THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par **l'Institut Supérieur de l'Aéronautique et de l'Espace**
Spécialité : Systèmes embarqués

Présentée et soutenue par **Jean-Charles CHAUDEMAR**
le 27 janvier 2012

Étude des architectures de sécurité de systèmes autonomes. Formalisation et évaluation en Event B

JURY

Mme Régine Laleau, présidente
M. Yamine Aït-Ameur, rapporteur
M. Éric Bensana, co-directeur de thèse
M. Dominique Méry, rapporteur
Mme Marielle Petit-Doche
Mme Christel Seguin, directrice de thèse

École doctorale : **Systèmes**

Unité de recherche : **Équipe d'accueil ISAE-ONERA CSDV**

Directrice de thèse : **Mme Christel Seguin**
Co-directeur de thèse : **M. Éric Bensana**

Remerciements

Le moment des remerciements tant attendu est un grand moment d'exaltation où l'on se dit que l'on a enfin réussi ce nouveau défi. Mais ne vous y trompez pas ! Cela n'a pas été une mince affaire ! Je n'y serais jamais arrivé sans l'aide, les encouragements, les conseils d'autres personnes. Alors, pour démarrer cette valse des remerciements, je souhaite remercier tous les membres du jury de soutenance qui ont été très enthousiastes et très encourageants. Ils ont fait de cet instant "un instant d'échanges scientifiques de haut niveau" dont je me souviendrai longtemps. Parmi ces membres, j'adresse de sincères remerciements à mes directeurs de thèse Eric Bensana et Christel Seguin, ainsi que Charles Castel parti à la retraite au cours de ma thèse. Chacun à sa manière m'a fait partager son savoir et son expérience de chercheur averti.

Je veux également remercier Patrick Fabiani, le Directeur du Département de Commande des Systèmes et Dynamique du vol (DCSD) de l'ONERA, Jean-François Gabard puis Jean-Loup Farges, chefs de l'unité de recherche Conduite et Décision, qui m'ont chaleureusement accueilli dans leur équipe. Ils ont tout mis en œuvre pour que je puisse mener ma recherche dans les meilleurs conditions. Merci à l'ensemble des chercheurs de l'ONERA que j'ai côtoyés tout au long de ma thèse et avec qui je souhaite encore collaborer. Merci aux nombreux doctorants et aux stagiaires qui m'ont aidé, en particulier Romain Adeline avec qui j'ai beaucoup discuté et qui savait toujours trouver les mots justes pour me remotiver, Julien Guitton qui m'a souvent dépanné avec LaTeX.

Je tiens à remercier Jean-Henri Llareus, Professeur de SUPAERO à la retraite, qui m'a guidé dans le monde académique. Il m'a fait découvrir la joie de la recherche et de l'enseignement. Il m'a également appris à développer des convictions et à me battre pour celles-ci. J'en profite pour remercier tous mes collègues de l'ISAE qui ont largement contribué à la bonne marche de ma thèse par l'aménagement de mes activités et leur soutien permanent.

J'adresse une spéciale dédicace à mes parents, à mes frères et sœurs, à ma famille restée en Guadeloupe lors de ma soutenance. Mes pensées m'ont perpétuellement conduit auprès d'eux avec fierté afin d'être digne de leur confiance. Qu'ils continuent à me donner la force pour être encore meilleur ! Ce désir est

encore plus intense depuis que je suis père. Grâce à mes enfants, je me suis vu faire des choses que je n'aurais jamais faites auparavant. Alors, je remercie ma tendre épouse Véronique pour son soutien, sa patience dans les moments difficiles qu'elle a dû endurer pendant cette période de thèse.

Bien sûr n'y voyez aucun acte intentionnel si j'ai oublié de faire explicitement allusion à votre participation dans cette aventure. En signe de gratitude, je vous adresse à tous également mes remerciements.

À mes enfants chéris
d'amour que j'aime
de tout mon cœur

« *Je crois beaucoup en la chance ; et je constate que plus je travaille, plus la chance me sourit.* »
Thomas Jefferson

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction Générale

« *Que ceux qui ont en vue des applications jugent avec indulgence les tentatives pour préparer le futur.* »
PIT 85

Avec l'essor des technologies modernes, on assiste au développement de nombreux systèmes complexes capables d'agir, de réagir, voire de décider. Ces systèmes autonomes sont amenés à évoluer dans des environnements où les risques pour l'être humain ne sont pas négligeables. Par conséquent, l'un des enjeux principaux de conception réside dans la maîtrise des risques et la sûreté de fonctionnement (SdF) pour le contrôle de ces systèmes. En outre, la complexité des systèmes peut se traduire par un nombre important d'éléments constitutifs, mais également par une connaissance limitée de certains phénomènes. Elle peut alors être maîtrisée par l'interaction et l'échange d'information qui sont des piliers du fonctionnement sûr des systèmes actuels.

