemble de ressources mutualisées de calculs. Pour conserver l'indépendance fonctionnelle des différentes fonctions hébergées, la démonstration du partitionnement temporel, même sous des perturbations extérieures, est fondamentale.

Ce chapitre clôt la partie consacrée aux ordonnancements temps réel tolérants aux fautes. Dans la partie suivante, la thématique abordée concerne la robustesse des ordonnancements. L'étude de la robustesse des ordonnancements est essentiellement fondée sur la recherche de métriques permettant de caractériser la capacité d'un ordonnancement à tolérer des fautes.

Troisième partie

Robustesse des ordonnancements

Chapitre 6

Introduction à la robustesse des ordonnancements

Sommaire

Les analyses présentées jusqu'à présent partent du postulat que les caractéristiques du modèle de distribution de fautes sont connues, que les moyens de protection et de gestion d'erreurs sont choisis, et que les analyses statiques d'ordonnançabilité intégrant ces mécanismes de tolérance aux fautes sont effectuées. Ces différentes hypothèses sont tout à fait cohérentes avec une démarche de conception et de validation de système temps réel. Ainsi, les caractéristiques du modèle de distribution de fautes peuvent être issues d'expérimentation et de retour d'expérience, et leur prise en compte dirige le choix des mécanismes de tolérance aux fautes, et par ricochet, la technique de validation.

Ce chapitre propose de s'écarter de cette démarche. En premier lieu, si la forme générale des agressions dûes à l'environnement est connue, ces caractéristiques propres peuvent ne pas l'être. En second lieu, la plateforme d'accueil du système informatique peut être imposée, par exemple pour des raisons économiques (cas des *Commercial Off The Shelf* - COTS) et donc ne pas bénéficier d'un développement dédié. Dans ce cas, seuls les mécanismes de tolérance aux faites présents au sein du COTS seront disponibles.

Enfin, une fois qu'un système temps réel est spécifié et produit pour un ensemble de missions, les utilisateurs peuvent souhaiter l'affecter à d'autres missions, dont les environnements de fonctionnement peuvent changer et sortir du cadre de conception initial. Ainsi, pour un concepteur ou un utilisateur, disposer de la capacité à évaluer les limites d'un système, en fonction de plusieurs paramètres, peut se révéler essentiel.

Ce chapitre aborde la question de la **robustesse des ordonnancements** et présente les approches issues de l'état de l'art. Ce chapitre se structure comme suit. La section 6.1 introduit la problématique des métriques de cette robustesse. Puis la section 6.2 décrit l'approche probabiliste et la section 6.3 montre l'approche statistique. Enfin la section 6.4 conclut ce chapitre.

6.1 Définir la robustesse des ordonnancements

La robustesse des ordonnancements temps réel peut se définir comme la capacité maximale d'un ordonnancement à gérer les erreurs internes des tâches, et à continuer à assurer le respect des échéances temporelles. La problématique générale repose sur les métriques permettant d'évaluer et de comparer la robustesse des systèmes temps réel.

Une première approche est de considérer que la robustesse dépend directement de l'environnement de fonctionnement du système temps réel. Cette approche aboutit donc à considérer que ce sont les modèles de fautes, qualifiables de limite, qui définissent la métrique de la robustesse. Ainsi, pour un modèle de fautes donné, la recherche des paramètres limites maximisant les effets néfastes sur l'ensemble de tâches est la ligne directrice. La section 6.2 part de cette idée, pour aboutir à la notion de garantie probabiliste.

Une seconde approche considère que l'agression par l'environnement n'est pas une métrique pertinente. L'objectif de cette seconde approche est de construire des scénarios de simulation, et de les soumettre à des erreurs aléatoires. L'idée centrale est alors la recherche des scénarios dans lesquel les effets des erreurs sont maximum. La section 6.3 montre l'emploi de cette approche, qui aboutit à l'établissement d'un niveau de confiance dans le système temps réel.

6.2 Approche probabiliste

Les travaux de [Bur+99 ; Pun97] proposent la notion de **garantie probabiliste** pour caractériser la robustesse d'un ordonnancement tolérant aux fautes. L'idée directrice est d'évaluer la probabilité qu'une ou plusieurs tâches ratent leurs échéances dans des conditions limites données. Le modèle de fautes concerné par ces travaux est le modèle de fautes pseudo-périodique.

Les conditions limites sont définies comme la valeur T_{max} de la pseudo-période telle que l'ensemble de tâches ne soit plus faisable. La recherche de cette valeur est effectuée par la méthode de **fonction de sensibilité** [Ves94]. Une fonction de sensibilité permet d'évaluer la variation possible des différents paramètres des tâches telle que l'ensemble de tâches soit toujours ordonnançable. Par exemple, cette fonction de sensibilité peut porter sur le pire temps d'exécution C_i, et aboutir aux raisonnements suivants pour un ensemble de tâches Γ :

1. Si Γ est faisable, quelle augmentation de C_i est autorisée pour chaque tâche sans que la faisabilité soit remise en cause.
2. Si Γ est non faisable, quelle diminution de C_i doit être appliquée à chaque tâche pour que Γ soit faisable.

Dans le cas des travaux de [Bur+99 ; Pun97], la fonction de sensibilité définie concerne le nombre maximal d'erreurs que peut tolérer un ordonnancement soumis à des fautes

pseudo-périodiques, traduit par le terme T_{max}.

Une fois ce terme évalué, une probabilité d'échec de l'ordonnancement peut être établie. La distribution temporelle des erreurs est définie suivant une loi de poisson, le paramètre λ définissant l'intervalle de temps entre deux erreurs. Ainsi, la garantie probabiliste évalue donc la probabilité qu'un ordonnancement soit non faisable sachant que $\lambda \geqslant T_{max}$

6.3 Approche statistique

Dans les travaux de [LN09 ; LNL10], la robustesse des ordonnancements tolérants aux fautes est qualifiée de **résilience aux fautes**. L'évaluation de cette résilience aux fautes suit une approche statistique, c'est à dire cherche à calculer un intervalle de confiance [Tri08], auquel appartient le nombre de fautes affectant l'ensemble de tâches tel qu'il ne soit plus faisable.

La ligne directrice de ces travaux est la définition d'une métrique indépendante du modèle de fautes. Cette métrique permet la comparaison de différents systèmes temps réel du point de vue de la résilience aux fautes. Ces travaux sont motivés par les difficultés de comparaison des techniques issues de l'état de l'art. En premier lieu, les résultats prouvés pour un ordonnancement à priorités fixes sont non applicables à un ordonnancement à priorités dynamiques. Ensuite, deux techniques similaires ne sont pas comparables si le modèle de fautes change. Enfin, la plupart des techniques donnant des conditions suffisantes pour un modèle de fautes donné, la violation du scénario n'entraîne pas nécessairement une défaillance du système temps réel.

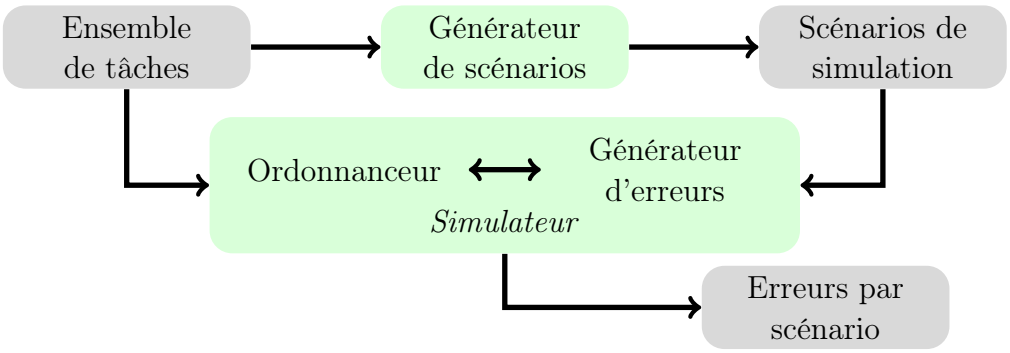


FIGURE 6.1 – Environnement de simulation des travaux de [LN09 ; LNL10]

Les travaux de [LN09 ; LNL10] mesurent la résilience aux fautes en se basant sur la simulation des ensembles de tâches (figure 6.1). Puisque simuler tous les scénarios possibles est une tâche trop complexe et possède une combinatoire très élevée, le choix est fait de générer aléatoirement des scénarios de simulation. Pour n tâches, un scénario de simulation est $n - uplet$ représentant une série de dates de réveil, chaque tâche étant

associée à une date de réveil. Ensuite, deux actions sont accomplies. La première est de reconstituer le contexte d'exécution de chaque tâche, c'est à dire de déterminer un instant critique t avant la date de réveil afin de maximiser les interférences dues aux tâches de plus hautes priorités (*backlog*).

La seconde action est de simuler l'ensemble de tâches. Cette simulation se fait sur $[t; a_i + D_i)$, car dans cette approche, seule une instance de la tâche τ_i, correspondant au scenario de simulation, est étudiée. Pendant la simulation, un générateur injecte des erreurs dans les différentes tâches avec pour objectif de faire échouer la tâche τ_i depuis t. Ainsi, plus l'effort du générateur d'erreurs est important (nombre d'erreurs injecté, recherche des instants d'injection d'erreurs), plus la tâche est résiliente aux fautes. Sur cette base, pour chaque instance de tâche du scénario, la résilience aux fautes est évaluée.

Dans le but d'évaluer l'intervalle de confiance, un échantillon aléatoire de scénarios de simulation est généré. Sur la base du nombre d'erreurs conduisant les instances de chaque tâche à échouer, la résilience de l'ensemble de tâches est inférée pour chaque scénario. D'après ces résultats, l'intervalle de confiance, et le degré de confiance associé, peuvent être calculés.

6.4 Conclusion

Ce chapitre a décrit l'état de l'art autour de la question de la robustesse des ordonnancements. Il en ressort que les travaux préexistants sont peu nombreux, et s'articulent autour de l'approche probabiliste et de la notion de garantie ou autour de l'approche statistique et la notion de résilience aux fautes.

Deux considérations différentes sont sous-jacentes à ces approches. Dans le cadre de l'approche statistique, l'objectif est d'obtenir une métrique indépendante d'un modèle de fautes, ou même d'une stratégie de recouvrement d'erreurs. Un objectif d'évaluation de robustesse « adimensionné » permettant de comparer différents systèmes est recherché.

L'approche probabiliste est quant à elle dépendante d'hypothèses sur la stratégie de recouvrement d'erreurs et du modèle de fautes. Ainsi, l'objectif recherché est la caractérisation des limites du système. Ces deux approches n'obéissent donc pas aux mêmes objectifs, qui ne sont pour autant pas antinomiques.

La définition d'une métrique pour évaluer la robustesse des ordonnancements est donc fortement corrélée à la recherche de propriété sur un ordonnancement temps réel. Même si la réflexion devrait être poussée plus loin, la recherche d'une métrique unique permettant de remplir les objectifs des approches probabilistes et statistiques ne semble pas pertinente. Par extension, la robustesse d'un ordonnancement regroupe alors plusieurs métriques permettant de caractériser l'ordonnancement de différentes manières.

Le parti pris de ces travaux de thèse est qu'un système, pris au sens général, est développé pour un ensemble de missions donné, dans un environnement physique défini. Cet

environnement, et notamment ses agressions envers le système, sont alors caractérisables. De fait, le ou les métriques utlisables pour évaluer la robustesse d'un ordonnancement dépendent des paramètres physiques de l'environnement physique dans lequel le système est immergé.

Dans le chapitre suivant, l'environnement est modélisable par des rafales de fautes. La robustesse des ordonnancements est donc évaluée suivant la durée Δ_F de la perturbation, qui en constitue donc la métrique.

Chapitre 7

Evaluation de la robutesse des ordonnancements

Sommaire

Le parti pris de ces travaux de thèse, au travers des sections 7.1 et 7.2, est de considérer que, pour un système temps réel donné, seul l'agressivité de l'environnement est dimensionnant dans la robustesse. La conception d'un tel système répondant à des spécifications techniques liées à un ou plusieurs environnements de fonctionnement, la forme générale des agressions de cet environnement est déterminante pour évaluer la robustesse. Les résultats proposés cherchent donc à donner à l'utilisateur ou au concepteur des certitudes sur les limites du système. Enfin, la section 7.3 conclut ce chapitre

7.1 Evaluation pour des rafales de fautes

Dans ces travaux de thèse, la résilience aux rafales de fautes se définit à deux niveaux. En premier lieu, une évaluation pour chaque tâche de la valeur maximale Δ_F telle que celle-ci soit toujours ordonnançable. En second lieu, une évaluation au niveau de l'ensemble de tâches, afin de rechercher la valeur maximale Δ_F telle que la faisabilité soit conservée.

7.1.1 Rafale de fautes maximale pour une tâche

Pour une tâche τ_i donnée, il existe une borne supérieure de la durée de la rafale de fautes telle que la tâche τ_i et les tâches de plus hautes priorités soient toujours ordonnançables. Cette borne supérieure, notée δ_i, peut être évaluée par l'intermédiaire de l'équation (4.2). Pour des raisons évidentes de cohérence, la borne δ_i ne peut en aucun cas être supérieure au minimum des bornes supérieures calculées pour les tâches de plus haute priorité. Dans le cas contraire, une des tâches de plus haute priorité deviendrait non ordonnançable. La borne supérieure respectant cette contrainte est notée $sup(\Delta_F^i)$, et son expression est donnée par l'équation (7.1).

$$sup(\Delta_F^i) = \min\left(\delta_i, \min_{hp(i)}\left(sup(\Delta_F^j)\right)\right) \tag{7.1}$$

Pour autant, l'existence de la borne $sup(\Delta_F^i)$ ne signifie pas nécessairement que le temps de réponse $\mathcal{R}_i^{\Delta_F}$ correspondant est atteignable. Pour cette raison, la borne supérieure de la rafale de fautes telle que le temps de réponse $\mathcal{R}_i^{\Delta_F}$ soit atteignable est appelée rafale de fautes maximale, notée $\Delta_F^{i,max}$.

Puisque le temps de réponse $\mathcal{R}_i^{\Delta_F}$ est une fonction linéaire de Δ_F, l'évaluation de la borne supérieure $sup(\Delta_F^i)$ permet de déduire la valeur de la rafales de fautes maximale $\Delta_F^{i,max}$. Ce résultat remarquable est déduit du théorème suivant :

Théorème 2 *Pour toute tâche τ_i, la rafale de fautes maximale $\Delta_F^{i,max}$ est égale à la borne supérieure $sup(\Delta_F^i)$.*

Preuve 1 *L'évaluation de $\mathcal{R}_i^{\Delta_F}$ s'appuie sur une suite de la forme $u_{n+1} = f(u_n)$. Le théorème est prouvé si la fonction f est une fonction linéaire.*

Le premier terme de la suite est $\mathcal{R}_{i,0}^{\Delta_F} = \mathcal{R}_i + F_i + \Delta_F$, noté par la suite $\alpha(\Delta_F)$. La valeur suivante de la suite est $\mathcal{R}_{i,1}^{\Delta_F} = \alpha(\Delta_F) + I_{i,1}^{\Delta_F}$. Le terme d'interférences $I_{i,1}^{\Delta_F}$ contient l'expression $\mathcal{R}_{i,0}^{\Delta_F} - (\mathcal{R}_i + \Delta_F)$ qui est égale à F_i. Ainsi, le terme $I_{i,1}^{\Delta_F}$ ne dépend pas de Δ_F. Par récurrence, la démonstration apportée montre que la valeur de la suite $I_{i,n}^{\Delta_F}$ ne dépend pas de Δ_F. Elle est fonction de la valeur de F_i et d'une combinaison linéaire de C_j.

De plus, chaque élément $\mathcal{R}_{i,n}^{\Delta_F}$ de la suite $\mathcal{R}_i^{\Delta_F}$ se construit par addition d'un élément $I_{i,n}^{\Delta_F}$ et du terme $\alpha(\Delta_F)$. En conséquence, l'hypothèse $\Delta_F^1 \neq \Delta_F^2$ implique la contradiction suivante :

$$\mathcal{R}_{i,n}^{\Delta_F^1} - \mathcal{R}_{i,n}^{\Delta_F^2} = \alpha(\Delta_F^1) - \alpha(\Delta_F^2) = \Delta_F^1 - \Delta_F^2$$

Ce résultat prouve que toute variation de $\mathcal{R}_i^{\Delta_F}$ est égale Δ_F. L'augmentation de $\mathcal{R}_i^{\Delta_F}$ en fonction de Δ_F est donc linéaire, ce qui démontre que $sup(\Delta_F^i)$ est une valeur atteignable.

La linéarité de l'équation conduit à un résultat corollaire qui facilite l'évaluation de $\mathcal{R}_i^{\Delta_F}$, dont l'expression est présentée dans l'équation (7.2).

$$\forall x > 0,\ \mathcal{R}_i^{\Delta_F = x} = \mathcal{R}_i^{\Delta_F = 0} + x \tag{7.2}$$

La connaissance d'un couple $(\mathcal{R}^{\Delta_F},\ \Delta_F)$ est alors suffisante pour déduire l'évolution du pire temps de réponse d'une tâche τ_i en fonction de la variation de la durée de la rafale de fautes.

7.1.2 Evaluer la borne supérieure δ_i

Ces travaux de thèse considèrent des systèmes temps réel durs et des échéances de tâches inférieures ou égales à la période. Ce cadre de travail impose donc l'échéance comme une borne supérieure du pire temps de réponse en l'absence de fautes. Sous l'hypothèse de rafales de fautes, cette contrainte reste de mise, et s'exprime sous la forme $\mathcal{R}_i^{\Delta_F} \leqslant D_i$.

L'évaluation de la borne supérieure δ_i suit la méthodologie suivante. Initialement, toutes les occurences de $\mathcal{R}_i^{\Delta_F}$ sont remplacées par l'échéance D_i dans l'équation (4.2). Les termes $\mathcal{R}_i$ et F_i ne subissent aucune modification. De plus, le terme F_i est indépendant de la durée de la rafale de fautes Δ_F. En conséquence, la séquence de recouvrement d'erreurs restant inchangée, l'évaluation de la borne supérieure δ_i est uniquement fonction de Δ_F. L'équation (7.3) exprime cette première étape.

$$D_i = \mathcal{R}_i + \Delta_F + \left(I_i^{\Delta_F}(D_i) + F_i\right) \tag{7.3}$$

Le terme d'interférences évalué dans l'équation (7.3) s'obtient en remplaçant le terme $\mathcal{R}_i^{\Delta_F}$ par D_i. La nouvelle expression de $I_i^{\Delta_F}$ est donnée par l'équation (7.4).

$$I_i^{\Delta_F}(D_i) = \sum_{hp(i)} \left\lceil \frac{D_i - (\mathcal{R}_i + \Delta_F)}{T_j} \right\rceil C_j \tag{7.4}$$

Les termes D_i, $\mathcal{R}_i$ et F_i sont constants et calculés indépendamment. Cette constatation conduit à poser $\alpha = D_i - R_i - F_i$ et $\beta = D_i - R_i$ pour simplifier l'équation (7.3). La n-ième valeur de δ est notée $\delta_{i,n}$. L'équation (7.5) est une réécriture de l'équation (7.3) intégrant ces différentes notations.

$$\delta_{i,n+1} = \alpha - \sum_{hp(i)} \left\lceil \frac{\beta - \delta_{i,n}}{T_j} \right\rceil \times C_j \tag{7.5}$$

Dans cette équation, l'inconnue δ_i est présente de chaque côté de l'égalité. L'application du théorème du point fixe permet donc de résoudre l'équation (7.5). Puisque chaque terme est positif, l'inégalité $0 < \delta_i \leqslant \alpha$ est vérifiée. La valeur initiale $\delta_{i,0}$ est égale à α_1. Le terme $I_i^{\Delta_F}$ décroît lorsque le terme $\delta_{i,n}$ croît. Cette variation assure la convergence du point fixe vers une valeur inférieure ou égale à 0.

Les conditions d'arrêt de l'algorithme sont soit $\delta_i < 0$ soit $\delta_{i,n+1} = \delta_{i,n}$. Dans le premier cas, la tâche considérée est non ordonnançable quelque soit la durée de la rafale de fautes. Dans le second cas, l'algorithme fournit une borne supérieure δ_i égale à $\delta_{i,n}$.

Si par ailleurs $\delta_i = 0$, alors la tâche τ_i est ordonnançable sous l'hypothèse d'une faute instantanée qui impacte celle-ci et toutes les tâches de plus haute priorité. Ce résultat est intéressant car il permet de traiter le cas limite $\Delta_F = 0$, qui fait la frontière avec le modèle de distributions de fautes pseudo-périodiques.

7.1.3 Résilience aux rafales de fautes d'un système temps réel

La résilience aux rafales de fautes d'un système temps réel, notée $\Delta_F^{resilient}$, est définie comme la durée maximale d'une rafale de fautes telle que l'ensemble de tâches Γ soit toujours faisable. Cette définition impose donc que chaque tâche soit toujours ordonnançable Ce résultat est obtenu si la résilience aux rafales de fautes du système est le minimum de l'ensemble des résiliences des tâches (équation (7.6)).

$$\Delta_F^{resilient} = \min_{i \in n}(\Delta_F^{i,max}) \tag{7.6}$$

La connaissance de $\Delta_F^{resilient}$ implique que pour toute durée de rafale inférieure ou égale à cette résilience, l'ensemble de tâches Γ est faisable. La propriété 3 en est déduite.

Propriété 3 $\forall \Delta_F,\ \Delta_F \leqslant \Delta_F^{resilient}$*, l'ensemble de tâches est faisable*

La propriété 3 est une condition suffisante pour valider un ensemble de tâches sous l'hypothèse d'une rafale de fautes.

7.1.4 Exemple

La Table 7.1 regroupe les résultats du calcul de la résilience aux rafales de fautes, au niveau tâche et ensemble de tâches. L'ensemble de tâches est identique à celui des tables 4.1, 4.2 et 4.3. Les stratégies ED/FR/S et ED/FR/M sont considérées. Dans le cadre de cette dernière, les deux méthodes d'évaluation du terme F_i sont appliquées.

	δ_i			$\Delta_F^{i,max}$			
	τ_1	τ_2	τ_3	τ_1	τ_2	τ_3	$\Delta_F^{resilient}$
ED/FR/S	270	310	100	270	270	100	100
ED/FR/M	270	360	260	270	270	260	260
ED/FR/Mr	270	360	270	270	270	270	270

TABLE 7.1 – Exemple de résilience aux fautes sous rafales de fautes

Ces résultats illustrent la possible différence entre la valeur de la borne supérieure δ_i et la durée maximale de la rafale de fautes $\Delta_F^{i,max}$. Par exemple, pour la stratégie ED/FR/S,

la borne δ_2 est supérieure à la borne δ_1 ($310 > 270$). Cependant, pour que la tâche τ_1 reste ordonnançable, il est nécessaire d'avoir $\Delta_F^{2,max} = \Delta_F^{1,max} = 270$.

L'application de la stratégie ED/FR/M (calcul de F_i raffiné) montre qu'il est contre-intuitif de penser que la tâche de moins haute priorité est nécessairement celle dont la résilience dimensionne le système. En effet, dans cet exemple, la résilience $\Delta_F^{resilient}$ est telle que $\Delta_F^{3,max} = \Delta_F^{1,max} = \Delta_F^{resilient}$.

En complément, il est intéressant de noter que la stratégie ED/FR/M, quelque soit son raffinement, offre une meilleure résilience aux rafales de fautes que la stratégie ED/FR/S. Dans le cadre de cet exemple, cette résilience est presque trois fois supérieure (260/270 contre 100 unités de temps). L'efficacité de la stratégie ED/FR/M est de nouveau mise en évidence.

7.2 Evaluation pour des fautes pseudo-périodiques

Cette section est consacrée à l'évaluation de la robustesse des ordonnnancements, sous l'hypothèse de fautes pseudo-périodiques. De prime abord, cette digression vers un autre modèle de distribution de fautes peut paraître incongrue. Cependant, celle-ci trouve sa justification dans la sous-section 7.1.2.

En effet, si l'évaluation de la borne supérieure conduit à un résultat négatif ($\delta_i < 0$), cela signifie que même sous l'hypothèse d'une faute ponctuelle impactant τ_i et toutes les tâches de plus haute priorité, la tâche n'est pas ordonnançable, et donc que l'ensemble de tâches n'est pas faisable. Hors, comme $\Delta_F = 0$ représente le cas limite entre le modèle en rafales de fautes et le modèle de fautes pseudo-périodiques, l'évaluation de la robustesse peut donc être poursuivie sur ce second modèle.

Cette évaluation doit se faire avec des hypothèses pour partie différente. Le cas limite $\Delta_F = 0$ fourni comme information principale que l'ensemble des tâches n'est pas robuste à une faute ponctuelle impactant toutes les tâches. L'hypothèse peut alors être faite que toute faute impacte une unique tâche à la fois, hypothèse à la base des travaux de [Pun97].

L'analyse de ce cas a conduit à la définition de la méthode décrite ci-après. Cette méthode s'inscrit donc le même cadre d'hypothèses que celles décrites dans la section 6.2. Elle s'en distingue en proposant une méthode analytique directement basée sur l'exploitation et l'inversion de l'équation 2.3.

Dans ces travaux de thèse, la résilience aux fautes pseudo-périodiques se définit comme l'intervalle T_F minimal séparant l'occurence de deux fautes. La minimisation de la pseudo-période T_F maximise la valeur du temps de réponse $\mathcal{R}_i^{T_F}$ atteignable. Ce pire temps de réponse atteignable est noté $\mathcal{R}_{i,max}^{T_F}$.

7.2.1 Minimiser l'intervalle entre deux fautes

Sur les bases de l'équation 2.3, le pire temps de réponse $\mathcal{R}_{i,max}^{T_F}$ s'exprime comme une combinaison linéaire de C_i et $\overline{C_i}$. L'ensemble de tâches est supposé indépendant ($\forall i, B_i = 0$). Les scalaires de cette combinaison sont notés λ_j (nombre de préemptions de la tâche τ_j sur τ_i), et σ, nombre de fautes intervenant pendant le calcul du pire temps de réponse. Les expressions de ces scalaires sont $\left\lceil \frac{\mathcal{R}_{i,max}^{T_F}}{T_j} \right\rceil$ et $\left\lceil \frac{\mathcal{R}_{i,max}^{T_F}}{T_F} \right\rceil$ respectivement. La combinaison linéraire correspondant à $\mathcal{R}_{i,max}^{T_F}$ est exprimée dans l'équation (7.7).

$$\mathcal{R}_{i,max}^{T_F} = C_i + \sum_{hp(i)} \lambda_j C_j + \sigma \times \max_{j \in i \cup hp(i)} \overline{C_j} \tag{7.7}$$

Le terme σ est une fonction de T_F. Puisque l'opérateur *ceiling* retourne le plus petit entier égal ou supérieur à la valeur de son argument, la valeur de T_F telle que $\mathcal{R}_i^{T_F}$ soit maximal est potentiellement non unique. En conséquence, T_F appartient à un intervalle éventuellement réduit à un seul point. Cette intervalle est noté $\mathcal{I}_i^{max}$. Ces travaux proposent d'évaluer la borne inférieure et la borne supérieure de cet intervalle semi-ouvert. L'équation (7.8) fournit l'expression de l'intervalle $\mathcal{I}_i^{max}$.

$$T_F \in \mathcal{I}_i^{max} = \left[\left\lfloor \frac{\mathcal{R}_{i,max}^{T_F}}{\sigma} \right\rfloor ; \left\lfloor \frac{\mathcal{R}_{i,max}^{T_F}}{\sigma - 1} \right\rfloor \right[\tag{7.8}$$

Dans l'équation (7.8), l'opérateur *floor* ($\lfloor\ \rfloor$)retourne la partie entière de son argument. Le paramètre σ représente le nombre maximal de fautes tel que la tâche considérée soit toujours ordonnançable.

7.2.2 Evaluation de l'intervalle $\mathcal{I}^{max}$

Cette sous-section propose une méthode pour évaluer le temps de réponse $\mathcal{R}_{i,max}^{T_F}$. La n-ième valeur de $\mathcal{R}_i^{T_F}$ est notée x_n. Par ce biais, l'équation (7.7) est modifiée pour ne plus dépendre que de T_F. L'équation (7.9) en donne la nouvelle expression.

$$x_n = C_i + \sum_{hp(i)} \lambda_{j,n} C_j + \left\lceil \frac{x_n}{T_F} \right\rceil \max_{j \in i \cup hp(i)} \overline{C_j}, \ avec\ \lambda_{j,n} = \left\lceil \frac{x_n}{T_j} \right\rceil \tag{7.9}$$

Dans l'équation (7.9), pour toute valeur de x_n, les termes C_i et $\sum_{hp(i)} \lambda_{j,n} C_j$ sont constants. Ainsi, en posant $\alpha(x_n) = x_n - C_i - \sum_{hp(i)} \left\lceil \frac{x_n}{T_j} \right\rceil C_j$, l'équation (7.9) se simplifie en l'équation (7.10).

$$\alpha(x_n) = \left\lceil \frac{x_n}{T_F} \right\rceil \max_{j \in i \cup hp(i)} \overline{C_j} \tag{7.10}$$

Puisque la valeur de $\left\lceil \frac{x_n}{T_F} \right\rceil$ est un entier, la fonction $\alpha(x_n)$ est discontinue. L'équation (7.10) n'est par conséquent vérifiée que si $\alpha(x_n)$ est un multiple entier de $\max_{j \in i \cup hp(i)} C_j$. La méthode d'évaluation de $\mathcal{I}_i^{max}$ et $\mathcal{R}_{i,max}^{T_F}$ est basée sur la vérification de l'équation (7.10) et s'articule autour de quatre étapes décrites ci-après :

Étape 0 : *Soit* $x_0 = D_i$ Les contraintes de temps réel dur pour les tâches imposent que la borne supérieure du temps de réponse soit égale à l'échéance D_i. La présence de fautes pseudo-périodiques ne remet pas en cause cette contrainte, qui s'exprime $\mathcal{R}_i^{T_F} \leqslant D_i$.

La méthode est donc initialisée avec $x_0 = D_i$, ce qui maximise la valeur de la combinaison linéaire $\sum \lambda_{j,0} C_j$. Puisque la valeur de x_n est positive, et que $x_0 > x_1 ... > x_n$, la convergence et la terminaison de la méthode est assurée.

Étape 1 : *Equation* (7.10) L'équation (7.10) est vraie si et seulement si $\alpha(x_n)$ est un multiple de $\max_{j \in i \cup hp(i)} C_j$. Si l'égalité est vérifiée, alors les valeurs de $\mathcal{R}_{i,max}^{T_F}$ et de σ sont égales respectivement à x_n et $\left\lceil \frac{x_n}{T_j} \right\rceil$. L'équation (7.8) permet de déduire la valeur de $\mathcal{I}_i^{max}$.

Si l'égalité n'est pas vérifiée, alors la nouvelle valeur de σ_n est exprimée par $\left\lfloor \frac{\alpha(x_n)}{\max C_j} \right\rfloor$. Si $\sigma_n > 0$, alors la prochaine étape de la méthode est l'étape 2. Sinon, l'étape 3 doit être appliquée.

Étape 2 : $\sigma_n > 0$ Dans ce cas de figure, l'expression $\sum \lambda_{j,n} C_j + \sigma_n \max(C_j)$ est évaluée et fournit la valeur de x_{n+1}. La méthode proposée recherchant une valeur de σ telle que l'équation (7.10) soit vérifiée, l'inégalité $\sigma_n > 0$ siginfie qu'une faute de trop est prise en compte. Cette faute supplémentaire provoque la non ordonnançabilité de la tâche. Ainsi, la valeur σ_n décroit d'une unité le nombre de fautes prise en compte dans l'évaluation de x_{n+1}, sans remettre en cause la combinaison linéaire maximale considérée.

La méthode conclut en posant $\mathcal{R}_i^{T_F} = x_{n+1}$ et $\sigma = \sigma_n$. L'équation (7.8) permet alors de déduire la valeur de l'intervalle $\mathcal{I}_i^{max}$.

Étape 3 : $\sigma_n = 0$ Ce cas de figure signifie que pour la combinaison linéaire $\sum \lambda_{j,n} C_j$ considérée, aucune faute n'est tolérée. Pour poursuivre la méthode, il est nécessaire de diminuer la valeur de la combinaison linéaire. Cette diminution est obtenue en retranchant une préemption dûe à l'une des tâches de plus haute priorité.

La tâche retenue est celle qui préempte le plus la tâche τ_i, donc celle tel que $\lambda_j = max_{hp(i)}(\lambda_{j,n})$. Pour assurer l'unicité de λ_j, il est nécessaire que la valeur de $\frac{x_n}{T_j}$ soit minimale. La valeur de de λ_j est décrémentée d'une unité. La déduction de la valeur x_{n+1} correspondante permet de retourner à **l'étape 1**.

7.2.3 Exemple

L'exemple suivant représente l'évaluation de l'intervalle $\mathcal{I}_i^{max}$ pour un ensemble de trois tâches ordonnancées sous Rate Monotonic. Les tâches sont à échéance sur requête et l'ensemble est faisable en l'absence de fautes.

P	T	C	D	$\mathcal{R}_{i,max}^{T_F}$	σ	$\mathcal{I}^{max}$
1	300	10	300	300	29	10
2	500	50	500	470	8	$[58; 67[$
3	800	150	800	730	3	$[243; 365[$

TABLE 7.2 – Exemple de calcul de l'intervalle $\mathcal{I}^{max}$

En premier lieu, force est de constater que les échéances D_2 et D_3 ne sont pas des valeurs maximales atteignables en présence de fautes pseudo-périodiques. La discontinuité mise en évidence par l'équation (7.10) rend donc ces valeurs inatteignables.

La résilience aux fautes pseudo-périodiques varie selon les tâches. Ainsi, la tâche τ_1 voit son intervalle $\mathcal{I}_1^{max}$ réduit à un unique point. Les tâches τ_2 et τ_3 ont des intervalles $\mathcal{I}_2^{max}$ et $\mathcal{I}_3^{max}$ avec des bornes inférieures et supérieures différentes. Les intervalles $\mathcal{I}_i^{max}$ issus des résultats sont disjoints.

7.2.4 Résilience de l'ensemble de tâches

Une exploitation directe de ces résultats permet de conclure sur la résilience aux fautes pseudo-périodiques de l'ensemble de tâches. Cette résilience est évaluée comme l'intervalle $\mathcal{I}^{max}$ tel que l'ensemble de tâches reste faisable. La borne inférieure de cet intervalle est la borne inférieure la plus élevée parmi les $\mathcal{I}_i^{max}$. Le raisonnement est identique pour la borne supérieure.

Dans l'exemple décrit, le caractère disjoint des $\mathcal{I}_i^{max}$ conduit à considérer l'intervalle $\mathcal{I}_3^{max}$ comme la résilience de l'ensemble de tâches. Le comportement de la tâche τ_3 en présence de fautes pseudo-périodiques est donc le paramètre dimensionnant du système en regard de cette perturbation.

7.3 Conclusion

Ce chapitre a présenté deux résultats concernant la robustesse des ordonnancements tolérants aux fautes. Le premier donne une méthode analytique pour évaluer la durée maximale Δ_F d'une rafale de fautes telle qu'un ensemble de tâches soit faisable. Cette méthode repose sur la démonstration que le pire temps de réponse sous rafales de fautes varie linéairement avec la durée de la rafale, puis sur un algorithme telle que les conditions d'arrêt sont soit une durée négative, soit une convergence de la valeur Δ_F.

Le second résultat concerne les tâches affectées par des fautes pseudo-périodiques. En effet, le cas $\Delta_F = 0$ est le cas limite entre le modèle en rafales de fautes et celui pseudo-périodique. Si la première méthode analytique conclut que même une faute de durée nulle rend l'ensemble de tâches non faisable, la seconde méthode proposée permet de poursuivre l'analyse sous l'hypothèse d'une distribution pseudo-périodique où chaque faute impacte au plus une tâche.

Ce chapitre clôt la partie consacrée à la robustesse des ordonnancements temps réel. La partie suivante s'intéresse à l'intégration des ordonnancements tolérants aux fautes dans les analyses de niveau système.

Quatrième partie

Vers une analyse de niveau système

Chapitre 8

Modéliser un système soumis à des rafales de fautes

Sommaire

Les résultats présentés aux chapitre 4 et montrent comment analyser l'ordonnançabilité d'un ensemble de tâches soumis à une rafale de fautes. Implicitement, la rafale de fautes, telle que définie au chapitre 3, rend compte de la perturbation ressentie par l'équipement hébergeant l'ensemble de tâches. Elargir l'analyse au niveau système, en tenant compte de l'impact de la perturbation sur l'architecture matérielle et fonctionnelle, tout en mettant en évidence le rôle de l'ordonnanceur, justifie ce chapitre d'ouverture vers d'autres travaux.

L'objectif de ce chapitre d'ouverture est de proposer une ébauche d'étude de sûreté de fonctionnement tenant compte du rôle de l'ordonannceur. Cette étude s'appuie sur une modélisation intégrant les éléments d'architecture matérielle et logicielle, et les échanges de données. Le modèle d'un système temps réel est ensuite stimulé par une perturbation extérieure, et le comportement du système en présence de la perturbation mis en évidence. C'est sur ces bases que l'analyse de l'impact de l'ordonnanceur dans la sûreté de fonctionnement devient accessible.