Un des objectifs de cette thèse a été de donner des guides sur la construction d'architectures et en particulier de l'organisation des mécanismes de sécurité, pour des systèmes complexes autonomes. Un autre objectif visé par cette thèse a été de proposer des méthodes et des outils pour prouver la sécurité des architectures étudiées. Nous pensons qu'une formalisation des architectures fournit un cadre rationnel de modélisation et de raisonnement utile pour l'analyse de sécurité.

L'approche adoptée associe le pragmatisme fondé sur un retour d'expériences sur l'application des procédés de conception dans les domaines aéronautique et robotique mobile, et la rigueur apportée par la formalisation des architectures. On montre la possibilité d'utiliser une méthode formelle depuis une description simple et abstraite jusqu'à une représentation détaillée du comportement du système considéré : approche à base de raffinements par la méthode Event-B. D'autre part, notre approche s'appuie sur la décomposition des systèmes dits "embarqués" en composants matériels et en composants logiciels, afin de mettre en évidence les propriétés fonctionnelles de ces composants associées à des exigences de sécurité.

Le mémoire de thèse est organisé comme suit : dans le premier chapitre, nous présentons les tenants et les aboutissants de la SdF. Étant donné un contexte critique lié aux systèmes embarqués autonomes, nous introduisons la notion de SdF et quelques concepts généraux associés. Ensuite, nous décrivons un processus courant pour évaluer la sécurité d'un système, ainsi qu'un processus de conception classique.

Dans le deuxième chapitre, nous nous intéressons plus particulièrement à la conception des architectures dédiées à la SdF en présentant les principes sous-jacents et un état de l'art succinct de ces architectures et des mécanismes associés. Nous concluons ce chapitre par une présentation de l'architecture adoptée pour notre étude.

Ensuite, dans le troisième chapitre, nous présentons le formalisme retenu pour spécifier notre architecture : Event-B. Nous débutons ce chapitre par un rappel sur la logique des prédicats du premier ordre, à la base de la notation et

de la vérification en Event-B. Nous présentons et détaillons le langage formel Event-B en mettant l'accent sur son principe de raffinement et sur ses règles d'obligations de preuve. Seules les grandes lignes du cadre de raisonnement sont données pour vérifier la correction de modèles réalisés en Event-B.

Puis, dans le quatrième chapitre, nous formalisons le problème étudié en modélisant en Event-B une architecture générique en couches fonctionnelles intégrant des mécanismes de SdF. Nous présentons le processus de modélisation adopté ainsi que les modèles développés couche par couche. Dans chaque couche, nous validons notre modélisation vis-à-vis des propriétés de sécurité spécifiées.

Dans le cinquième chapitre, nous détaillons le système de contrôle du drone autonome visé en application et exposons une instanciation des modèles développés au cas d'étude de ce drone. Pour clore ce chapitre, la vérification de notre modélisation par cette instanciation est complétée par la validation des règles d'obligation de preuve lors de la modélisation.

Enfin, dans le sixième chapitre, nous dressons une liste non exhaustive de travaux similaires sur les architectures et les mécanismes de sécurité, ainsi que quelques formalismes concurrents et complémentaires à Event-B.

Nous concluons cette thèse par une synthèse de nos travaux et présentons les perspectives qui nous paraissent les plus intéressantes.

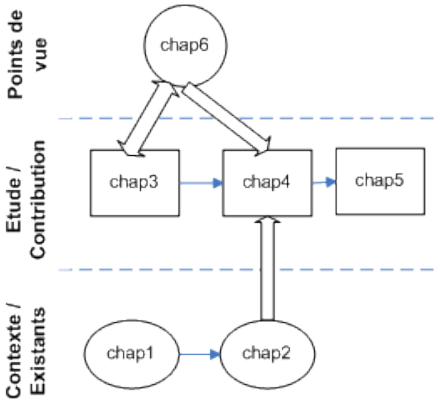

FIG. 1 – Schéma de lecture.

Chapitre 1

La problématique de sûreté de fonctionnement

Résumé
La sûreté de fonctionnement (SdF) et la maîtrise des risques forment une discipline importante pour la conception des systèmes embarqués critiques. Elles analysent les dysfonctionnements et les effets des défaillances au même titre que les fonctionnements sûrs pour rendre acceptables les conditions d'occurrences de fautes. Elles s'appuient pour cela sur des concepts et des méthodes de conception et d'évaluation introduits dans ce chapitre.