Le chapitre se structure comme suit. La section 8.1 présente la problématique générale du chapitre, tandis que la section 8.2 en donne un exemple. La section 8.3 présente les différents éléments du modèle, les composants et les données. La section 8.4 introduit la formalisation des perturbations subies par un système temps réel. La section 8.5 présente les différentes fonctions nécessaires pour faire évoluer le comportement du système modélisé durant une rafale de fautes. Sur ces bases, la section 8.6 conclut sur le rôle de l'ordonnanceur dans l'exemple introductif. La section 8.7 présente les problématiques de chaînes descendantes et de boucles pour lesquelles les travaux présentés ici constituent des premiers pas pour les analyser. Enfin la section 8.8 conclut ce chapitre.

8.1 Introduction

L'exemple illustratif de la section 3.1.1 présentait un aéronef dans un champ életromagnétique. La rafale de fautes alors définie correspondait à la perturbation subie, *id est* au temps d'exposition par balayage du radar. Cependant, ce temps d'exposition à une perturbation électromagnétique peut différer du temps de la perturbation effectivement ressentie par le système.

En effet, les éléments de l'architecture matérielle du système n'ont pas tous la même réaction lorsqu'ils sont exposés à une perturbation quelconque. Cette réaction, qui a potentiellement un effet retard sur l'effet de la perturbation (*résilience*) et un effet post-perturbation (*recouvrement*), transforme une **perturbation subie** en une **perturbation résultante**. Ces effets sont détaillés dans la section 8.3.

La problématique de la perturbation résultante se complique dès lors qu'en plus des réactions individualisées des différents équipements, la communication entre les équipements est prise en compte. Ainsi, pour un équipement perturbé, utilisant des données issues d'un autre équipement, la durée de la perturbation résultante peut être plus importante que la durée de la perturbation subie.

L'évaluation de la perturbation résultante permet d'approfondir l'étude du comportement de l'ordonnancement tolérant aux fautes. L'aboutissement d'une telle étude est ainsi une mesure de l'impact de l'ordonnancement au sein des analyses de sûreté de fonctionnement. En effet, l'ordonnanceur doit alors gérer les effets de la perturbation sur l'équipement qui l'héberge, et les effets de la perturbation sur les données qui sont utilisées

par les différentes tâches.

8.2 Exemple introductif

Ce chapitre propose de suivre la méthodologie de modélisation au travers d'un exemple simple. Le système est considéré est un capteur relié à un calculateur par un réseau (figure 8.1). Le capteur est un capteur numérique émettant une donnée d toutes les 60 ms. Chacune des émissions se produit au mieux instantanément, et au pire au bout de 5ms. La donnée d émise est ensuite transmise sur le réseau, qui assure un délai de transmission borné, entre 5 ms et 10 ms. Enfin, la donnée d est utilisée par le calculateur par l'intermédiaire d'une tâche τ_1. De fait, la tâche τ_1, telle que $D_1 = T_1 = 300$ ms et $C_1 = 50$ ms, est dépendante de la donnée d.

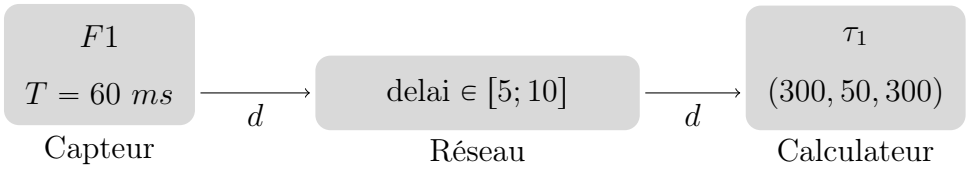


FIGURE 8.1 – Exemple

Soumis à une rafale de fautes, chaque composant du système temps réel se comporte comme suit. A partir du début de la rafale, le capteur dysfonctionne au bout de 10 ms, et retrouve un fonctionnement nominal 5 ms après la fin de la rafale. De la même façon, le réseau dysfonctionne dès le début de la perturbation, et retrouve un fonctionnement nominal à la fin de la rafale. Enfin, le calculateur est considéré comme insensible à la perturbation. La stratégie de l'ordonnanceur est ED/FR/S. Avant chaque réexécution, la tâche τ_1 met à jour la valeur de la donnée d. Enfin, la rafale de fautes possède une durée de 150 ms, de pseudo-période 700 ms.

8.3 Modélisation du système

Le choix de modélisation qui va être décrit ci-après répond à plusieurs exigences. En premier lieu, son caractère formel doit aboutir à une description sans ambiguïté du système et de son comportement, permettant des analyses réalistes et pertinentes. En second lieu, en lien avec la problématique de ce chapitre, il doit mettre en évidence les dysfonctionnements de chacun des composants du sytème durant la rafale de fautes, et leurs effets sur les différents flots de données. Enfin, cette modélisation, qui inclut le niveau ordonnancement et le lien entre les données et les tâches, conduit à la définition d'un modèle multi-niveau, à même d'étudier l'effet de l'ordonnancement tolérant aux fautes sur la sûreté de fonctionnement du système.

Cette section se structure comme suit. La sous-section 8.3.1 présente les états des éléments modélisés. La sous-section 8.3.2 décrit les données considérées dans cette étude. La sous-section 8.3.3 définit la notion de composant matériel, pierre angulaire du modèle du système temps réel. Enfin, la sous-section 8.3.4 montre les différents types de composants qui seront manipulés dans la modélisation.

8.3.1 Etats des différents éléments modélisés

Dans ce chapitre d'ouverture, les éléments matériels, ainsi que les données au sein des flots de données possèdent un statut interne. Ce statut interne permet d'observer l'état de l'élément modélisé durant son fonctionnement, et notamment pendant les perturbations externes qui conduisent à des dysfonctionnements.

Ce statut interne appartient à l'ensemble $\mathcal{S} = \{ok, err\}$. Un état correspondant à une valeur *ok* représente un fonctionnement nominal d'un élément matériel, ou à une donnée valide. *A contrario*, un statut *err* représente un dysfonctionnement matériel, ou une donnée erronée.

L'ensemble $\mathcal{S}$ pourrait représenter plus d'états, comme, pour un élément matériel le fonctionnement intempestif ou la perte totale. De même, une donnée pourrait être perdue, au sens inexploitable par le système ou absente. Ces cas sont visiblement à considérer, mais ne sont pas couverts par ce chapitre d'ouverture.

En conséquence, ce chapitre utilise donc une logique à deux états qui est analogue à la logique booléenne. En considérant que l'état *ok* est analogue à la valeur booléenne *vrai*, et l'état *err* à la valeur booléenne $faux$, les opérateurs *et* ($\wedge$) et *ou* ($\vee$) sont applicables. Les tables de vérité qui sont associées à ces opérateurs sont donc utilisées dans le cadre de l'ensemble $\mathcal{S}$.

8.3.2 Données

Une donnée représente l'élément échangé entre différents composants matériel (cf. section suivante). Dans ces travaux, les caractéristiques des données concernent leur validité et leur localisation temporelle au sein du système temps réel. La modélisation s'intéresse au caractère valide ou non de la donnée, *id est* si la valeur de la donnée est correcte ou erronée. Pour autant, le niveau de détail choisit dans le cadre de ces travaux ne s'intéresse pas à sa valeur exacte, qui peut être numérique ou littérale.

L'autre caractéristique concerne la localisation temporelle de la donnée. En effet, pour des raisons liées aux contraintes de déterminisme, la présence d'une donnée au sein d'un système temps réel doit être associée avec des instants temporels t. Cependant, les divers équipements du système temps réel ajoutent des gigues et des latences, conduisant à transformer ces intants t en des intervalles. La conservation de la propriété de déterminisme implique donc de pouvoir évaluer ces intervalles. Le choix de modélisation d'une

donnée intègre donc cet intervalle temporel.

Formalisation d'une donnée Ce paragraphe propose une définition d'une donnée. Cette formalisation reprend les principales caractéristiques décrites précédemment.

Définition 5 *Donnée*

Une donnée est un triplet $D = (s, \delta^-, \delta^+)$ *où :*
– s *est le statut interne,* $s \in \mathcal{S}$
– $\delta^- \in \mathbb{N}$, *borne inférieure d'un intervalle*
– $\delta^+ \in \mathbb{N}$, *borne supérieure d'un intervalle*

Le statut interne s répond donc à l'intérêt porté par cette étude sur la validité des données échangées. De même, la définition intègre le déterminisme temporel des données. Dans la mesure où les instants d'émission, de transmission d'une donnée dans le système réel sont variables mais bornés, la formalisation de la donnée intègre cette incertitude par la définition d'un intervalle borné par δ^- et δ^+.

8.3.3 Composant matériel

Un **composant matériel** est l'élément atomique de la modélisation du sytème temps réel. Un composant représente une ressource physique de l'architecture matérielle. Toute ressource physique, soumise à une agression extérieure, voit son comportement perturbé, ce qui conduit à des dysfonctionnements. La description d'un composant intègre cette notion de dysfonctionnement au travers d'un statut interne appartenant à l'ensemble $\mathcal{S}$.

8.3.3.1 Description fontionnelle et dysfonctionnelle

Dans le cadre de ce chapitre, la notion de dysfonctionnement correspond à un comportement erroné de la ressource physique, comportement qui remet en cause la validité des données émises, calculées ou transmises. Ce dysfonctionnement dépend de la perturbation considérée, une perturbation électromagnétique n'ayant pas le même effet, par exemple, qu'une perturbation due à des vibrations. De plus, les différentes ressources physiques, suivant leurs spécifications techniques, leur placement au sein du système temps réel (par exemple dans une baie avionique d'un aéronef, ou à l'extérieur sur une aile), n'ont pas la même réaction en présence d'une perturbation d'un type donné.

Résilience et recouvrement Pris individuellement, chaque équipement, exposé à une perturbation, va se comporter différemment. La visibilité des effets, par exemple l'émission de données erronées, peut ne pas être immédiate. Ce délai, qualifié précédemment d'effet retard, est appelé **résilience** de l'équipement. Plus cette résilience est élevée, plus la durée

de la perturbation devra être longue pour affecter l'équipement. Typiquement, l'usage d'un blindage sur un équipement permet d'en améliorer sa résilience.

Par ailleurs, un équipement peut continuer à dysfonctionner après la fin de la perturbation. Par exemple, ce dysfonctionnement est observable au travers des données erronées. Le temps écoulé entre la première donnée valide émise après la perturbation et la dernière donnée erronée durant la perturbation correspond alors à un délai nécessaire pour que l'équipement fonctionne nominalement. Ce délai peut, par exemple, être égal à la période d'émission des données. Dans le cadre de ces travaux, ce délai est appelé **recouvrement** d'un équipement.

Ainsi, pour un équipement matériel, la résilience et le recouvrement influent sur l'évaluation de la perturbation résultante (figure 8.2). Chaque équipement matériel, et au travers du modèle chaque composant matériel, a une perturbation résultante qui lui est propre. La question de la perturbation résultante du point de vue du système pris dans sa globalité nécessite quant à elle de s'intéresser aux flots de données échangés.

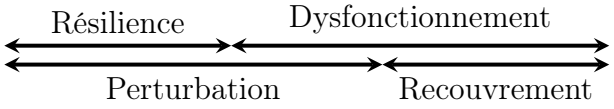


FIGURE 8.2 – Représentation temporelle de la résilience et du recouvrement

Flots de données Chaque composant contient un ensemble de ports d'entrée et de sortie, ainsi que des fonctions internes. Un système étant modélisé par un ensemble de composants, il est donc nécessaire de pouvoir les interfacer, afin de rendre compte de l'architecture fonctionnelle. De fait, chaque composant peut recevoir des flots données par ses ports d'entrée, effectuer des tâches de calcul dessus et les transmettre via les ports de sortie.

Les différents flots de données du système temps réel représentent des suites de données échangées entre des composants. Pour une donnée émise ou transmise, le statut interne de chacune de ses données dépend d'une fonction associant un ensemble de données d'entrée au statut interne du composant émetteur ou transmetteur.

8.3.3.2 Formalisation d'un composant matériel

A partir des différentes descriptions d'une ressource physique, la formalisation d'un composant matériel est possible. La définition d'un composant matériel est donnée ci-après :

Définition 6 *Composant matériel*

Un composant matériel est un n-uplet $C = (s, res, rec, I, O, F)$ *où :*

- s *est le statut interne,* $s \in \mathcal{S}$
- res *est la résilience,* $res \in \mathbb{N}$
- rec *est le recouvrement,* $rec \in \mathbb{N}$
- I*, ensemble des données d'entrée,*
- O*, ensemble des données de sortie,*
- F*, ensemble des fonctions liant des entrées et le statut interne aux sorties.*

L'ensemble F contient des fonctions f qui, à partir d'un sous-ensemble de I et du statut interne, associent une donnée de sortie de l'ensemble O. Ainsi pour un composant au statut interne s, si $I = \{i_1, i_2, i_3\}$, et $O = \{o_1, o_2, o_3\}$, l'ensemble F peut contenir trois fonctions telles que :

- $o_1 = f_1(s, i_1)$
- $o_2 = f_2(s, i_2, i_3)$
- $o_3 = f_3(s, i_1, i_3)$

Le caractère dynamique du modèle dépend du comportement temporel des différents composants. Chacun de ses composants est relié à un autre par des flots de données, connectés par des ports. Au sein des flots de données, chaque donnée peut être identifiée temporellement par son intervalle de temps correspondant à son temps d'existence au sein du système temporel.

Les différences fonctionnelles et matérielles entre les différentes ressources du système impliquent donc d'instancier la notion de composant. Cette instanciation se fait alors par la définition de types de composants, et est présentée dans la sous-section suivante.

8.3.4 Types de composants

La notion de composant pécédemment définie est une abstraction de haut niveau des caractéristiques de toute ressource. Cette abstraction est insuffisante pour introduire le comportement temporel de chaque ressource, et par extension la dynamique même du système. Ainsi, la nécessité de raffiner la notion de composant, suivant la nature de la ressource modélisée, impose de construire plusieurs types. Ce chapitre d'ouverture propose trois types pour décrire un système temps réel : les capteurs, les ressources de calculs et enfin les connecteurs. Les sous-sections suivantes présentent séquentiellement ces trois types, et montrent leur application dans le cadre de l'exemple introductif.

8.3.4.1 Capteurs

Un **capteur** est une ressource physique dont le rôle est de produire périodiquement des données. Après acquisition, ces données sont émises dans un intervalle de temps correspondant à une émission au plus tôt, et une émission au plus tard. Ces travaux de thèse se concentrant sur les émissions de données, chaque émission peut être considérée comme

une tâche périodique, ayant un pire temps d'exécution de durée égale à la longueur de l'intervalle d'émission.

Formalisation d'un capteur Un capteur est donc un composant qui possède une ou plusieurs données de sortie, chacune possédant une période d'émission. Cette période d'émission est bornée par un meilleur temps d'émission et un pire temps d'émission. Un modèle de capteur est défini formellement comme suit :

Définition 7 *Capteur*

Un capteur est un composant $C = (s, res, rec, \varnothing, O, F)$ *où chaque fonction* $f_i \in F$ *est définie de la façon suivante :*

$$\begin{array}{rccc} f_i : & \mathcal{S} & \longrightarrow & O \\ & C.s & \longmapsto & o_i = (C.s, \delta_i^- + kT_i, \delta_i^+ + kT_i) \end{array} \tag{8.1}$$

Dans la définition de la fonction f_i*, la valeur* δ_i^- *est le meilleur temps d'émission,* δ_i^+ *le pire temps d'émission et* T_i *la période d'émission associée à la donnée* o_i*. Le facteur k représente la k-ième instance de la donnée. Ainsi, le flot de données émis par un capteur peut s'exprimer comme la suite* $(o_i)_k$*.*

Retour sur l'exemple introductif Appliqué à l'exemple introductif, le capteur numérique est donc un composant Ca tel que $Ca = (s, 10, 5, \varnothing, \{d\}, \{f\})$. La fonction f représente l'émission de la donnée d :

$$\begin{array}{rccc} f : & \mathcal{S} & \longrightarrow & O \\ & Ca.s & \longmapsto & d = (Ca.s, 5 + 60k, 10 + 60k) \end{array}$$

La figure 8.3 montre un exemple de modélisation, sous forme d'échéancier.

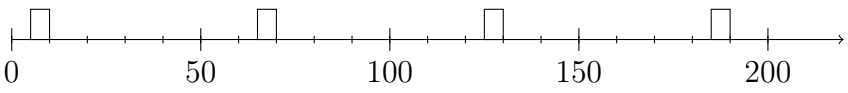


FIGURE 8.3 – Modélisation des émissions du capteur Ca

8.3.4.2 Ressources de calcul

Une **ressource de calcul** décrit une ressource dont le rôle est d'exploiter un ensemble de données issues de capteurs ou d'autres ressources de calcul dans le but d'effectuer des tâches précises (calculateur embarqué, concentrateur de données). Ce modèle de ressources de calculs intègre un ensemble de tâches Γ.

Description fonctionnelle Les tâches de l'ensemble Γ peuvent dépendre des données d'entrée. Cette dépendance est considérée comme une conjonction des statuts internes des données, *i.e* si une tâche dépend de trois données, il suffit qu'au moins une des données aie un statut interne valant *err* pour que le résultat de la fonction soit *err*. Le modèle de tâches associé étend celui présenté dans la sous-section 1.2.2 en prenant en compte la dépendance aux données. Chacune des données est prise au début de l'exécution de la tâche, et relachée à la fin.

La présence d'un ensemble de tâches implique la présence d'un ordonnanceur. Dans le cadre de ces travaux de thèse, à chaque ordonnanceur est associé une stratégie présentée dans la section 3.2. Dans ce choix de modélisation, la notion d'erreur détectable couvre deux aspects, qui peuvent éventuellement être couplés. Soit une tâche échoue parce qu'au moins une des données d'entrée a un statut *err*, auquel cas la tâche s'exécute correctement mais produit un résultat erroné. Soit les données d'entrée sont valides, mais la rafale de fautes affecte l'exécution de la tâche, ce qui conduit aussi à un résultat erroné.

La détection d'erreur entraîne alors une méthode de correction, avec une incertitude sur l'origine de l'erreur. Ainsi, le contexte d'exécution de la méthode de correction d'erreur est à préciser. Lorsque, par exemple, un test d'acceptance échoue, les données prises par la tâche erronée sont libérées. Ce processus permet alors de mettre à jour les données, palliant éventuellement à une origine de défaillance due aux données. Ensuite ces données sont reverrouillées, puis la méthode de correction est appliquée. De façon évidente, suivant la durée de la rafale de fautes, la possibilité que plusieurs applications de méthodes correctives soient nécessaires pour aboutir à un résultat juste est réaliste.