« Il est dans l'ordre des choses que nous soyons déçus, comme nous-mêmes, nous passons notre vie à décevoir ceux qui ont mis leur foi en nous . . . »
Gilbert Choquette - La défaillance

"L'échec n'est pas une option" (en anglais, "Failure is not an option"). Telle est la célèbre phrase de Gene Kranz, directeur de vol d'Apollo 13, qui lui a permis de motiver ses troupes et de mobiliser toutes les énergies du Centre de Contrôle de Houston à ramener sain et sauf les trois astronautes d'Apollo 13 au terme d'une opération sans précédent de sauvetage dans l'espace. Bien qu'ayant avorté, cette mission lunaire fait partie des grands moments de l'histoire de l'Homme dans l'espace.

Tout système est par essence susceptible de défaillir avec des conséquences variées pour l'environnement opérationnel, mais également pour le constructeur. Comme on ne peut maîtriser rigoureusement la conception et les conditions opérationnelles d'un système complexe, tout l'art du fiabiliste (le responsable de la SdF) est de rendre peu probable tout arrêt de service ou tout échec du système entraînant des conséquences inacceptables sur le plan humain ou économique par exemple, selon le système considéré. Il dispose à cet effet d'une batterie de méthodes et de techniques lui permettant d'analyser rigoureusement le fonctionnement anormal. Ainsi, les exploits de nombreux projets aéronautiques et spatiaux sont le résultat des nombreuses études menées en SdF permettant d'appréhender les défaillances et leurs effets et de prendre en compte très tôt cette problématique dans la conception d'un système. Cela est d'autant plus vrai lorsque le système en question est un système embarqué autonome dont la complexité peut être génératrice de nouvelles défaillances.

Le plan de ce chapitre se présente comme suit : dans la section suivante (section 1.1), on prend soin de définir les spécificités des systèmes embarqués autonomes pour comprendre l'intérêt d'une démarche rigoureuse en SdF dans la conception de ces systèmes. Mais, avant d'aborder la démarche, on pose dans la section 1.2 les concepts et les méthodes qui sous-tendent cette discipline qu'est la SdF. En fait, la terminologie explicitant la dynamique des défaillances (sous-section 1.2.1) permet de dégager des éléments d'une défaillance les attributs qui la caractérisent (sous-section 1.2.2). Il en découle des propriétés garantissant la qualité de la SdF (sous-section 1.2.3) que l'on évalue par des méthodes spécifiques (sous-section 1.3.2). Enfin, la section 1.3 de ce chapitre présente une démarche rigoureuse d'analyse de sécurité basée sur des normes (sous-section 1.3.1), et s'appuyant sur des activités importantes de vérification et de validation (V & V) appliquées aux architectures résultantes de l'analyses de sécurité (sous-section 1.4).

1.1 Contexte des systèmes embarqués autonomes

Le terme système embarqué désigne avant tout un système, dont la définition est donnée par [1] : “un système est un ensemble complexe d’éléments en interaction dynamique, organisé pour atteindre un but”. Un système embarqué est donc généralement assimilé à un ensemble de composants électroniques piloté par un logiciel, dont le but est de contrôler un autre système qu’il est censé intégrer. Le recours aux avancées technologiques numériques se justifie principalement par leur impact minimal sur le système englobant pour respecter des contraintes strictes fonctionnelles et non fonctionnelles, relatives par exemple à l’encombrement spatial ou bien à la consommation en énergie électrique limitée. Le système embarqué a également pour objectif de répondre aux besoins de rendre plus intelligents et plus communiquant des engins ou des objets courants. Ainsi, un opérateur humain peut interagir plus aisément avec ces engins et partager l’autorité sur des décisions à prendre. On parle alors d’autonomie du système.

L’autonomie d’un système est définie comme étant l’aptitude de celui-ci à prendre seul des décisions à partir d’observations pertinentes et indépendantes. Cette autonomie est parfois requise dans le cas de systèmes ayant des missions complexes et compliqués avec des communications réduites avec un opérateur humain [2]. Par exemple, un satellite en orbite basse peut être équipé d’un contrôle d’orbite autonome pour réduire les opérations au sol. Ainsi, tout engin “mobile autonome dispose de capacités significatives d’adaptation à la mission, d’autonomie décisionnelle et de mobilité”. Par ailleurs, il existe plusieurs niveaux d’autonomie décisionnelle liés à la mobilité de l’engin considéré [3]. Par exemple, le niveau 0 d’autonomie peut être assimilé à une absence d’autonomie et il se manifeste par un contrôle du système complètement réalisé par un opérateur qui perçoit les informations de l’environnement et réagit en conséquence en commandant directement les actionneurs du système. On parle ensuite de niveau 1 lorsque l’on établit une boucle bas niveau de perception-action permettant la régulation et la stabilisation par des actions réflexes qui maintiennent le système dans un fonctionnement sûr. Les niveaux 2-3 peuvent être alors associés à une boucle de contrôle de plus haut niveau comprenant une prise de décision issue du suivi de situation et de l’estimation de l’état du système. Cependant, dans chacun de ces niveaux l’opérateur peut toujours intervenir et reprendre la main comme une autonomie absolue n’est pour l’instant pas envisageable ni souhaitable.