Formalisation d'une ressource de calcul Sur les bases de la description fonctionnelle précédente, la formalisation de la ressource de calcul conduit à la définition suivante :

Définition 8 *Ressource de calcul*

Une ressource de calcul est un n-uplet $R = (C, \Gamma, ordo, strat)$ *tel que :*
- C *est un composant, tel que* $C = (s, res, rec, I, O, F)$
- $\Gamma = \{\tau_1, ..., \tau_n\}$ *est un ensemble de n tâches*
- *ordo est un algorithme d'ordonanncement*
- *strat est une stratégie de recouvrement d'erreurs*

Soit α *la fonction qui permet d'associer les fonctions du composant avec les tâches de la ressource :*

$$\begin{array}{rccc} \alpha : & F & \longrightarrow & \Gamma \\ & f_j & \longmapsto & \tau_i \end{array} \tag{8.2}$$

La fonction α *est injective :*

$$\forall (f_1, f_2) \in F \times F,\ (\alpha(f_1) = \alpha(f_2) \implies f_1 = f_2) \tag{8.3}$$

La fonction α, et sa propriété d'injectivité, permettent donc de garantir qu'une tâche implante au plus une seule fonction du composant lié à la ressource de calcul. Par transitivité, une tâche devient donc dépendante des données de la donction qu'elle implante. Il en est de même pour la validité de la donnée de sortie, liée au calcul fait par la tâche.

En revanche le nombre de tâches peut être supérieur au nombre de fonctions du composant ($card(\Gamma) \geqslant card(F)$). En effet, certaines tâches peuvent être indépendantes vis à vis des données entrantes dans la ressource de calcul, et n'avoir aucun lien de précédence avec d'autres tâches.

Par ailleurs, la ressource de calcul possède un statut interne. Le flot de données émis par cette ressource sera donc fonction de ce statut. Ainsi, ce chapitre considère que si, à l'isssue de la terminaison d'une instance de tâche, la donnée produite correspondante est de statut ok, mais que le statut interne du composant est err, la donnée émise sera de statut err.

Retour sur l'exemple introductif Appliqué à l'exemple introductif, le calculateur se modélise donc par l'intermédiaire d'une ressource de calcul. Ainsi, son composant C interne, possède un statut s, une résilience infinie pour la perturbation considérée, et un recouvrement nul. Son ensemble de donnée d'entrées est réduit à la donnée d, et une donnée o de sortie est considérée, permettant ainsi d'observer le résultat de la fonction. Les données d et o sont liées par une fonction f. Le composant C est donc tel que $C = (s, \infty, 0, \{d\}, \{o\}, \{f\})$.

Le modèle du calculateur est complété par la définition de l'ensemble de tâches Γ qui contient une tâche τ_1. Par l'intermédiaire de la fonction α issue de la définition de la ressourde ce calcul, τ_1 est associée à f. Enfin, l'algorithme d'ordonnancement est *Rate Monotonic* ($ordo = RM$) et la stratégie de recouvrement d'erreurs ED/FR/S (section 3.2).

8.3.4.3 Connecteur

Un **connecteur** décrit une ressource permettant de connecter deux équipements d'une architecture matérielle. Dans le choix de modélisation, le connecteur modélisé peut être soit un câble (A429 [AEE04], CAN [Sta03]), être un réseau au complet (A664 [AEE09], avec switchs). Ces différents connecteurs peuvent être point à point, ou d'une source vers plusieurs destinations.

Description fonctionnelle Dans un système temps réel, la prise en compte des délais dus à la transmission des données est essentielle. Cependant, l'évaluation de ces délais sortant du cadre de la thèse, ces délais sont supposés connus. Ce sont ainsi des données d'entrée de la modélisation. Pour approfondir la question de l'évaluation des temps

de transmission dans les réseaux, ce manuscrit renvoie le lecteur aux travaux sur les méthodes de trajectoires [MM06], le *Network Calculus* [LBT01], ainsi qu'aux documentations techniques relatives au type de réseaux.

Ainsi, vis-à-vis d'une donnée d, un connecteur se comporte donc comme une fonction de translation temporelle. Le statut interne de la donnée transférée dépend alors du statut interne de la donnée entrante, et du statut interne du connecteur.

Formalisation d'un connecteur Ce paragraphe formalise la description fonctionnelle précédente d'un connecteur. Un connecteur est alors un composant particulier dont les fonctions associent à une donnée d'entrée une donnée de sortie décalée dans le temps. Cette formalisation aboutit à la définition suivante :

Définition 9 *Connecteur*

Un connecteur est composant $C = (s, res, rec, I, O, F)$ *tel que chaque fonction* $f_i \in F$ *est définie comme suit :*

$$\begin{array}{rccc} f_i : & I & \longrightarrow & O \\ & i_i & \longrightarrow & o_i = (i_i.s \wedge C.s, \delta_i^- + \theta^-, \delta_i^+ + \theta^+) \end{array} \tag{8.4}$$

Les valeurs θ^- *et* θ^+ *représentent respectivement le meilleur temps de transmission et le pire de transmission de la donnée* i_i *sur le connecteur. Le composant* C *est tel que* $Card(I) = Card(O)$*. Chaque fonction* f_i *est bijective.*

Retour sur l'exemple introductif Dans l'exemple introductif, le réseau n'a ni résilience, ni recouvrement. Son rôle est de transmettre la donnée d entre le capteur et le calculateur, son meilleur temps de transmission étant de 5 ms et son pire temps de transmission 10 ms.

Appliqué à l'exemple introductif, le réseau se modélise alors comme un connecteur L. La fonction f_d du connecteur L permet donc de prendre en compte la latence du réseau dans la transmission de la donnée d entre le capteur et le calculateur. Cette fonction est telle que :

$$\begin{array}{rccc} f_d : & I & \longrightarrow & O \\ & d & \longrightarrow & o = (L.s \wedge d.s, \delta^- + 5, \delta^+ + 10) \end{array}$$

En reprenant le schéma d'émission du capteur Ca de la figure 8.3, le décalage temporel dû au réseau est alors observable sur la figure 8.4.

8.4 Formalisation des perturbations

L'introduction des notions de résilience et de recouvrement dans la définition d'un composant matériel permet d'ouvrir la discussion autour des rafales de fautes. La définition donnée dans la section 3.1 suppose que la perturbation résultante, décrite sous forme

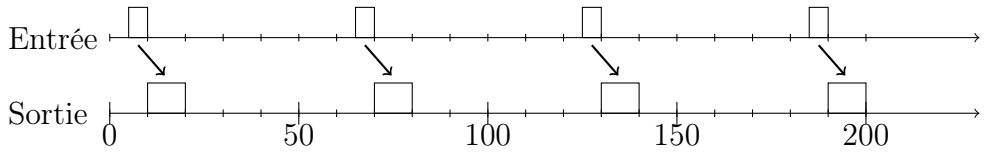


FIGURE 8.4 – Modélisation de l'impact du réseau sur la transmission de la donnée d

de rafales de fautes, est égale à la perturbation subie. Et, de fait, c'est cette rafale de fautes qui est une donnée d'entrée de toute analyse d'ordonnançabilité tolérant aux fautes. Ainsi le besoin de définir la perturbation subie, afin de la dissocier de la perturbation résultante, se fait sentir.

Cette section se structure comme suit. La sous-section 8.4.1 présente la formalisation des perturbation subies. Puis la sous-section 8.4.2 décrit la formalisation des perturbations résultantes, qui tient compte des caractéristiques du composant impacté.

8.4.1 Perturbation subie

Tout phénomène perturbant le système est décrit par un ensemble d'intervalles temporels. Chaque intervalle de temps représente l'exposition du système à une fraction de l'ensemble de la perturbation subie. Ainsi, une perturbation peut être une séquence apériodique d'intervalles, ou une séquence périodique d'intervalles.

8.4.1.1 Perturbation séquentielle

Une perturbation séquentielle, ou apériodique, décrit un phénomène dont les effets sont transitoires ou sans mode connu. Lorsqu'un aéronef traverse une zone de turbulences, ou une automobile roule sur une chaussée mal entrenue, l'ensemble du système est soumis à une séquence de perturbations de durée finie. La définition suivante formalise la perturbation séquentielle :

Définition 10 *Perturbation séquentielle*

Une perturbation séquentielle, notée σ, est un ensemble fini de couples (S, E) tel que $\sigma = \{(S_1, E_1), ..., (S_N, E_N))\}$.

Les valeurs S et E représentent les bornes de l'intervalle. La valeur S décrit le décalage dans le temps par rapport au début de la perturbation prise dans son ensemble.

La longueur de la perturbation, notée λ, représente la durée écoulée entre le début du premier intervalle, et la fin du dernier, soit $\lambda = E_N - S_1$.

La définition ci-dessus décrit des intervalles décalés dans le temps par rapport au début de la perturbation « prise dans son ensemble ». Cette terminologie signifie qu'une

perturbation peut exister, sans pour autant que le système temps réel ne subisse d'effets néfastes. Plus avant, les effets néfastes ne sont ressentis qu'à certains moments.

Par exemple, si un avion se trouve dans une zone de turbulences, certaines d'entre elles pourront être plus violentes que d'autres, les premières générants des effets visibles sur l'aéronef. De même, des phénomènes comme la foudre peuvent frapper l'aéronef de manière aléatoire, suivant des durées différentes. La prise en compte de ces différents types de phénomènes externes à l'aéronef nécessitent le modèle de perturbation séquentielle.

Dans la conception d'un système temps réel, plusieurs perturbations séquentielles σ peuvent être prises en compte, suivant l'origine de la perturbation. L'ensemble des perturbations séquentielles subies est noté Σ.

Exemple L'ensemble fini de couples $\sigma = \{(1,2),(4,6),(9,10),(11,14)\}$ définit ainsi une perturbation séquentielle composée de quatre intervalles de temps. La figure 8.5 en donne une représentation graphique. La longueur de la perturbation est alors $\lambda = 13$ unités de temps.

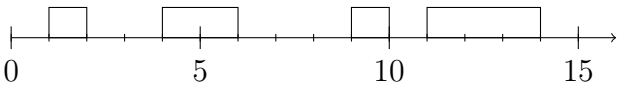


FIGURE 8.5 – Exemple de perturbation séquentielle

8.4.1.2 Perturbation périodique

Une perturbation périodique étend le modèle mathématique de la rafales de fautes. Par définition, une rafales de fautes correspond à un unique intervalle de perturbation. Dans ce chapitre, une suite d'intervalles de longueur potentiellement différente est considérée, cette suite se répétant avec une période connue. Dans un mode vibratoire connu (par exemple la vibration due à un turbopropulseur), les fréquences affectant le système peuvent être réparties sur différents intervalles de temps. Ce mode ayant une période d'occurences connue, le tout est alors modélisable sous la forme d'une perturbation périodique.

Définition 11 *Perturbation périodique*

Une perturbation périodique, notée π, est un couple (σ, T) où σ suite finie de couples (S,E) tel que $\sigma = \{(S_1,E_1),...,(S_N,E_N))\}$ et T est la période de répétition de la suite.

Dans la conception d'un système temps réel, plusieurs perturbations périodiques π peuvent être prises en compte, suivant l'origine de la perturbation. L'ensemble des perturbations séquentielles subies est noté Π.

Exemple Le couple $(\sigma, 6)$ où $\sigma = \{(1,2),(4,5)\}$ représente une perturbation de période 6 formée de deux intervalles de temps. La figure 8.6 en donne une représentation graphique. La longueur λ de chaque occurence de la perturbation est égale à 4 unités de temps.

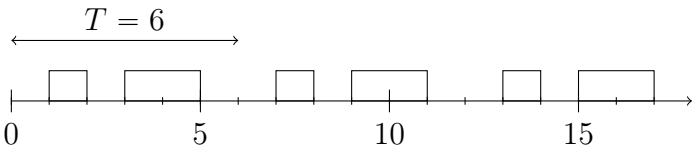


FIGURE 8.6 – Exemple de perturbation périodique

8.4.2 Perturbation résultante

Chaque composant, au travers de ses caractéristiques de résilience et de recouvrement, réagit différemment à une perturbation donnée. Cette réaction est appelée au sein de ce chapitre d'ouverture la **perturbation résultante**.

Les différents intervalles de temps définissant une perturbation subie sont déformés par la prise en compte de la résilience et du recouvrement du composant perturbé. La définition ci-après présente donc la formalisation de la pertubation résultante.

Définition 12 *Perturbation résultante*

Soit C un composant, avec res sa résilience et rec son temps de recouvrement.

Si σ est une perturbation séquentielle qui affecte le composant C, alors la perturbation résultante, notée σ_r est définie comme suit :

$$\sigma_r = \{(S_1 + res, E_1 + rec), ..., (S_N + res, E_N + rec))\})$$

Si π est une perturbation périodique qui affecte le composant C, alors la perturbation résultante, notée π_r est définie comme suit :

$$\pi_r = (\{(S_1 + res, E_1 + rec), ..., (S_N + res, E_N + rec))\}), T)$$

De même que pour les perturbations subies séquentielles et periodiques, les ensembles Σ_r et Π_r sont respectivement l'ensemble des perturbations résultantes séquentielles et l'ensemble des perturbations résultantes périodiques.

8.5 Evolution temporelle du modèle

A présent, la problématique est de pouvoir évaluer, sur la base du modèle précédemment défini, et suivant la perturbation subie, la perturbation résultante et son impact

sur le système. L'objectif de cette section est double : pour chaque composant, proposer une évaluation de la perturbation résultante sur les flots de données. Puis étudier la propagation de cette perturbation dans le système et son impact sur les autres éléments.

La question de la propagation de la perturbation résultante pose indirectement celle de la synchronisation des différents éléments du modèle. En effet, entre la résilience et le recouvrement des divers composants, la périodicité de production et de consommation de données, synchroniser le modèle n'est pas trivial. Pour contourner ce problème, l'approche retenue est conservative.

Une échelle absolue des temps, associée à la perturbation subie, est nécessaire. Suivant cette échelle des temps, le résultat de la perturbation sur chaque composant est évaluée. Ensuite, la logique fonctionnelle est respectée, c'est à dire que le point de vue se concentre sur une donnée, et étudie son parcours dans le système, de son instant de production jusqu'à sa consommation. Lorsqu'un composant nécessite plusieurs données en entrée pour produire une donnée de sortie, le flot de données doit être reconstitué et resynchronisé, dans sa configuration la plus pénalisante pour le composant.

L'approche conservative tient au fait que du point de vue fonctionnel, le pire scénario est toujours considéré. Pour une tâche présente sur une ressource de calcul, cela signifie qu'elle sera toujours synchronisée de telle façon que la configuration des statuts internes des données d'entrée soit la plus pénalisante.

La suite de la section se structure comme suit. La sous-section 8.5.1 présente l'opérateur d'indisponibilité, qui permet d'évaluer le statut interne d'un composant pendant une perturbation. Sur cette base, la sous-section 8.5.2 décrit l'opérateur d'intersection évaluant la perturbation résultante sur les données émises. Enfin, la fonction de translation, présentée dans la sous-section 8.5.3, permet de propager les flots de données dans le modèle.

8.5.1 Fonction d'indisponibilité d'un composant

En réponse à chaque intervalle d'une perturbation subie, le statut interne du composant devient *err*, en tenant compte de l'impact de la résilience et du recouvrement. La perturbation résultante déforme les intervalles de temps en décalant l'instant de passage au statut *err* de la durée de résilience, et en retardant le retour à l'état *ok* de la durée de recouvrement.

Ce comportement se formalise par l'intermédiaire de la fonction d'indisponibilité ψ, définie ci-après.

Définition 13 *Fonction d'indisponibilité ψ*

Soit C un composant et φ une perturbation subie quelconque.

La fonction d'indisponibilité, notée ψ, associe à un couple $(C.s, \varphi)$ où $C.s$ est le statut

interne du composant, le couple (err, φ_r)*, où* err *est le statut interne du composant sur tous les intervalles de la pertubation résultante* φ_r*.*

Si φ *est une perturbation séquentielle, alors* ψ *est définie comme suit :*

$$\begin{array}{rccc} \psi : & \mathcal{S} \times \Sigma & \longrightarrow & \mathcal{S} \times \Sigma_r \\ & C.s, \varphi & \longmapsto & err, \{(S_1 + res, E_1 + rec), ..., (S_N + res, E_N + rec))\} \end{array} \tag{8.5}$$

Si φ *est une perturbation périodique, alors* ψ *est définie comme suit :*

$$\begin{array}{rccc} \psi : & \mathcal{S} \times \Pi & \longrightarrow & \mathcal{S} \times \Pi_r \\ & C.s, \varphi & \longmapsto & err, (\{(S_1 + res, E_1 + rec), ..., (S_N + res, E_N + rec)\}, T) \end{array} \tag{8.6}$$

Retour sur l'exemple introductif Le calculateur est supposé insensible à toute perturbation, ce qui signifie que le statut interne ne change pas au cours du temps. En revanche, l'application de la fonction d'indisponibilité au capteur numérique permet d'en déduire l'évolution temporelle du statut interne. Puisque la perturbation considérée est de type périodique, soit $\Sigma = (\{(0, 150)\}, 700)$, et que la résilience du capteur vaut $5ms$ et son recouvrement $10ms$, alors le capteur est en état erroné sur $(\{(5, 160)\}, 700)$ (figure 8.7).

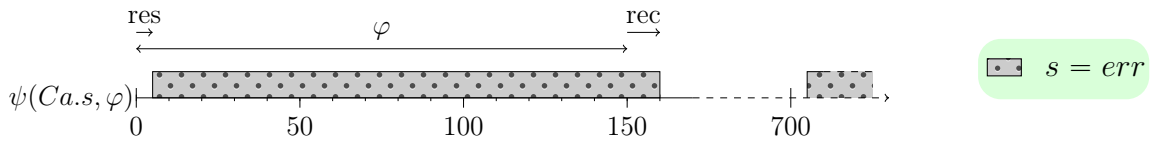


FIGURE 8.7 – Application de la fonction d'indisponibilité au capteur Ca

8.5.2 Fonction d'intersection

Dans ce chapitre d'ouverture, les données sont supposées émises de façon périodiques. Toute émission de donnée intervenant pendant un intervalle où le statut interne du composant est err est considérée comme err. Ainsi, la conséquence de la perturbation subie par le composant est une perturbation résultante au niveau du flot de données.

Cette perturbation résultante correspond à une suite de données de statut interne err, suite qui peut être entrecoupée de données valides de statut interne ok. Le flot de données émis s'évalue comme suit. Chaque donnée possède une période T est un intervalle d'émission $[\delta^-, \delta^+]$.

Devant l'incertitude de l'instant exact d'émission de chaque donnée, ces travaux considèrent que toute intersection entre l'intervalle potentiel d'émission et un intervalle de statut interne err conduit à un donnée de statut interne err. Cette approche, potentiellement pessimiste, évite l'oubli de données perturbées par le comportement du composant.

Ainsi, le problème est avant tout de rechercher les intersections entre les différents intervalles, puis pour chaque intersection trouvée, d'identifier la donnée ayant un statut interne *err*. Pour cela, une fonction d'intersection est définie :

Définition 14 *Fonction d'intersection* $\cap$

Soit d_k *la k-ième donnée émise par le composant C de fonction d'indisponibilité* ψ*. La donnée* d_k *est émise dans l'intervalle* $[\delta^- + kT, \delta^+ + kT]$*.*

La fonction d'intersection, notée $\cap$*, est telle que s'il existe au moins un intervalle de* ψ *dont l'intersection est non vide avec l'intervalle* $[\delta^- + kT, \delta^+ + kT]$*, le statut interne de la donnée* d_k *est err.*

La période d'étude dépend de la forme de la fonction d'indisponibilité. Si cette fonction est périodique, alors la période d'étude correspond au plus petit commum multiple entre la période d'émission T_E des données et la période T liée à ψ. Le nombre d'occurences de données à étudier via la fonction d'intersection est alors fonction de la longueur de la perturbation résultante λ. Ce nombre est au plus égal à $\left\lceil \frac{\lambda}{T_E} \right\rceil$ sur toutes les périodes d'activation de la perturbation concernée. Dans le cas d'une fonction d'indisponibilité séquentielle, le calcul est réduit à $\left\lceil \frac{\lambda}{T_E} \right\rceil$.

Retour sur l'exemple introductif La longueur de la perturbation résultante est de $\lambda = 155$ ($\lambda = 160 - 5$). De fait, il y a alors au plus $\left\lceil \frac{\lambda}{T_E} \right\rceil = 3$ données issues du capteur à considérer (Figure 8.8). Puisque la perturbation résultante est réduite à une fenêtre unique, ces trois données sont dans le pire cas toutes erronées. Le flot de données perturbé résultant de l'application de la fonction $\cap$ correspond donc à trois données erronées sucessives, ce qui correspond, à une durée de perturbation d'émission égale à 180 ms (depuis le début de la première émission, jusqu'au début de la quatrième émission, correspondant à une nouvelle donnée valide).

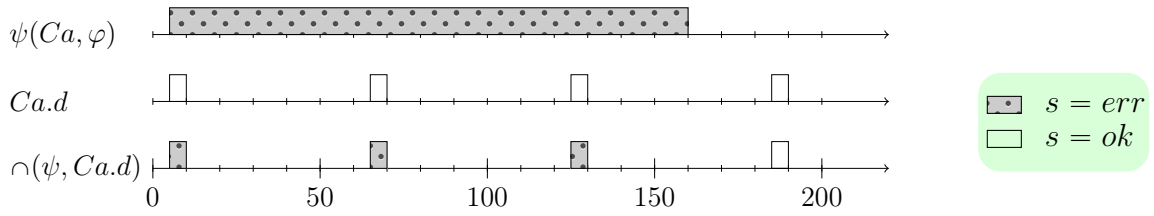


FIGURE 8.8 – Application de la fonction d'intersection au capteur Ca

8.5.3 Fonction de translation

Une donnée d est transmise entre deux composants via un connecteur. L'entrée de la donnée sur le connecteur appartient à l'intervalle $[\delta^-, \delta^+]$, puisque les instants d'émis-

sion et d'entrée sont confondus. De même, depuis l'instant d'entrée, la donnée sort du connecteur au bout d'un temps compris entre $[\theta^-; \theta^+]$.

Ainsi, dans le meilleur des cas, la donnée transite sur le connecteur dans l'intervalle $[\delta^-, \delta^- + \theta^-]$. *A contrario*, dans le pire des cas, la donnée transite sur le connecteur dans l'intervalle $[\delta^+, \delta^+ + \theta^+]$. L'incertitude liée à l'instant d'émission et au temps de transmission conduit, dans une approche pessimiste, à considérer que la donnée peut être présente sur l'union de ces deux intervalles, c'est à dire sur l'intervalle $[\delta^-, \delta^+ + \theta^+]$.

Puisque la définition d'un connecteur est une extension de la définition d'un composant, l'état de la donnée transmise dépend donc de la fonction d'intersection définie dans la précédente sous-section. Si la donnée présente était déjà erronée, alors cette donnée reste erronée. En revanche, si la donnée était valide, elle devient erronée.

L'effet principal du connecteur est, du point de vue de l'échelle de temps absolue, de décaler temporellement la donnée. Du point de vue du composant auquel se destine la donnée, celle-ci ne sera disponible pour être consommée que dans l'intervalle de temps $[\delta^- + \theta^-, \delta^+ + \theta^+]$. La fonction de translation qui définit ce comportement est décrite ci-après :

Définition 15 *Fonction de translation ξ*

Soit D l'ensemble des instances de la donnée d, représentant ainsi le flot de données.

Soit d_k la k-ième donnée émise par un composant source Sce. La donnée d_k est émise dans l'intervalle $[\delta^- + kT, \delta^+ + kT]$.

Soit L un connecteur tel que le temps de transmission de la donnée d_k appartienne à l'intervalle $[\theta^-; \theta^+]$.

La fonction de translation, notée ξ, évalue l'intervalle auquel l'instant de livraison de la donnée à un composant destination Dest appartient :

$$\begin{array}{rccc} \xi : & D \times \mathcal{S} & \longrightarrow & D \\ & d_k, L.s & \longmapsto & d_k = [d_k.s \wedge L.s, \delta^- + kT + \theta^-, \delta^+ + kT + \theta^+] \end{array} \tag{8.7}$$

Retour sur l'exemple introductif Les données issues du capteur et transitant sur le réseau sont donc disponibles pour le calculateur dans l'intervalle $[5 + 60k; 15 + 60k]$. Si la donnée transmise est erronée, alors du point de vue du calculateur, cette donnée est erronée sur l'ensemble de cet intervalle dans le pire cas (figure 8.9).

8.6 Place de l'ordonnanceur dans la modélisation

Une fois le modèle du système établi et les différentes fonctions d'évolution temporelle appliquées, l'analyse peut se centrer sur le rôle de l'ordonnanceur. La composition du flot de données étant dès lors connue, celle-ci devient une entrée du problème d'ordonnancement. En effet, puisque l'exploitation par une tâche d'une donnée de statut *err* conduit à

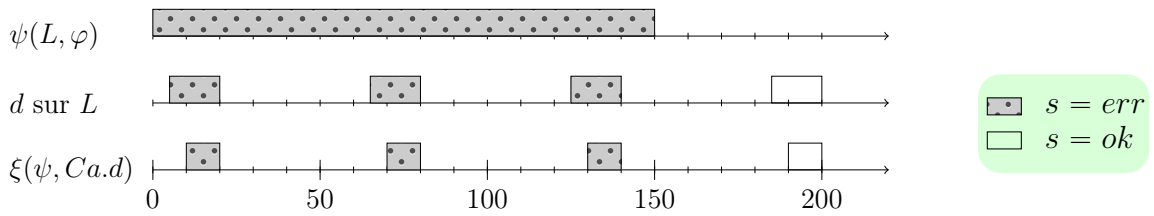


FIGURE 8.9 – Application de la fonction de translation au connecteur L

la détection d'une erreur, le comportement de l'ordonnanceur devient analysable. L'effet de la stratégie de recouvrement d'erreurs est alors mis en évidence, et associé avec la source d'erreurs.

Cette section se structure comme suit. La sous-section 8.6.1 achève l'analyse de l'exemple introductif en abordant la question du rôle de l'ordonnanceur. La sous-section 8.6.2 conclut quant à la pertinence de la modélisation et l'apport de l'analyse de l'ordonnanceur proposée.

8.6.1 Conclusion du point de vue temps réel de l'exemple

La ressource de calculs est reliée au capteur Ca par le connecteur L. Par l'application des fonctions d'indisponibilité, d'intersection et de translation, la séquence de données en entrée de la ressource calcul est alors le résultat de $\xi(\psi, Ca.d)$ (figure 8.10). Dans le pire cas, trois occurences successives de la donnée d ont un statut interne égal à err. La longueur de cette séquence est égale à 190 ms (durée écoulée entre deux données valides). L'analyse de l'ordonnanceur et de son comportement doit donc être effectuée durant cette séquence.

Dans l'exemple introductif, la perturbation subie par le système est sans effet sur la ressource de calcul. Le dysfonctionnement observable sur la tâche τ est de ce fait lié à sa dépendance à l'état des occurences de la donnée d. La première étape de l'analyse est celle de synchronisation. En effet, l'étude de l'ordonnancement doit être faite dans le pire cas.

Ce cas s'établit suivant le raisonnement suivant. Si la donnée prise au début de l'exécution par la tâche τ est valide, alors la terminaison de la tâche sera correcte. Donc, pour que la terminaison de la tâche soit erronée, la donnée d doit donc être de statut err. De plus, à chaque application de la méthode de correction, la donnée est relachée puis reprise. De fait, tant que la donnée est erronée, la méthode de correction se termine de manière erronée. Le pire cas est donc tel qu'au début de la tâche τ la donnée d est erronée et la durée de la séquence la plus longue. La synchronisation est donc telle que présentée sur la figure 8.10.

Depuis cette synchronisation, le comportement de l'ordonnanceur est le suivant. L'occurence de la donnée d correspondant à $k = 0$ est prise par la tâche τ. Cette occurence

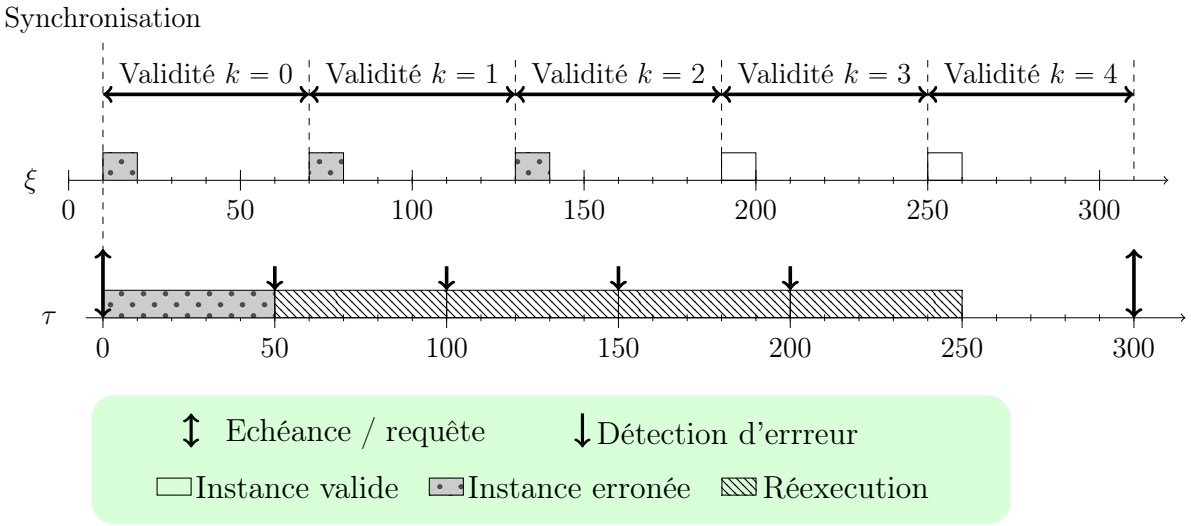


FIGURE 8.10 – Mise en évidence du rôle de l'ordonnanceur dans l'exemple introductif

étant erronée, une erreur est détectée dans la tâche à $t = 50$. Une réexécution complète (stratégie ED/FR/S) débute, l'occurence de donnée étant toujours $k = 0$. Cette réexécution échoue ($t = 100$), et une autre réexécution est appliquée, avec l'occurence de donnée $k = 1$, elle aussi erronée. De même, des réexécutions échouent à $t = 150$ (occurrence $k = 1$) et à $t = 200$ (occurence $k = 2$). La tâche τ se termine correctement à $t = 250$ avec l'occurence $k = 3$ valide de la donnée d. Le pire temps de réponse déduit de la tâche dans cet exemple est égal à 250 ms.

8.6.2 Commentaires

L'exemple introductif met en évidence le rôle de l'ordonnanceur dans la sûreté de fonctionnement. Le calculateur considéré est tel que sa résilience propre rend la perturbation subie sans effet. En d'autres termes, pris individuellement, le calculateur fonctionne normalement. Globalement, l'absence de dépassement d'échéance montre qu'il n'y a pas eu de dysfonctionnement. L'analyse de la trace d'exécution montre en revanche que la stratégie de recouvrement d'erreurs a permis à l'ordonnanceur de gérer un flot de données d'entrée non valides. La présence d'un ordonnanceur tolérant aux fautes contribue donc à une résilience aux données erronées.

8.7 Chaînes descendantes et boucles

Les travaux précédemment décrits sont des étapes initiales dans la conduite d'analyse de systèmes temps réel complexes. Ces travaux de thèse identifient deux problématiques

majeures, les chaînes descendantes et les boucles, qui doivent pouvoir être analyser sur la base des résultats des sections précédentes.

Dans cette section de ce chapitre d'ouverture, ces deux problématiques sont présentées, et illustrées par des cas d'études issus du monde aéronautique. L'analyse des contraintes liées à ces cas d'études permet de se projeter dans les compléments de modélisation nécessaire, et d'anticiper une partie des propriétés qui peuvent émerger de l'analyse de tels systèmes.

En premier lieu, la sous-section 8.7.1 présente un cas d'étude basé sur une chaîne descendante, au travers de la chaîne anémométrique d'un aéronef. La sous-section 8.7.2 adresse d'abord la problématique des boucles entre équipements, puis décrit les questions ouvertes autour des actions collaboratives entre aéronefs.

8.7.1 Problématiques des chaînes descendantes

Une chaine descendante peut se décrire comme une chaine fonctionnelle liant un capteur à un affichage visuel pour les pilotes. Entre ces deux « points », plusieurs composants vont intervenir, réseau, concentrateur de données, convertisseurs et autres composants, qui vont successivement produire, consommer et transmettre des données. La chaine est qualifiée de descendante, car les données ne « remontent » pas la chaîne, *id est*, il n'y a pas d'échange de données bilatéral entre deux composants.

Exemple de système La **chaîne anémométrique** d'un avion de ligne représente une chaîne descendante au sens de la définition donnée ci-dessus. Fonctionnellement, la chaîne anémométrique d'un avion sert à mesurer sa vitesse dans la masse d'air. La valeur mesurée et transmise au pilote est importante du point de vue de la sécurité. Suivant l'attitude de l'avion, une valeur trop importante risque de détruire le fuselage (vitesse maximale à ne pas dépasser), et à l'opposé, une vitesse trop faible présente le risque de faire tomber l'avion dans la masse d'air (décrochage).

Une chaîne anémométrique classique se compose, pour les capteurs, de trois sondes pitot et de trois prises statiques de pression. Dans les avions modernes, tel que l'Airbus A330 (figure 8.11), ces données sont numérisées, via les *Air Data Module* (ADM), puis transmises aux *Air Data and Inertial Reference Units* (ADIRU). C'est dans ces ADIRU que sont ensuite élaborées les valeurs numériques de la vitesse de l'appareil, de l'altitude, etc... Pour des raisons de sûreté de fonctionnement, ces ADIRUs sont au nombre de trois, permettant ainsi d'obtenir des triplets de valeurs indépendants.

Ces triplets sont ensuite transmis aux différents systèmes qui sont connectés à la chaîne anémométrique, tel que le système de commandes de vols électriques ou le système de gestion du vol. Puisque pour chaque paramètre issu des ADIRUs, trois valeurs sont à chaque fois disponibles, des mécanismes de voteurs sont ensuite mis en place par les différents systèmes. Ces mécanismes de voteurs permettent, par exemple, d'écarter une

valeur qui semble incohérente par rapport aux deux autres. Différents mécanismes de voteur existent [Sgh10].

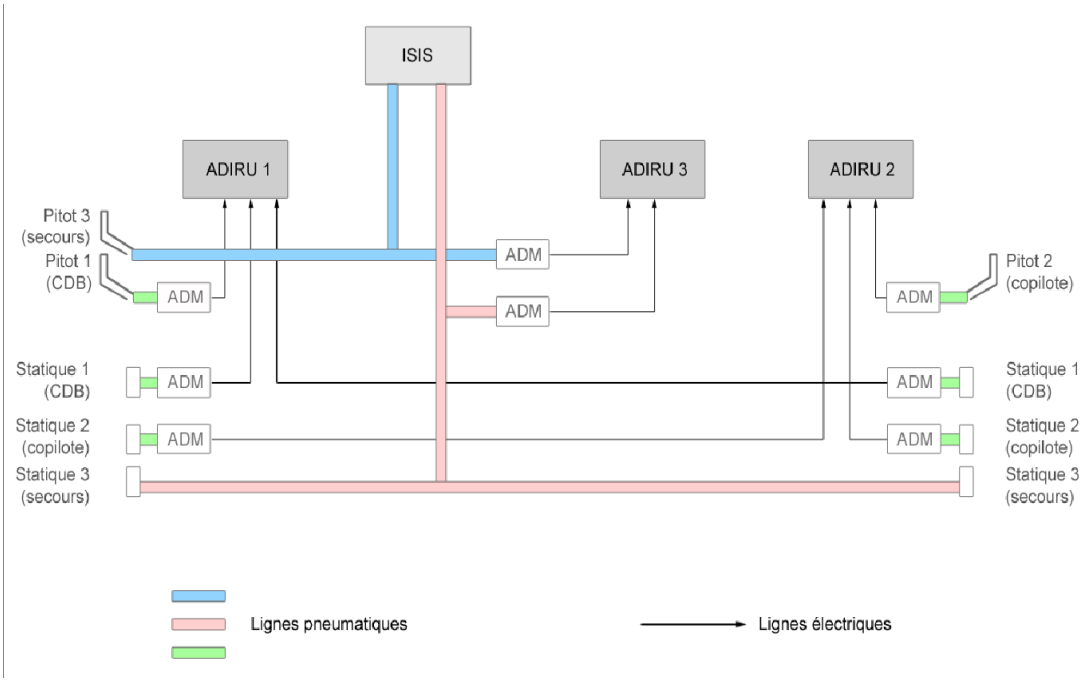


FIGURE 8.11 – Chaîne anémométrique d'un Airbus A330

Problématiques Ces chaînes fonctionnelles sont relativement éprouvées, mais l'incident récent de l'A330 de Quantas [ATS11] a montré que la sûreté de fonctionnement de ce système pouvait cependant encore être étudiée. Du point de vue de ce chapitre d'ouverture, l'analyse d'un tel système (et d'un tel incident) permettrait d'obtenir plusieurs résultats pour compléter ces travaux.