En fait, l’autonomie du système est souvent obtenue par la coopération de fonctionnalités diverses, principalement :

- la perception (mesures de capteurs, informations de suivi de situation),
- la conduite et le suivi de situation nominale (actions pour la gestion de la charge utile et le contrôle de l’engin),

– la décision (gestion des opérations, replanification, supervision générale).

Ces fonctionnalités sont usuellement développées séparément et doivent coopérer pour répondre aux spécifications de comportement sûr du système. Elles contribuent donc à la SdF du système.

En bref La problématique des systèmes embarqués dits "de contrôle" consiste à doter un engin ou un objet d'une autonomie décisionnelle en préservant la nature et la finalité de l'engin ou de l'objet en question. Les systèmes embarqués doivent être conceptuellement suffisamment sûrs pour ne pas introduire de nouvelles défaillances, mais également ils doivent assurer, au tant que faire se peut, la protection ou la mitigation de défaillances existantes.

1.2 Concepts et méthodes en SdF

1.2.1 Dynamique du dysfonctionnement

Tout système embarqué est susceptible d'être confronté, lors de sa phase opérationnelle, à des dysfonctionnements traduisant une non-conformité avec sa spécification ou une spécification impropre et inappropriée [4]. Le dysfonctionnement d'un système fait l'objet d'une étude approfondie en SdF en rapport avec ses conditions d'apparition ou d'occurrence, sa dynamique et ses conséquences internes et externes. Contrairement à l'idée suggérée par cette appellation, la sûreté de fonctionnement (SdF) s'intéresse essentiellement au comportement pouvant mener au dysfonctionnement d'un système autour de 3 notions de base que sont la faute, l'erreur et la défaillance. Les définitions des concepts de SdF qui font référence en la matière ont été formulées dans [5] [6] [7].

Définition 1.1 : Défaillance

Une défaillance correspond à une cessation de l'aptitude d'une entité à accomplir une ou plusieurs fonctions requises. ■

Définition 1.2 : Erreur

Le comportement d'un système étant caractérisé par une séquence d'états de ses composants, une erreur est définie comme un ou plusieurs états non désirés qui peuvent engendrer des défaillances. ■

Définition 1.3 : Faute

La cause d'une erreur, c'est une faute. Une faute est souvent définie par un événement déclencheur d'une erreur. Une faute représente une déviation non acceptable d'au moins une propriété caractéristique ou d'un paramètre du système. ■

Après une défaillance, on considère que le système est en panne. Dans [6], une panne est définie par l'inaptitude d'une entité à accomplir une fonction requise. Une panne est la conséquence d'une faute qui est donc toujours associée à une défaillance.

Pour appréhender le dysfonctionnement d'un système, il convient de définir son fonctionnement ou sa fonction. L'étymologie du mot "fonction" est rattachée à "faire". Donc, la fonction d'un système signifie "ce que doit faire le système", "ce pour quoi le système existe", parfois on emploie le terme de "mission d'un système". La fonction d'un système se traduit alors par des services attendus et rendus à d'autres systèmes. Par une approche systémique, on décrit généralement un système de façon récursive par un ensemble de composants eux-mêmes définis comme des systèmes. Les interactions entre les composants sont associées aux services mutuellement échangés. Ainsi, une faute impactant un composant se manifeste par une erreur propre à ce composant, qui peut engendrer de nouvelles erreurs internes par propagation. Si l'accumulation de ces erreurs perturbe un service rendu par le composant incriminé, on identifie alors une défaillance du composant. A son tour, cette défaillance peut devenir la cause d'une erreur engendrée dans un autre composant. Ce phénomène de propagation peut se poursuivre jusqu'à endommager le (ou les) service(s) rendu(s) par le système tout entier, et on conclut ainsi à la défaillance du système.

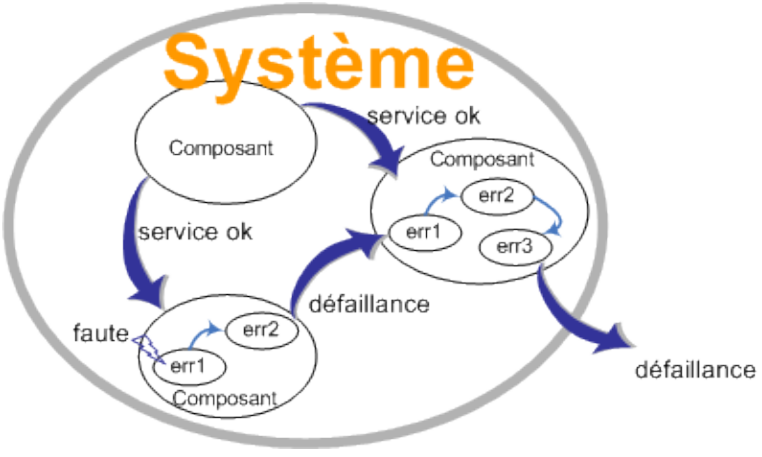

FIG. 1.1 – Propagation d'un dysfonctionnement.