En premier lieu, la modélisation proposée permet de localiser temporellement chaque donnée dans un intervalle de temps. Un premier résultat intéressant serait de vérifier si ces intervalles d'incertitude n'induisent pas de pessimisme qui rendraient le système modélisé non analysable. En second lieu, les composants déjà inclus dans la modélisation pourraient être complétés par des modèles de voteurs et de concentrateurs de données. L'objectif serait alors d'étudier les cas où le voteur pourrait avoir un effet néfaste sur la sécurité du vol, par exemple si la voie donnant les bonnes données ne peut pas être exclue à tort. De même, la question de la disponibilité des données et de leur intégrité pourrait être ajoutée à la modélisation.

Enfin, l'incident de l'A330 montre qu'il existe déjà des stratégies de recouvrement d'erreurs, mais qui dans le cas du vol incriminé, se sont révélées néfastes et non adaptées. A la lumière de ce fait technique, une approche intéressante serait d'intégrer la stratégie

retenue, de modéliser le phénomène perturbateur à l'origine de l'incident, et de vérifier si l'effet néfaste était anticipable sur la base de la modélisation proposée.

8.7.2 Problématiques des boucles

Les systèmes temps réel modernes ne reposent pas que sur des équipements communiquants de manière descendante. Certaines fonctions d'un aéronef reposent sur des équipements interagissant et s'échangeant des données.Ces échanges, qui représentent des boucles au sein du système temps réel, ont généralement pour objectif d'affiner le comportement de l'aéronef dans une situation qui lui est critique.

Exemple de système Le **TCAS** (*Traffic Alert and Collision Avoidance System*) est un système présent sur les avions de ligne présentant une boucle. Fonctionnellement, un TCAS remplit la fonction d'anticollision aérienne. Son rôle est de détecter les aéronefs environnants, d'afficher au pilote ces différents trafics (*TA - Traffic Advisory*), et le cas échéant, d'alerter l'équipage d'un risque potentiel de collision et de générer des manoeuvres d'évitement (*RA - Resolution Advisory*).

Le TCAS est un système collaboratif, *id est* il suppose l'équipement réciproque des avions de ligne en conflit pour que la détection puisse se faire. La composition d'un TCAS est la suivante (figure 8.12). Un calculateur TCAS assure la surveillance de l'espace en interrogeant les transpondeurs des aéronefs environnants, détermine la trajectoire des autres aéronefs, évalue les risques de collision et élabore les manoeuvres d'évitement.

Le calculateur, qui possède des antennes dédiées, est relié au transpondeur de l'avion, qui possède lui aussi ses propres antennes. Par ailleurs, un haut parleur pour les annonces sonores et des visualisateurs (TA-RA VSI) de situation des pilotes sont aussi reliés. L'ensemble du système est connecté aux informations de l'appareil, notamment à la chaine anémométrique, et à un radioaltimère (permettant de vérifier qu'une manoeuvre de descente d'urgence ne sera pas demandée trop basse).

Problématiques Le calculateur TCAS interroge toutes les cinq secondes les transpondeurs des avions environnants. En cas de forte densité de trafic, cette interrogation peut être réduite à toutes les secondes. Une première problématique peut être esquissée ici. En considérant un trafic de forte densité, dans un environnement perturbé par des ondes radars, comment se comporte le calculateur TCAS ? Quelles quantités de données invalides peuvent être gérées, sachant que ce calculateur doit gérer plusieurs tâches, telles que la construction de la situation et le calcul des manœuvres d'évitement. Obtenir la réponse à ce premier point permettrait d'obtenir un exemple concret de calculateur devant gérer des données invalides. La question de la stratégie de recouvrement d'erreurs est alors centrale.

La problématique des boucles se présente dans le lien entre le calculateur TCAS et le transpondeur. Pour obtenir une manoeuvre coordonnée (par exemple, l'intrus monte, et

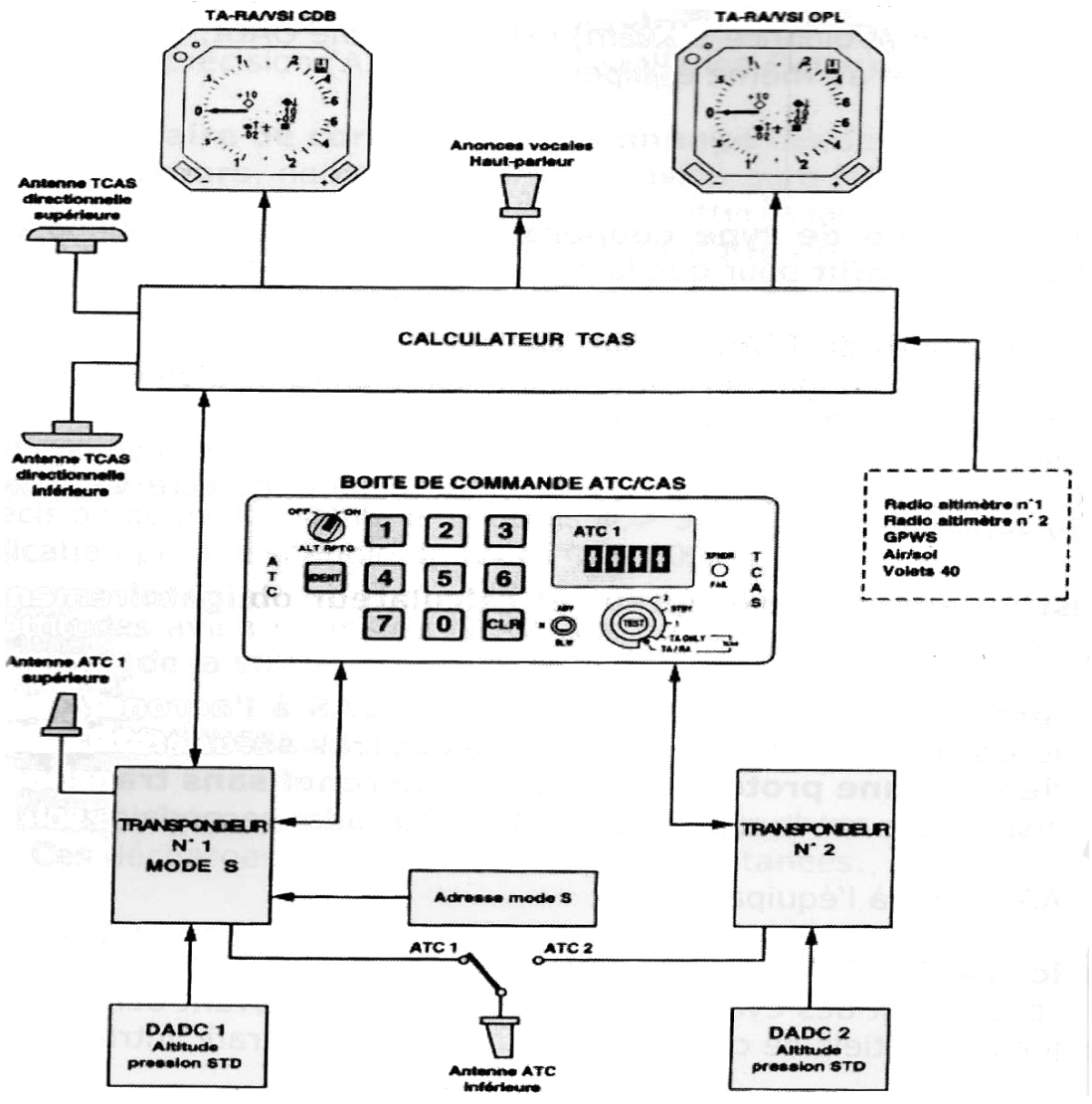


FIGURE 8.12 – Composition d'un TCAS II

l'avion descend), le calculateur TCAS transmet ses données de *Resolution Advisory* au transpondeur, tandis que celui-ci transmet les informations *RA* de l'intrus. De ce fonctionnement ressort une boucle dans laquelle l'information *RA* qui conduit à une manœuvre d'urgence est élaborée à partir de l'information reçue via le transpondeur. La problématique est donc de savoir comment peut se propager une erreur, potentiellement non détectée au niveau du calculateur TCAS. Le risque envisageable, mais à démontrer, serait une « gangrène » du système, *i.e* des données erronnées non détectées qui aboutissent à un fonctionnement erroné du système TCAS.

Ce dernier présente deux axes de recherche. Le premier est l'étude d'un comportement dysfonctionnel du système, centré sur la progression de l'erroné dans le système. Ce phénomène qualifiable de « contamination », deviendrait observable au travers de la modélisation proposée. L'aboutissement de cette contamination serait une gangrène complète du système. L'autre point intéressant serait de lever l'hypothèse d'une couverture parfaite de la détection d'erreurs, et d'étudier des stratégies de recouvrement d'erreurs basées sur une approche imparfaite de la couverture d'erreurs.

8.8 Conclusion

Ce chapitre d'ouverture s'est attaché à formaliser un ensemble de « briques » dans le but d'évaluer l'impact de perturbations durables dans le temps au niveau système, en mettant en évidence le rôle des ordonnancements temps réel tolérants aux fautes. Ainsi, une modélisation du système est proposée permettant de décrire les données échangées, l'architecture du système et les composants matériels.

Les éléments modélisés possèdent un état interne permettant de décrire un fonctionnement correct ou erroné (cas des composants matériels), ou le caractère valide ou erroné d'une donnée. Le modèle de ressources de calcul inclut l'ensemble de tâches, l'ordonnanceur et la stratégie de recouvrement d'erreurs. Un ensemble d'équations est donné pour décrire l'évolution temporelle du modèle et ainsi fournir la dynamique du système.

Sur ce système décrit est appliqué une perturbation. Ces travaux de thèse proposent une formalisation de la perturbation subie par le système, mais aussi une formalisation de la perturbation résultante. La perturbation résultante est la réponse de chaque élément du système à la perturbation subie.

Ainsi, suivant la résilience de l'élément modélisé, l'application de la perturbation subie donne une perturbation résultante, caractérisation temporelle des dysfonctionnements du composant matériel. En connaissant l'effet de la perturbation sur les composants matériels et la dynamique du système, ces travaux montrent comment analyser au niveau système le rôle de l'ordonnancement temps réel tolérant aux fautes. Cette analyse est menée sur un exemple introductif.

In fine, cette approche d'une analyse au niveau système doit être consolidée. L'analyse

des chaînes descendantes (système anémométrique) et des boucles (TCAS) doit permettre de vérifier la validité de l'approche en la confrontant à des systèmes complexes réels.

Conclusions, perspectives et réflexions

Ces travaux de thèse proposent plusieurs contributions dans le domaine de la conception des plateformes avioniques. L'étude combinée des aspects temps réel et sûreté de fonctionnement menée démontre la nécessité d'analyser les impacts bilatéraux de chaque domaine. Ces différentes analyses conduisent à de nouveaux apports pour l'étude des systèmes temps réel. Ces apports sont de plusieurs ordres, et servent ensuite de base aux perspectives sur la suite à donner à ces travaux, et à des réflexions complémentaires issues des résultats présentés.

Conclusions sur les résultats issus de ces travaux de thèse

Rafale de fautes

Ces travaux de thèse proposent d'étudier les systèmes temps réel dans un environnement dont les effets sont perturbants et durables. La modélisation de l'environnement est une première contribution, au travers du modèle en rafale de fautes. Dès lors, la prise en compte des effets des perturbations dues à des radars, ou des vibrations dues aux moteurs, est accessible et intégrable à la validation des systèmes informatiques temps réel.

L'apport de ce modèle est double. Il permet d'abord de considérer que l'occurence de fautes au sein d'un système temps réel peut être durable dans le temps, et périodique. Par sa définition, ce modèle généralise les modèles de fautes pseudo-périodiques et en motif de fautes. En second lieu, ce modèle permet d'envisager des distributions de fautes trop complexes pour être définies. Ainsi, définir la perturbation d'un système sur un intervalle borné revient à considérer non plus une faute « ponctuelle » d'instant nul, mais une faute étendue dans le temps.

Taxonomie de la gestion des erreurs

Sur la base de l'état de l'art, et en adéquation avec l'objectif de valider un système temps réel soumis à des rafales de fautes, ce manuscrit propose une taxonomie des mécanismes de gestion des erreurs. Cette taxonomie s'articule autour des notions de stratégie de recouvrement d'erreurs et de tactique de recouvrement d'erreurs. Elle permet une classification des travaux préexistants sur deux niveaux : la caractérisation du rôle de l'ordonnanceur lors de la détection d'une faute, avec la mise en évidence de l'instant de

détection d'erreurs dans une tâche, et le mécanisme de correction appliqué à la tâche erronée.

Par ce biais, le surcoût temporel dû aux mécanismes de gestion des erreurs est évaluable, et peut alors être une entrée des analyses d'ordonnançabilité. Par ailleurs, la classification de l'état de l'art met en évidence que le rôle de l'ordonnanceur est peu ou pas explicité lors de la détection d'une erreur. Ainsi, la question des corrections anticipées de tâches potentiellement erroné est poussée plus avant que dans les travaux de [PM98].

Ordonnancement tolérant aux fautes

L'analyse de l'impact de la tolérance aux fautes sur les aspects temps réel se traduit dans cette thèse par l'analyse des ordonnancements tolérants aux fautes. L'apport principal réside dans l'établissement d'une équation de pire temps de réponse mettant en évidence la durée de la rafale de fautes et le terme de recouvrement.

Du point de vue purement théorique, l'évaluation d'un pire temps de réponse inférieur à l'échéance relative d'une tâche permet de conclure sur son ordonnançabilité. En se basant sur la taxonomie de la gestion d'erreurs, il est alors possible, pour chaque stratégie de recouvrement d'erreurs, de poser l'équation correspondant à l'évaluation de son surcoût temporel. Ainsi, comme le montre l'étude quantitative, certains ensembles de tâches deviennent faisables lorsque la stratégie change, avec une dominance des stratégies multiples sur les stratégies simples. Cette dominance est un résultat remarquable. En effet, la validation reposant sur l'étude du pire cas, la précision de la stratégie simple, au sens où seules les tâches erronées sont corrigées, est pénalisante. Le fait d'anticiper d'éventuelles erreurs, et de corriger préventivement des tâches se revèle être une bonne stratégie pour valider le système en présence de rafales de fautes.

Du point de vue de la signification de cette analyse d'impact, cette technique de validation met en évidence l'avancée suivante. Lors de l'apparition d'une rafale de fautes, et durant le temps où cette rafale est présente, le comportement du système peut être difficilement prédictible, et donc, le déterminisme du système temps réel est remis en cause. L'approche d'analyse retenue tient compte de ce constat, et répond à ce besoin de déterminisme en garantissant que le système sera capable de se reprendre dans un temps borné. Ainsi, la garantie, au niveau de l'ensemble de tâches, qu'à l'issue de la rafale de fautes il n'y aura pas de dépassement d'échéances permet de valider un système dans l'hypothèse d'un environnement agressif.

Robustesse des ordonnancements

Traditionnellement, la place de l'ordonnancement dans les analyses de sûreté de fonctionnement est centrée autour du caractère potentiellement catastrophique de tout dépassement d'échéances. Ces travaux de thèse proposent de dépasser ce fait en montrant la

contribution de l'ordonnanceur tolérant aux fautes à la sûreté de fonctionnement dans un environnement perturbé.

L'étude de l'impact de l'ordonnancement tolérant aux fautes sur la sûreté de fonctionnement implique deux questions : la robustesse de l'ordonnanceur, et la place de l'ordonnanceur dans l'architecture fonctionnelle et logicielle du système temps réel (cf. analyses de niveau système). Ainsi, une méthode analytique est proposée pour évaluer la durée maximale Δ_F telle qu'un ensemble de tâches soumis à une rafales de fautes soit toujours faisable. Cette durée maximale est dépendante de la stratégie de recouvrement d'erreurs.

De plus, ces travaux de thèse analyse le cas limite $\Delta_F = 0$, et qui ramène le modèle en rafale de fautes au modèle pseudo-périodique. Si la méthode conclut que $\Delta_F = 0$, alors une faute de durée nulle peut impacter toutes les tâches à la fois. En revanche, si la méthode conclut que $\Delta_F < 0$, alors une faute ponctuelle sur toutes les tâche rend l'ensemble de tâches infaisable.

Dans ce cas précis, ces travaux de thèse proposent une méthode analytique permettant de poursuivre l'analyse sous l'hypothèse d'une distribution de fautes pseudo-périodique, et impactant au plus une tâche, comme dans les travaux de [Bur+99]. La méthode proposée offre une méthode directe de calcul de la valeur minimale de la pseudo-période tel que l'ensemble de tâches soit toujours faisable.

Analyses de niveau système

Dernière partie de ce manuscrit, l'ouverture vers des analyses de niveau système est une contribution pour l'analyse et la validation des systèmes temps réel. L'ambition de cette partie est de proposer un modèle de système temps réel mettant en évidence à la fois le caractère dysfonctionnel des composants matériels, leur impact sur les flots de données et le comportement de l'ordonnanceur. Cette modélisation, associée à des équations de propagation des flots de données, permet d'observer le comportement du système face à une perturbation durable dans le temps. Ainsi, la visualisation de l'impact de l'ordonnancement tolérant aux fautes dans la sûreté de fonctionnement du système modélisé devient accessible.

Cette modélisation met en évidence l'importance de la résilience matérielle et de la cadence d'échantillonnage. Une résilience des tâches aux données erronées apparaît dès lors que les tâches deviennent dépendantes à une ou plusieurs données d'entrée. Ces travaux de thèse montrent donc la capacité d'un ordonnancement à tolérer le comportement erroné de son environnement d'exécution, et surtout sa capacité à contribuer à la robustesse du système temps réel pendant une agression extérieure.

Perspectives directes sur la suite à donner à ces travaux

Pour compléter ces travaux, et les étendre, plusieurs voies sont à explorer. Ce manuscrit

en propose quatre, qui concernent le modèle de fautes, les ordonnancements temps réel, l'analyse de niveau système et l'application réelle des résultats issus de cette thèse.

Modèle de fautes

La pertinence du modèle en rafales de fautes pourrait être éprouvée à la lumière de distributions temporelles de fautes issues d'essais ou de retour d'expériences industriels. L'idée sous-jacente serait alors de disposer de valeurs de T_F et Δ_F exactes, afin de pouvoir mener une étude plus précise des effets des perturbations durables dans le temps sur les systèmes temps réel.