Un exemple simple de la vie courante qui illustre bien ces notions consiste à considérer une voiture. Un défaut de démarrage constitue une défaillance

du moteur, car la fonction du moteur qui permet de traduire une sollicitation du conducteur en mouvements mécaniques n'est pas réalisée. Cette même défaillance empêchant l'utilisation du véhicule pour un déplacement est également une défaillance du véhicule. En regardant de plus près dans le moteur, on peut s'apercevoir par une étude plus poussée que la quantité d'air admise dans les cylindres est insuffisante pour un démarrage du moteur, donc erronée. Et cette erreur peut être causée par une faute de maintenance due au non remplacement d'une pièce du circuit d'admission d'air, ou bien elle peut être due à une faute externe au circuit d'admission d'air, comme une batterie déchargée.

Dans la section suivante, on cherche à décrire des attributs des fautes, des erreurs et des défaillances pertinents pour la suite de notre étude. Le lecteur trouvera une taxonomie détaillée de ces notions dans [4].

1.2.2 Typologie

Les fautes sont des événements non désirés et imprévisibles que l'on distingue par des caractéristiques relatives à trois points de vue. La première caractéristique concerne l'aspect temporel des fautes désigné par deux valeurs distinctes : *permanente* et *temporaire*. Une faute permanente est présente indépendamment de conditions internes ou externes, tandis que la présence d'une faute temporaire est limitée. La seconde et la troisième caractéristiques se rapportent au domaine des fautes, à savoir à l'aspect spatial traduisant respectivement la localité et l'origine des fautes. La caractéristique de localité dévoile des fautes *internes* qui appartiennent à des composants du système et qui sont considérées comme "dormantes" tant qu'elles ne sont pas activées par une action. Par opposition, les fautes *externes* sont dues à d'autres systèmes ou à l'environnement physique ou humain du système. Enfin, la caractéristique liée à la structure des fautes distingue des fautes *indépendantes* dont les causes sont distinctes, et des fautes *corrélées* ou de causes communes qui peuvent être activées simultanément par un même événement.

Dans le dysfonctionnement d'un système les défaillances étant des événements observables, leurs caractéristiques sont un peu plus nombreuses. De même que pour les fautes, parmi les caractéristiques des défaillances on trouve l'aspect temporel distinguant des défaillances *permanentes*, qui persistent tant qu'il n'y a pas d'opérations de maintenance corrective, et des défaillances *intermittentes*, qui ont une durée limitée sans qu'il y ait besoin d'opérations de maintenance. Puis, on attribue aux défaillances un domaine relatif à la qualité de la fonction réalisée par le système. Dans le cas où cette fonction est délivrée mais est incorrecte, on parle alors d'une défaillance *erronée*. Si cette fonction n'est plus délivrée, on parle dans ce cas d'une défaillance silencieuse (en anglais, *silent fail*). Par ailleurs, une autre caractéristique traduit la structure des défaillances, leur indépendance (défaillances *indépendantes*, sans lien les unes avec les autres) ou leur corrélation (défaillances *corrélées* ou de causes communes). La dernière

caractéristique considérée ici se rapporte à la perception de la défaillance par les différents utilisateurs du service. La défaillance est dite *cohérente* dans le cas où tous les utilisateurs perçoivent la défaillance de la même manière. Dans le cas contraire, elle est dite *incohérente* ou est nommée défaillance byzantine (l'origine de ce nom est expliqué plus loin) lorsque certains utilisateurs perçoivent un service correct alors que d'autres suggèrent une défaillance. Une configuration de l'ensemble de ces caractéristiques constitue ce qu'on appelle un *mode de défaillance*. Le mode de défaillance identifie la manière dont le système défaille. En général, on fixe la plupart des caractéristiques et on distingue les défaillances par leur domaine. Ainsi, dans notre étude, on fait les hypothèses de défaillances permanentes, indépendantes et cohérente, ce qui implique des modes de défaillance soit *erroné* soit *perdu* (pour des défaillances *silencieuses*).

De la même manière, les erreurs peuvent être caractérisées par les modes de défaillances qu'elles engendrent :

- mode *erroné* : les données en sortie du traitement dans le composant sont incorrectes ;
- mode *perdu* : il n'y a plus de donnée en sortie du traitement.

Les tableaux 1.1 donne une synthèse des différentes caractéristiques des fautes (tableau 1.1-a), des erreurs (tableau 1.1-b) et des défaillances (tableau 1.1-c).

Faute
– temporel : {permanente, temporaire}
– domaine : {interne, externe}
– structure : {indépendante, corrélée}

(a)

Erreur
– mode : {erroné, perdu}

(b)

Défaillance
– temporel : {permanente, intermittente}
– domaine : {erronée, silencieuse}
– structure : {indépendante, corrélée}
– cohérence : {cohérente, byzantine}

(c)

TABLEAU 1.1 – Synthèse des caractéristiques des fautes (a), erreurs (b), et des défaillances (c).

Des illustrations de ces caractéristiques sont présentes dans la littérature scientifique ou bien sont apportées par des faits réels. Par exemple, le problème des généraux byzantins (Byzantine Generals Problem, BGP) est un problème de référence symbolisant une défaillance incohérente du point de vue des utilisateurs.

Ce problème est formulé de la manière suivante : des divisions de l'armée byzantine, commandées chacune par un général, assiègent une ville ennemie. Parmi ces généraux, certains sont des traîtres. Les généraux communiquent entre eux par envoi de messagers. L'objet du problème est alors de faire en sorte que tous les généraux loyaux se mettent d'accord sur un plan de bataille. Les généraux loyaux recevant les mêmes informations, le problème se ramène à l'envoi d'une valeur par un général à plusieurs lieutenants chargés de la transmettre aux autres généraux. Cependant, il peut y avoir des traîtres parmi les lieutenants. Les conditions de cohérence interactive à respecter pour résoudre ce problème garantissent un accord sur le résultat des échanges :

– *Tous les lieutenants loyaux obéissent au même ordre.*
– *Si un général est loyal, alors chaque lieutenant loyal obéit à son ordre.*

Une résolution de ce problème a été apportée sous forme algorithmique par [8].

Une autre illustration de fautes externes et d'un mode de défaillance *perdu* est couramment mentionnée au sujet du vol 501 d'Ariane 5 le 04 juin 1996 : après 37 secondes de vol, la fusée a explosé suite à une défaillance des deux systèmes inertiels. Les calculateurs inertiels sont tous deux tombés en panne à cause d'un dépassement de capacité d'une variable qui a généré une exception logicielle et l'arrêt des calculateurs après envoi de valeurs erronées au calculateur principale. A propos de cette catastrophe, un rapport de recherche [9] indique :

> "Les véritables causes de la défaillance sont des fautes de capture des besoins applicatifs et des hypothèses d'environnement relatifs à Ariane 5, ainsi que des fautes de conception et de dimensionnement du système informatique embarqué à bord d'Ariane 5."

Maintenant, voyons quelles sont les qualités propres à une expertise du fonctionnement du système.

1.2.3 Propriétés de SdF

Après avoir défini les trois notions qui constituent des entraves au fonctionnement correct, on va maintenant s'intéresser aux propriétés du fonctionnement sûr i.e. le fonctionnement de qualité pendant des utilisations répétées du système dans des conditions données.

La propriété de fiabilité est sans doute la propriété la plus recherchée puisqu'elle concerne l'aptitude d'un système à fournir de façon continue un service correct. En général, elle est mesurée par la probabilité de ne pas tomber en panne jusqu'à une date t. Elle permet également d'estimer quantitativement le temps moyen de fonctionnement correct avant la défaillance (en anglais, *Mean Time To Failure* ou *MTTF*).

De même, la disponibilité caractérise la condition du système prêt à rendre un service correct. Cet attribut de la SdF est généralement mesuré par la probabilité de fonctionner correctement à l'instant t. Dans le cas d'un système

réparable, il permet d'exprimer la proportion de temps où le service est correct avant défaillance par rapport au temps moyen entre deux défaillances : $disponibilité = A = MTTF/(MTTF + MTTR) = MTTF/MTBF$, avec MTTR signifiant temps moyen de réparation (en anglais, *Mean Time To Repair*) et MTBF temps moyen entre deux défaillances (en anglais, *Mean Time Between Failure*).

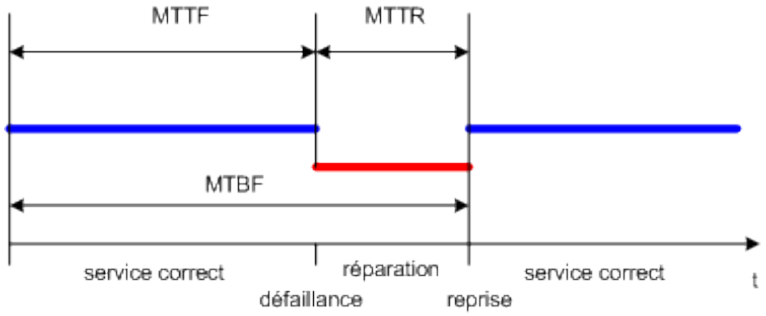

FIG. 1.2 – Estimation de la fiabilité et de la disponibilité.