Par ailleurs, le cas $\Delta_F = 0$ est la limite entre le modèle en rafales de fautes et le modèle pseudo-périodiques. Envisager la distribution interne de fautes au sein de la rafale, soit sur la base d'une distribution interne pseudo-périodique, soit sur la base d'une distribution stochastique, est une voie de prospection pour faire le lien avec le modèle en motif de fautes.

Ordonnancement tolérant aux fautes

Du point de vue de la validation des ordonnancements tolérants aux fautes, plusieurs points peuvent être explorés. Dans la conception des plateformes avioniques, des tâches informatiques correspondant à des fonctions de différentes criticités peuvent être hébergées sur le même processeur. De fait, la définition de stratégies de recouvrement d'erreurs telles que les tâches les plus critiques soient corrigées prioritairement lors de la détection d'erreurs est un axe à explorer.

De même, la question de l'implantation des tâches critiques et de leur tactique de recouvrement d'erreurs associée se pose. La réactivité du système peut être accrue par l'emploi de mécanismes de correction d'erreurs peu couteux temporellement quitte à dégrader la précision du résultat. Un axe d'étude d'ordonnancements tolérants aux fautes sur la base de tactiques hétérogénes est alors à considérer.

Analyse de niveau système

Ces travaux présentent une modélisation théorique de systèmes temps réel nécessitant une implantation concrète. L'objectif est notamment de pouvoir manipuler des modèles de tailles plus importantes issus de cas d'études industriels. Cette modélisation doit pouvoir s'intégrer au processus de validation, notamment du point de vue des analyses de sûreté de fonctionnement.

Une approche potentielle serait de s'appuyer sur un langage dysfonctionnel existant, tel que l'altarica [Arn+99]. Puisque la modélisation proposée intègre l'ordonnancement temps réel, cet aspect doit être ajouté au langage de modélisation retenu. Les possibilités

sont soit d'étendre le langage altarica en intégrant, par exemple, des automates temporisés [Pag04], soit d'établir un simulateur auquel serait dédié l'évolution temporelle du modèle.

Ainsi, le problème s'orienterait autour du calcul des intervalles de temps contenant l'évolution des états des différents composants du modèle. Un résultat complémentaire, nécessaire lors des phases de conception, serait, via par un exemple un solveur de contraintes, de calculer l'ensemble des résiliences nécessaires (ordonnancement, matériel et aux données) pour que les objectifs de sûreté de fonctionnement soit remplis.

Eprouver les stratégies par un test physique réel

In fine, l'implantation des différentes stratégies de recouvrement d'erreurs est à mettre en pratique au sein d'un système d'exploitation. La réalisation d'un système temps réel, basé sur ce système d'exploitation, et validé suivant les résultats issus de cette thèse est nécessaire pour en évaluer la pertinence. Ce système serait ensuite perturbé, par exemle dans une chambre anéchoïque avec une installation générant les champs électromagnétiques correspondant à la certification des aéronefs décrite dans [SR01]. L'obtention de ce résultat expérimental serait le complément indispensable pour juger en condition opérationnelle de la pertinence des travaux décrits.

Réflexions sur le dimensionnement des systèmes

Ce chapitre consacré aux conclusions et aux perspectives est complété par un ensemble de réflexions sur le dimensionnement des systèmes temps réel embarqués tolérants aux fautes. Plus précisément, la conception de plateformes avioniques ne se limite pas aux seuls avions. Si, de par leurs contraintes vibratoires, les hélicoptères sont un sujet d'étude à part entière, ces réflexions se concentrent sur le domaine des drones.

Les drones ne représentent pas un choix anodin. Sujet de recherche, au travers les questions d'autonomie, notamment décisionnelle, la multiplication des systèmes militaires, et leur émergence dans le domaine civil, a rapidement posé la question de leur développement et de leur dimensionnement. Ces systèmes se distinguent de l'aéromodélisme, à la fois par leur taille, de quelques kilogrammes à plusieurs tonnes, et surtout par leur pilotage déporté jusqu'à plusieurs milliers de kilomètres de l'aire d'évolution du drone.

Aujourd'hui, ces drones se déplacent dans des aires qui leur sont dédiées. Demain, la volonté des concepteurs est de les autoriser à se déplacer au sein des espaces aériens où se trouvent aussi les aéronefs tels que les compagnies aériennes, l'aviation de tourisme ou le transport d'urgence par hélicoptère. Or, cette vision d'avenir impose à la conception de ces drones, du point de vue des autorités aériennes, une démonstration de sûreté de fonctionnement équivalente à celle des aéronefs pilotés certifiés.

Par essence plus contraints en terme de poids et de volume, leur conception nécessitera donc de reposer pleinement sur leurs systèmes informatiques embarqués, et par l'intermédiaire de redondances temporelles. En effet, envisager un niveau de redondance

physique équivalent à celui d'un avion paraît inenvisageable, les coûts de miniaturisation devenant exhorbitants. La démonstration de sûreté de fonctionnement reposera donc sur l'informatique embarquée.

C'est dans ce cadre que s'établissent les réflexions qui vont suivre. Elles présentent, sur la base des résultats déjà établis, un ensemble de considérations en vue du dimensionnement de plateformes avioniques pour drones. Celles-ci traitent des données et de leur échantillonnage. du blindage (ou durcissement) des différents élements d'une plateforme avionique et du couplage entre les résiliences de diverses natures.

Données et échantillonnage des données

Les résultats issus de la section 8.6 mettent en évidence une résilience des ordonnancements aux données invalides, dès lors qu'ils sont conçus comme des ordonnancements tolérants aux fautes intégrant une stratégie de recouvrement d'erreurs. Cette résilience aux données invalides pose plusieurs problèmes liés à la validité des données, leur exploitation par les tâches qui en sont dépendantes et la cadence d'échantillonnement.

Validité des données

Dans le contexte de ces réflexions, les drones, la question de la validité des données peut se poser autour du même exemple de la sous section 3.1.1 qui illustrait les rafales de fautes. La phase d'approche d'un aéroport pour un drone, alors soumis à des perturbations électromagnétiques d'origine radar, se révèle délicate, dans la mesure où l'ensemble des données échangées pendant les balayages radar sont potentiellement erronées.

L'approche classique consiste à protéger les différentes données par des *Cyclic Redundant Check - CRC* et à dupliquer voir tripliquer les données primaires de vol, essentielles pour la sécurité du vol (comme sur la chaîne anémométrique présentée sur la figure 8.11). Cependant, pour les drones, la limitation en ressource de calcul et la limitation en terme de redondances physiques sont des contraintes majeures.

Dès lors, la réflexion de ce manuscrit est de contourner ce problème de dimensionnement en sélectionnant, parmi l'ensemble des données échangées, celles qui, faute d'être redondées, doivent être protégées par CRC. De même, si l'origine de la donnée est elle-même défaillante (telle qu'un capteur), la protection par CRC ne suffira pas à en garantir la validité. L'implantation d'un mécanisme tel que des moyennes sur les x dernières données, permettant d'exclure les données non valides est envisageable. De même, l'usage d'une dissimilarité « indirecte »(par exemple, position GPS et position inertielle), avec des capteurs ne défaillant pas de la même façon soumis à des perturbations, est une approche potentiellement efficace pour des systèmes aux capacités limitées.

Dépendance des tâches aux données

Dans l'exemple introductif 8.2, la tâche τ est dépendante de la donnée d. En l'absence de mécanismes permettant de vérifier la validité des données reçues, la tâche τ doit alors se réexécuter autant de fois que l'occurence de la donnée d est invalide. Le calculateur étant considéré comme insensible aux perturbations, seule une donnée erronée peut faire échouer la tâche.

Ce constat est à replacer dans la problématique des ressources partagées décrites dans la sous-section 1.2.4, et conduit à deux questions. La première concerne l'intégration aux stratégies de recouvrement d'erreurs des contraintes de ressources partagées. En effet, si deux tâches τ_1 et τ_2 exploitent, par exemple en lecture, une même donnée d, quel comportement adopter si une de ces deux tâches échoue à cause d'une invalidité de la donnée. Le scénario suivant peut arriver. La tâche τ_2, moins prioritaire, échoue du fait d'une occurence invalide d'une donnée d. Dans le mécanisme décrit dans ces travaux, l'occurrence de la donnée est relachée pour autoriser une éventuelle mise à jour. De fait, la tâche τ_1 peut alors l'utiliser.

Or, si cette donnée d est erronée, quel sens au niveau de l'ordonnancement peut avoir l'autorisation d'exécution d'une tâche dont l'échec est « programmé » ? Une réponse potentielle est de n'autoriser l'exécution de la tâche τ_1 qu'après une mise à jour, et donc implicitement de marquer l'occurence de la donnée d comme inexploitable pour le système. En corollaire, cette attente de mise à jour devient alors une gigue à prendre en compte, et à borner.

La considération de cette gigue amène à la seconde question. L'évaluation des temps de blocage B_i décrits dans la sous-section 1.2.4 ne tient pas compte de la problématique de la validité des données. Evaluer l'impact de la validité des données dans le calcul de B_i, notamment au travers d'un verrouillage et d'un déverrouillage des ressources partagées interdisant la réutilisation d'une donnée erronée est à analyser. De même, la gigue liée à l'exploitation d'une donnée rafraîchie doit être alors intégrée au calcul du B_i, gigue dépendant de la cadence d'échantillonnage des données.

Echantillonnage des données

En premiere approche, la gigue introduite par l'attente de l'exploitation d'une donnée valide peut être annulée par une cadence d'échantillonnage supérieure à la cadence de verrouillage et déverrouillage d'une donnée. Dans ce contexte, même si une donnée non valide cause l'échec d'une tâche, l'assurance est alors obtenue que soit la réexécution de la tâche, soit la tâche de plus haute priorité utilisant la donnée incriminée exploitera une occurence de cette donnée mise à jour. Si l'occurence d'une donnée non valide est faible, cette approche par suréchantillonnage est suffisante. Le cas du traitement de fautes pseudo-périodiques est alors implicitement traité.

Cependant, l'étude de l'exemple introductif du chapitre 8 montre que la prise en

compte des effets de rafales de fautes est plus complexe. En effet, dans les résultats présentés dans la sous-section 8.6.1, plusieurs données successives sont erronées. De fait, l'augmentation de la cadence d'échantillonnage sera sans effet sur le temps pendant lequel la perturbation est présente. L'évaluation d'une cadence d'échantillonnage devra alors tenir compte, en plus des verrouillages et déverrouillages de la donnée, des durées de perturbations pour lesquelles le système est conçu.

Résilience physique des éléments matériel

La modélisation introduite dans la section 8.3 met en évidence un attribut de résilience physique *res* pour chaque élement décrivant le système. Cette résilience est fonction du blindage de l'élément, de sa position sur le drone et de la nature de la perturbation. Dans un contexte où la masse et l'espace disponible sont limités, le choix des éléments de l'architecture matérielle à protéger ou à renforcer est un élément clé du dimensionnement total de la plateforme.

Dans l'exemple introductif du chapitre 8, le calculateur est insensible à la perturbation. De fait, le comportement du système modélisé pendant la perturbation du système est largement dépendant de cette propriété. Par extension, les propriétés attendues d'un système, par exemple en terme de résilience à une perturbation telle que des ondes électromagnétiques, dépendent des différentes valeurs des résiliences des éléments constitutifs du système.

Ainsi, l'approche suivante peut être envisagée. Dans la modélisation proposée, et dans les fonctions permettant de faire évoluer temporellement le modèle, le paramètre *res* est présent. Trouver la répartition des valeurs de *res* telle que la résilience du système attendue soit atteinte est envisageable. Dès lors, pour chaque élément, plus la valeur de *res* évaluée sera élevée, plus cet élément devra avoir une résilience importante en regard de la perturbation considérée.

Le dimensionnement de la plateforme devra alors tenir compte de ces contraintes, pouvant accroître le poids de la plateforme, au détriment par exemple de la charge utile. De plus, compte tenu du nombre d'éléments constituants la plateforme, plusieurs répartitions de *res* peuvent être possibles donnant lieu à plusieurs architectures matérielles. Un discriminant immédiat pourrait alors être la masse de chaque architecture.

Couplage entre résiliences au niveau calculateur

L'ensemble de ces travaux montre que trois types de résiliences sont à considérer : une résilience physique du calculateur, une résilience aux données erronées et une résilience liée à la robustesse des ordonnancements. Au niveau calculateur, la problématique est donc d'analyser comment ces résiliences se couplent entre elles.

L'observation des relations deux à deux permet d'orienter cette analyse. Pour le couple

résilience physique et résilience liée à la robustesse, le couplage est relativement simple. La résilience de l'ordonnancement est à prendre en compte dès que la résilience physique ne fait plus effet. Ainsi, si la perturbation subie est inférieure à la somme de ces deux résiliences, l'ensemble de tâches hébergées sur le calculateur ne connaîtra pas de défaillances.

De même, le couple résilience physique et résilience aux données erronées est appréhendable. Si des données erronées sont exploitées par des tâches durant la résilience physique, des tâches échoueront, entraînant leur correction. Du point de vue de l'ensemble de tâches, la résilience physique est donc « absorbée »par la résilience aux données erronées.

Le dernier couple concerne la résilience aux données erronées et la résilience liée à la robustesse. Lorsqu'un ordonnancement est exposé à des rafales de fautes, certaines tâches échoueront. Le résultat est le même si des données erronées sont exploitées. De fait, du point de vue de l'ordonnancement, ces tâches seront à corriger. La somme des résidences de ce couple sera donc telle que son début correspondra à la première des deux résiliences entamées, et la fin liée à la dernière deux qui se terminera, en tenant compte bien évidemment des contraintes liées aux échéances temporelles des tâches.

En conclusion, caractériser la résilience globale du calculateur passe par l'analyse cumulée des trois résiliences décrites. Cette caractérisation permettrait ensuite une évaluation de la sûreté de fonctionnement du système exploitant le calculateur, lorsque celui-ci est agressé durablement par une perturbation extérieure.

Bibliographie

[AEE04] AEEC. *Mark 33 Digital Information Transfer System (DITS)*. Rap. tech. ARINC-429. 2004.

[AEE06] AEEC. *Avionics Application Standard Software Interface*. Rap. tech. ARINC - 653. 2006.

[AEE09] AEEC. *Aircraft Data Network - P7 : Avionics Full-Duplex Switched Ethernet Network*. Rap. tech. ARINC-664P7-1. Sept. 2009.

[Arn+99] A. Arnold et al. « The AltaRica Formalism for Describing Concurrent Systems ». Dans : *Fundam. Inform.* 40.2-3 (1999), p. 109–124.

[ATS11] ATSB. *In Flight Upset, 154 km Westof Learmonth, Western Australian, 7 october 2008, VH-QPA, Airbus A330-303*. Rap. tech. AO-2008-070. Déc. 2011, 313 p.

[Aud+91] N.C. Audsley et al. « Hard Real-Time Scheduling : the Deadline Monotonic Approach ». Dans : *Proceedings of the 8th Workshop on Real-Time Operating Systems and Software* (1991), p. 127–132.

[AW96] N. Audsley et A. Wellings. « Analysing APEX Applications ». Dans : *In the Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS)*. 1996, p. 39–44.

[Avi85] A. Avizienis. « The N-version Approach to Fault-Tolerant Software ». Dans : *IEEE Transactions on Software Engineering* 12 (1985), p. 1491–1501.

[Avi+04] A. Avizienis et al. « Basic Concepts and Taxonomy of Dependable and Secure Computing ». Dans : *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), p. 11–33.

[AFG06] T. Ayav, P. Fradet et A. Girault. « Implementing Fault-Tolerance in Real-Time Systems by Automatic Program Transformations ». Dans : *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software*. 2006, p. 205–214.

[Ayd07] H. Aydin. « Exact Fault-Sensitive Feasibility Analysis of Real-Time Tasks ». Dans : *IEEE Transactions on Computers* (2007), p. 1372–1386.

[Bak90] Baker. « A Stacked-Based Resource Allocation Policy for Realtime Processes ». Dans : *Proceedings of the 11th Real-Time Systems Symposium (RTSS)* (1990), p. 191–200.

[BMR90] S.K. Baruah, A.K. Mok et L.E. Rosier. « Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor ». Dans : *Proceedings of the 11th Real-Time Systems Symposium (RTSS)*. 1990, p. 182–190.

[BRH90] S.K. Baruah, L.E. Rosier et R.R. Howell. « Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor ». Dans : *Real-Time Systems* 2.4 (1990), p. 301–324.

[Bar+99] S. Baruah et al. « Generalized Multiframe Tasks ». Dans : *Real-Time Systems* 17.1 (1999), p. 5–22.

[BMR99] A.A. Bertossi, L.V. Mancini et F. Rossini. « Fault-Tolerant Rate-Monotonic First-Fit Scheduling in Hard-Real-Time Systems ». Dans : *IEEE Transactions on Parallel and Distributed Systems* 10.9 (1999), p. 934–945.

[Bro79] A. Brock. « An Analysis of Checkpointing ». Dans : *ICL Technical Journal* 1.3 (1979), p. 211–228.

[BVDT06] M. Bruntink, A. Van Deursen et T. Tourwé. « Discovering Faults in Idiom-Based Exception Handling ». Dans : *Proceedings of the 28th International Conference on Software Engineering*. 2006, p. 242–251.

[BDP96] A. Burns, R. Davis et S. Punnekkat. « Feasibility Analysis of Fault-Tolerant Real-Time Task Sets ». Dans : *In the Proceedings of the 8th Euromicro Workshop on Real-Time Systems (Euro-RTS)*. 1996, p. 29–33.

[Bur+99] A. Burns et al. « Probabilistic Scheduling Guarantees for Fault-Tolerant Real-Time Systems ». Dans : *Dependable Computing for Critical Applications 7*. 1999, p. 361–378.

[CR72] K.M. Chandy et C.V. Ramamoorthy. « Rollback and Recovery Strategies for Computer Programs ». Dans : *IEEE Transactions on Computers* 100.6 (1972), p. 546–556.

[CC89] H Chetto et M Chetto. « Some Results of the Earliest Deadline Scheduling Algorithm ». Dans : *IEEE Transactions on Software Engineering* 15.10 (1989), p. 1261–1269.

[CHS09] M. Chrobak, M. Hurand et J. Sgall. « Algorithms for Testing Fault-Tolerance of Sequenced Jobs ». Dans : *Journal of Scheduling* 12.5 (2009), p. 501–515.

[CR10] J. Craveiro et J. Rufino. « Schedulability Analysis in Partitioned Systems for Aerospace Avionics ». Dans : *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2010, p. 1–4.

[DTB93] R.I. Davis, K.W. Tindell et A. Burns. « Scheduling Slack Time in Fixed Priority Preemptive Systems ». Dans : *Proceedings of the 14th IEEE Real-Time Systems Symposium (RTSS)*. 1993, p. 222–231.