Enfin, la sécurité concerne la capacité du système à empêcher ou au moins à limiter l'occurrence d'un événement redouté pour lui-même et pour son environnement en cours de fonctionnement. Dans l'aéronautique par exemple, elle est évaluée quantitativement par le taux d'occurrence d'un événement redouté par unité de temps de fonctionnement.

Concernant l'évaluation quantitative de sécurité, un exemple tiré de [10] met en évidence l'importance d'une analyse rigoureuse pour le calcul du taux d'occurrence d'un événement redouté : une situation redoutée provient de la panne triple de 3 composants indépendants ayant chacun un taux de défaillance horaire de $\lambda = 0,5.10^{-3}$. Un raisonnement simple conduit à considérer cet événement comme étant extrêmement improbable avec une probabilité inférieure à 10^{-9} et à conclure à l'acceptation du risque. Cependant, un calcul détaillé prenant en compte :

- le temps d'exposition au risque qui est de $T_0 = 3$ *heures* pour une mission,
- et le temps entre les contrôles de maintenance qui correspond à un contrôle tous les $N = 100$ *missions*,

conduit à une probabilité horaire de la panne triple égale à $N^2T_0^2\lambda^3 = 100^2 \times 9 \times 0,125.10^{-9} = 1,12.10^{-5}$.

Chacune de ces propriétés quantitatives fait l'objet de nombreuses études dans des secteurs d'applications divers comme l'aéronautique ou le nucléaire, par exemple. La fiabilité et la disponibilité sont largement traitées dans [6]. Nous encourageons donc le lecteur à se référer à cet ouvrage pour plus de détails. Il existe aussi diverses propriétés de SdF qualitatives, non probabilistes. On trouve ainsi les propriétés de tolérance à un nombre fixé de fautes. Typiquement, dans l'aéronautique, on doit tolérer qu'une unique faute ne doive pas conduire à un

événement catastrophique. Les exigences fonctionnelles que doivent satisfaire les mécanismes de sécurité constituent une autre grande classe de propriétés de sécurité qualitatives.

Par la suite, nous traiterons uniquement des propriétés de SdF qualitatives, qui, dans l'aéronautique, sont les seules applicables aux logiciels.

1.3 Analyse de la sécurité

1.3.1 Processus d'analyse

Dans la pratique, le processus d'analyse de sécurité repose sur une analyse préalable minutieuse et rigoureuse des fonctions du système. Cette analyse fonctionnelle spécifie le comportement idéal (sans faute) attendu. Sachant que la perfection n'est pas de ce monde, on introduit très tôt l'éventualité de fautes et de possibles imperfections dans la spécification fonctionnelle. Outre l'intérêt économique, cette démarche d'analyse permet d'améliorer ostensiblement la faisabilité et la viabilité du projet, puis du système. Alors, comment se déroule cette analyse ? Suivant les applications industrielles considérées, des standards sont élaborés pour guider le concepteur de systèmes critiques.

Les normes ARP (Aerospace Recommended Practice) 4754A [11] et 4761 [12] en vigueur dans l'aéronautique décrivent les processus à suivre afin d'accroître la confiance accordée au système développé. Une synthèse rapide de ces normes donne un aperçu d'une analyse de sécurité fiable menée parallèlement au processus de conception traditionnelle d'un système complexe dont elle suit la progressivité. D'abord, l'analyse de sécurité débute par une reconnaissance des risques fonctionnels par une analyse dite des risques fonctionnels ou en anglais Functional Hazardous Analysis (FHA). La FHA assure une identification des défaillances potentiellement associées à chaque fonction du système, dans des conditions d'opération particulières. De plus, la FHA classe ces conditions de défaillances en fonction de leur gravité pour l'avion ou ses passagers/équipage. La classification des conditions de défaillances est codifiée en aéronautique conformément au tableau 1.2. De cette classification, on en déduit systématiquement les exigences de sécurité applicables au système. Considérons, par exemple, le système de commande de vol électrique d'un avion qui dispose d'une fonction principale d'actionnement des gouvernes en fonction de consignes reçues. Le service concerné peut alors être altéré de plusieurs manières : arrêt de l'envoi des commandes aux gouvernes, envoi de commandes erronées, temps de réponse inapproprié aux consignes (trop rapide ou trop lent), qui n'auront pas les mêmes répercussions selon les conditions d'opération de l'avion. Par exemple, la perte du freinage des roues de l'avion lors de l'atterrissage sera plus critique si la piste est mouillée. Ces défaillances vont donc avoir des effets sur la capacité de l'avion à être piloté. Ainsi, on déduit de la FHA la criticité des situations engendrées