[DS91] X. Delord et G. Saucier. « Formalizing Signature Analysis for Control Flow Checking of Pipelined RISC Microprocessors ». Dans : *Proceedings of the International Test Conference*. 1991.

[DL97] Z. Deng et J. W.S. Liu. « Scheduling Real-Time Applications in an Open Environment ». Dans : *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*. 1997, p. 308–319.

[Der74] M.L Dertouzos. « Control Robotics : the Procedural Control of Physical Process ». Dans : *Information Processing Letters* (1974).

[DG09a] W. Ding et R. Guo. « An Effective RM-Based Scheduling Algorithm for Fault-Tolerant Real-Time Systems ». Dans : *In the Proceedings of the 12th IEEE International Conference on Computational Science and Engineering (CSE)*. 2009, p. 759–764.

[DG09b] W. Ding et R. Guo. « Enhancing the Success Rate of Primary Version While Guaranteeing Fault-Tolerant Capability for Real-Time Systems ». Dans : *Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*. 2009, p. 351–356.

[DAP08] R. Dobrin, H. Aysan et S. Punnekkat. « Maximizing the Fault Tolerance Capability of Fixed Priority Schedules ». Dans : *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2008, p. 337–346.

[Dud83] A. Duda. « Effects of Checkpointing on Program Execution Time. » Dans : *Information Processing Letters* 16.5 (1983), p. 221–229.

[EAM85] A. Ersoz, D.M. Andrews et E.J. McCluskey. *The Watchdog Task : Concurrent Error Detection Using Assertions*. Rap. tech. TR 85-8. Stanford University, Center for Reliable Computing, 1985.

[For+10] J. Forget et al. « Scheduling Dependent Periodic Tasks without Synchronization Mechanisms ». Dans : *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*. IEEE. 2010, p. 301–310.

[Gai94] J. Gaisler. « Concurrent Error-Detection and Modular Fault-Tolerance in a 32-bit Processing Core for Embedded Space Flight Applications ». Dans : *Proceedings of 24th International Symposium on the Fault-Tolerant Computing (FTCS)*. 1994, p. 128–130.

[Gel79] E. Gelenbe. « On the Optimum Checkpoint Interval ». Dans : *Journal of the ACM* 26.2 (1979), p. 259–270.

[GMM95] S. Ghosh, R. Melhem et D. Mosse. « Enhancing Real-Time Schedules to Tolerate Transient Faults ». Dans : *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS)*. 1995, p. 120–129.

[Gho+98] S. Ghosh et al. « Fault-Tolerant Rate-Monotonic Scheduling ». Dans : *Real-Time Systems* 15.2 (1998), p. 149–181.

[Goo75] J.B. Goodenough. « Exception Handling : Issues and a Proposed Notation ». Dans : *Communications of the ACM* 18.12 (1975), p. 683–696.

[HSW03] C.C. Han, K.G. Shin et J. Wu. « A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults ». Dans : *IEEE Transactions on Computers* 52.3 (2003), p. 362–372.

[Hor+74] J. Horning et al. « A Program Structure for Error Detection and Recovery ». Dans : *Operating Systems* (1974), p. 171–187.

[JP86] M. Joseph et P. Pandya. « Finding Response Times in a Real-Time System ». Dans : *The Computer Journal* 29.5 (1986), p. 390–395.

[KHM99] N. Kandasamy, J.P. Hayes et B.T. Murray. « Tolerating Transient Faults in Statically Scheduled Safety-Critical Embedded Systems ». Dans : *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*. 1999, p. 212–221.

[KS86] C.M. Krishna et K.G. Shin. « On Scheduling Tasks with a Quick Recovery from Failure ». Dans : *IEEE Transactions on Computers* 100.5 (1986), p. 448–455.

[KN96] N.V. Kuznetsov et R.A. Nymmik. « Single Event Upsets of Spacecraft Microelectronics Exposed to Solar Cosmic Rays ». Dans : *Radiation measurements* 26.6 (1996), p. 959–965.

[LBT01] J.Y. Le Boudec et P. Thiran. *Network Calculus*. T. 2050. LCNS. Springer Verlag, 2001.

[Lee+11] I. Lee et al. « Survey of Error and Fault Detection Mechanisms ». Dans : TR-LPH-2011-002 (2011).

[Lee+98] Y. H. Lee et al. « Partition Scheduling in APEX Runtime Environment for Embedded Avionics Software ». Dans : *Proceedings of the 5th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 1998, p. 103–109.

[Lee+00] Y. H. Lee et al. « Scheduling Tool and Algorithm for Integrated Modular Avionics Systems ». Dans : *Proceedings of the 19th Digital Avionics Systems Conferences*. T. 1. 2000, p. 1C2–1.

[LSD89] J. Lehoczky, L. Sha et Y. Ding. « The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior ». Dans : *IEEE Real-Time Systems Symposium* (1989), p. 166–171.

[LW82] J.Y.T Leung et J. Whitehead. « On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks ». Dans : *Performance Evaluation* 2.4 (1982), p. 237–250.

[LMM00] F. Liberato, R. Melhem et D. Mossé. « Tolerance to Multiple Transient Faults for Aperiodic Tasks in Hard Real-Time Systems ». Dans : *IEEE Transactions on Computers* 49.9 (2000), p. 906–914.

[LF90] C.C.J. Li et W.K. Fuchs. « CATCH-Compiler-Assisted Techniques for Checkpointing ». Dans : *Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS)*. 1990, p. 74–81.

[LB03] G.M.A. Lima et A. Burns. « An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems ». Dans : *IEEE Transactions on Computers* (2003), p. 1332–1346.

[LB05] G.M.A Lima et A. Burns. « Scheduling Fixed-Priority Hard Real-Time Tasks in the Presence of Faults ». Dans : *Dependable Computing* (2005), p. 154–173.

[LN09] G.M.A. Lima et F.M.S. Nascimento. « Simulation Scenarios : a Means of Deriving Fault Resilience for Real-Time Systems ». Dans : *Proceedings of the 11th Brazilian Workshop on Real-Time and Embedded Systems (WTR)*. 2009.

[LNL10] G.M.A. Lima, F.M.S. Nascimento et V.C. Lima. « Fault Resilience Analysis for Real-Time Systems ». Dans : *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2010, p. 18–23.

[LNL87] K.J. Lin, S. Natarajan et J.W.S. Liu. « Imprecise Results : Utilizing Partial Computations in Real-Time Systems ». Dans : (1987).

[LL73] C.L. Liu et J.W. Layland. « Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment ». Dans : *Journal of the ACM (JACM)* 20.1 (1973), p. 46–61.

[Lu09] C. Lu. « Robustesse du logiciel embarqué multicouche par une approche réflexive : application à l'automobile ». Thèse de doct. 2009.

[MD08] Florian Many et David Doose. *Transformations de modèles pour la validation de systèmes temps réel*. Projet de fin d'études 86. ISAE, 2008.

[MD11a] Florian Many et David Doose. « Fault Tolerance Evaluation and Schedulability Analysis ». Dans : *Proceedings of the 2011 ACM Symposium on Applied Computing* (2011), p. 729–734.

[MD11b] Florian Many et David Doose. « Scheduling Analysis under Fault Bursts ». Dans : *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium* (2011), p. 113–122.

[MM06] S. Martin et P. Minet. « Schedulability Analysis of Flows Scheduled with FIFO : Application to the Expedited Forwarding Class ». Dans : *Proceedings of the 20th Parallel and Distributed Processing Symposium*. Rhodes Island, Greece, 2006.

[MMG06] S. Martin, P. Minet et L. George. « Improving Fixed Priority Schedulability with Dynamic Priority as Secondary Criterion ». Dans : *J. Embedded Computing* 2.3-4 (2006), p. 327–345.

[MBS07] A. Meixner, M.E. Bauer et D. Sorin. « Argus : Low-Cost, Comprehensive Error Detection in Simple Cores ». Dans : *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. 2007, p. 210–222.

[MS07] A. Meixner et D.J. Sorin. « Error Detection Using Dynamic Dataflow Verification ». Dans : *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*. 2007, p. 104–118.

[MR77] P.M. Melliar-Smith et B. Randell. « Software Reliability : the Role of Programmed Exception Handling ». Dans : *ACM SIGOPS Operating Systems Review*. T. 11. 1977, p. 95–100.

[Mir+92] G. Miremadi et al. « Two Software Techniques for On-Line Error Detection ». Dans : *Proceedings of the 22th International Symposium on Fault-Tolerant Computing (FTCS)*. 1992, p. 328–335.

[Mok83] A.K. Mok. « Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment ». Thèse de doct. Department of Electrical Engineering et Computer Science, Cambridge, MA : Massachusetts Institute of Technology, 1983.

[MFC01] A. K. Mok, X. Feng et D. Chen. « Resource Partition for Real-Time Systems ». Dans : *Proceedings of the 7th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2001, p. 75–84.

[MMG03] D. Mossé, R. Melhem et S. Ghosh. « A Nonpreemptive Real-Time Scheduler with Recovery from Transient Faults and Its Implementation ». Dans : *IEEE Transactions on Software Engineering* (2003), p. 752–767.

[NLL09] F.M.S. Nascimento, G.M.A. Lima et V.C. Lima. « Deriving a Fault Resilience Metric for Real-Time Systems ». Dans : *Proceedings of the 10th Brazilian Workshop on Tests and Fault Tolerance (WTF)*. 2009.

[NB91] E. Nassor et G. Bres. « Hard real-time sporadic task scheduling for fixed priority schedulers ». Dans : *of : Proc. Intl. Workshop on Responsive Comp. Syst.* 1991, p. 44–47.

[NSS00] N. Navet, Y.Q. Song et F. Simonot. « Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over Controller Area Network ». Dans : *Journal of Systems Architecture* 46.7 (2000), p. 607–617.

[OSM02] N. Oh, P.P. Shirvani et E.J. McCluskey. « Error Detection by Duplicated Instructions in Super-Scalar Processors ». Dans : *IEEE Transactions on Reliability* 51.1 (2002), p. 63–75.

[OB94] P.M. O'Neill et G.D Badhwar. « Single Event Upsets for Space Shuttle Flights of New General Purpose Computer Memory Devices ». Dans : *IEEE Transactions on Nuclear Science* 41.5 (1994), p. 1755–1764.

[Pag04] Claire Pagetti. « Extension temps réel du langage AltaRica ». Thèse de doct. Ecole Centrale de Nantes, 2004.

[PM98] M. Pandya et M. Malek. « Minimum Achievable Utilization for Fault-Tolerant Processing of Periodic Tasks ». Dans : *IEEE Transactions on Computers* 47.10 (1998), p. 1102–1112.

[Pat10] R.M. Pathan. « Scheduling Algorithms for Fault-Tolerant Real-Time Systems ». Thèse de doct. 2010.

[Pun97] S. Punnekkat. « Schedulability Analysis for Fault-Tolerant Real-Time Systems ». Thèse de doct. University of York, Department of Computer Sciences, 1997.

[PBD01] S. Punnekkat, A. Burns et R. Davis. « Analysis of Checkpointing for Real-Time Systems ». Dans : *Real-Time Systems* 20.1 (2001), p. 83–102.

[PDB97] S. Punnekkat, R. Davis et A. Burns. « Sensitivity Analysis of Real-Time Task Sets ». Dans : *Advances in Computing Science-ASIAN'97* (1997), p. 72–82.

[RE92] RTCA et EUROCAE. *Software Considerations in Airborne Systems and Equipment Certification*. Rap. tech. ED 12B - DO 178B. 1992.

[RE05] RTCA et EUROCAE. *Environmental Conditions and Testprocedures for Airborne Equipment*. Rap. tech. ED 14E - DO 160E. Mar. 2005.

[RE07] RTCA et EUROCAE. *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*. Rap. tech. ED 124 - DO 297. Juin 2007.

[RE11] RTCA et EUROCAE. *Software Considerations in Airborne Systems and Equipment Certification*. Rap. tech. ED 12C - DO 178C. 2011.

[SAE96] SAE. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. Rap. tech. ED 135 - ARP 4761. Déc. 1996.

[SE96] SAE et EUROCAE. *Certification Considerations for Highly-Integrated or Complex Aircraft Systems*. Rap. tech. ED 79 - ARP 4754. Nov. 1996.

[SR01] SAE et RTCA. *Guide to Certification of Aircraft in a High Intensity Radiated Field (HIRF) Environment*. Rap. tech. ED 107 - ARP 5583. Mar. 2001.

[SSO05] R.M. Santos, J. Santos et J.D. Orozco. « A Least Upper Bound on the Fault Tolerance of Real-Time Systems ». Dans : *Journal of Systems and Software* 78.1 (2005), p. 47–55.

[San+04] R.M. Santos et al. « New Methods for Redistributing Slack Time in Real-Time Systems : Applications and Comparative Evaluation ». Dans : *The Journal of Ststems and Software* 69 (2004), p. 115–128.

[Sgh10] Manel Sghairi Houati. « Architectures innovantes de systèmes de commandes de vol ». Thèse de doct. Université de Toulouse, 2010.

[SRL90] L. Sha, R. Rajkumar et J.P. Lehoczky. « Priority Inheritance Protocols : An Approach to Real-Time Synchronization ». Dans : *IEEE Transactions on Computers* 39.9 (1990), p. 1175–1185.

[Sha+04] L. Sha et al. « Real Time Scheduling Theory : A Historical Perspective ». Dans : *Real-time systems* 28.2 (2004), p. 101–155.

[SPD08] F. Singhoff, A. Plantec et P. Dissaux. « Can We Increase the Usability of Real-Time Scheduling Theory ? The Cheddar Project ». Dans : *Reliable Software Technologies–Ada-Europe 2008* (2008), p. 240–253.

[Sin+04] F. Singhoff et al. « Cheddar : a Flexible Real-Time Scheduling Framework ». Dans : *ACM SIGAda Ada Letters*. T. 24. 2004, p. 1–8.

[Spu96a] Marco Spuri. *Analysis of Deadline Scheduled Real-Time Systems*. Rap. tech. RR-2772. 1996, 30 p. url : citeseer.ist.psu.edu/spuri96analysis.html.

[Spu96b] Marco Spuri. *Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems*. Rap. tech. RR-2873. 1996, 74 p.

[Sta03] International Organization for Standardization. *Controller Area Network*. Rap. tech. ISO-11898. 2003.

[Tin94] K. Tindell. *Adding Time-Offsets to Schedulability Analysis*. Rap. tech. YCS-94-221. University of York : Department of Computer Science, 1994.

[TC94] K. Tindell et J. Clark. « Holistic Schedulability Analysis for Distributed Hard Real-Time Systems ». Dans : *Microprocessing and Microprogramming* 40.2-3 (1994), p. 117–134.

[Tor00] W. Torres-Pomales. *Software Fault Tolerance : A Tutorial*. Rap. tech. TM-2000-210616. Langley Research Center, Hampton, Virginia : NASA, 2000.

[TB84] S. Toueg et O. Babaoglu. « On the Optimum Checkpoint Selection Problem ». Dans : *SIAM Jounal On Computing* 13 (1984), p. 630–649.

[Tri08] M.F. Triola. *Elementary Statistics*. Pearson, 2008.

[Ves94] S. Vestal. « Fixed Priority Sensitivity Analysis for Linear Compute Time Models ». Dans : *IEEE Transactions on Software Engineering* 20.4 (avr. 1994), p. 308–317.

[WH90] N.J. Warter et W. Hwu. « A Software Based Approach to Achieving Optimal Performance for Signature Control Flow Checking ». Dans : *Proceedings of the 20th International Symposium on Fault Tolerant Computing Systems (FTCS)*. 1990, p. 442–449.

[ZB97] A. Ziv et J. Bruck. « An On-Line Algorithm for Checkpoint Placement ». Dans : *IEEE Transactions on Computers* 46.9 (1997), p. 976–985.

Combinaison des aspects temps réel et sûreté de fonctionnement pour la conception des plateformes avioniques

La conception des plateformes aéronautiques s'effectue en tenant compte des aspects fonctionnels et dysfonctionnels prévus dans les scénarios d'emploi des aéronefs qui les embarquent. Ces plateformes aéronautiques sont composées de systèmes informatiques temps réel qui doivent à la fois être précises dans leurs calculs, exactes dans l'instant de délivrance des résultats des calculs, et robustes à tout évènement pouvant compromettre le bon fonctionnement de la plateforme.

Dans ce contexte, ces travaux de thèse abordent les ordonnancements temps réel tolérants aux fautes. Partant du fait que les systèmes informatiques embarqués sont perturbés par les ondes électromagnétiques des radars, notamment dans la phase d'approche des aéronefs, ces travaux proposent une modélisation des effets des ondes, dite en rafales de fautes. Après avoir exploré le comportement de l'ordonnanceur à la détection d'erreurs au sein d'une tâche, une technique de validation, reposant sur le calcul de pire temps de réponse des tâches, est présentée. Il devient alors possible d'effectuer des analyses d'ordonnançabilité sous l'hypothèse de la présence de rafales de fautes. Ainsi, cette technique de validation permet de conclure sur la faisabilité d'un ensemble de tâches en tenant compte de la durée de la rafale de fautes et de la stratégie de gestion des erreurs détectées dans les tâches.

Sur la base de ces résultats, les travaux décrits montre comment envisager l'analyse au niveau système. L'idée sous-jacente est de mettre en évidence le rôle des ordonnancements temps réel tolérants aux fautes dans la gestion des données erronées causées par des perturbations extérieures au système. Ainsi, le comportement de chaque équipement est modélisé, ainsi que les flots de données échangés et la dynamique du système. Le comportement de chaque équipement est fonction de la perturbation subie, et donne lieu à l'établissement de la perturbation résultante, véritable réponse dysfonctionnelle de l'équipement à une agression extérieure.

Combination of real-time and safety aspects for the design of avionic platforms

The design of avionic platforms takes into account the functional and dysfunctional aspects, which depend on the aircraft operation concept. These avionic platforms embed computer resources that must produce accurate results at the right time, and must be dependable whatever the disturbance.

In this specific context, we address the topic of fault tolerant real time scheduling. Since the embedded computer resources are disturbed by electromagnetic waves produced by radar, especially during the aircraft approach, we suggest a model of these wave effects named fault bursts. After the analysis of scheduler behaviour when an error is dectected inside a task, we present a validation technique based on the evaluation of the worst case response time. By this way, we are able to study the task set feasibility under fault burst assumption and according to the error recovery strategy.

Then, based on these results, we show a way to analyse the effects of disturbances such as electromagnetic waves at system level. The underlining idea is to demonstrate the main role of fault tolerant real time scheduler in the management of erroneous data. To do that, we suggest an equipment model which integrates the behaviour of the equipement when a disturbance occurs. We also describe the data flows in order to describe the avionics platform dynamics.