Taux de panne (Heure de vol)	$< 10^{-3}$	$< 10^{-5}$	$< 10^{-6}$	$< 10^{-7}$	$< 10^{-9}$
Probabilité	fréquente	probable	peu probable	très peu probable	extrêmement improbable
Description		plusieurs fois dans la vie d'un avion	plusieurs fois dans l'ensemble de la flotte	sans doute jamais dans la vie de la flotte	ne doit jamais se produire
Effet de la panne	mineur		majeur	critique	catastrophique
Marge sécurité	diminution		réduction significative	réduction importante	perte avion
Charge travail	augmentation		diminution efficacité équipage	l'équipage ne peut assurer toutes ses tâches	l'équipage ne peut assurer ses tâches
Passagers	inconfort		blessures	blessures graves ou pertes limitées	pertes importantes
Criticité de la fonction	non essentielle		essentielle		critique
Niveau d'assurance développement	D		C	B	A
Niveau logiciel	3		2B	2A	1

TABLEAU 1.2 – Classification des conditions de défaillances.

par ces défaillances dans des scénarios opérationnels. Ces situations sont communément appelés conditions des défaillances (en anglais, Failure Condition ou FC).

L'étape suivante est cruciale car elle détermine le comportement sûr attendu du système. Elle porte sur une description préliminaire de l'architecture du système (définissant l'organisation des fonctions et les relations entre ces fonctions), et elle correspond à l'évaluation préliminaire de sécurité (PSSA : Preliminary System Safety Assessment) réalisée en général à l'aide d'outils analytiques spécifiques (voir section 1.3.2). Il en sort de cette étape une meilleure compréhension du comportement nominal ou fautif et des actions d'amélioration du système.

Lors de l'étape suivante (SSA : System Safety Assessment), l'évaluation de sécurité est cette fois-ci appliquée à une architecture physique constituant une solution à l'architecture préliminaire élaborée dans l'étape précédente. Bien entendu, comme tout processus ayant trait aux systèmes complexes, ces étapes ne se déroulent pas en une fois mais nécessitent de nombreuses itérations.

Un point important à noter dans ce processus d'analyse de sécurité c'est l'obligation d'avoir une spécification rigoureuse et minutieuse de l'architecture fonctionnelle du système. En effet, toute l'analyse repose sur cette architecture, véritable clef de voûte du système sur lequel on s'appuie afin de conforter notre confiance dans la sécurité affichée. De plus, cette architecture peut être particulièrement complexe pour les systèmes à logiciels prépondérants étudiés dans cette thèse. C'est la raison pour laquelle notre étude s'est orientée vers une spécification formelle d'une architecture fonctionnelle particulière et la re-

cherche de propriétés qualitatives caractérisant cette architecture. Nous avons choisi pour cela une architecture suffisamment générique (architecture que nous présenterons dans le chapitre suivant) et habituellement appliquée aux systèmes autonomes mobiles.

1.3.2 Méthodes d'évaluation

On distingue plusieurs méthodes d'évaluation de la sécurité. L'analyse des modes de défaillances, de leurs effets et de leur criticité (AMDEC) est une méthode de type inductif qui s'utilise aisément lors d'une analyse préliminaire de risques. L'AMDEC a été initialement développée par l'armée américaine à la fin des années quarante sous la référence MIL-P-1629. Cette méthode informelle repose sur l'établissement d'un tableau décrivant les modes de défaillances des éléments du système étudié, leurs effets et leur criticité, éventuellement complétés par les mécanismes de détection et de reconfiguration ad hoc. Il existe plusieurs types d'AMDEC dont l'AMDEC fonctionnelle qui permet d'identifier les conséquences d'une simple faute. Les étapes de sa construction intègrent : l'association d'un mode de défaillance à chaque fonction du système, l'identification de scénarios de panne (ou conditions de panne) dans un contexte opérationnel et la détermination des conséquences des modes de défaillance. Des probabilités d'occurrence de panne peuvent être affectées aux éléments fonctionnels pour des traitements postérieurs par arbre de défaillances par exemple.

La méthode par arbre de défaillances (AdD) est une des méthodes les plus répandues pour les études de SdF [6]. Cette méthode d'analyse de type déductif est simple d'utilisation : la structure arborescente permet d'identifier les causes multiples d'un événement redouté placé au sommet. La construction descendante d'un AdD consiste à décomposer un événement redouté indésirable en combinant à l'aide d'opérateurs logiques des événements intermédiaires constituant les causes de l'événement de départ. Chacun de ses événements intermédiaires est à son tour décomposé jusqu'à parvenir à un niveau d'événements élémentaires indépendants dont la décomposition est jugée inutile ou impossible. L'exploitation qualitative des AdD vise alors à expliciter l'ensemble des combinaisons d'événements élémentaires, appelées coupes, conduisant à l'événement redouté à partir des règles de l'algèbre de BOOLE. De plus, l'association d'une probabilité d'occurrence aux événements élémentaires assure une exploitation plus quantitative des AdD en se basant sur la théorie des probabilités. Par principe, l'analyse par AdD s'applique à l'aspect statique des systèmes pour lequel l'ordre d'occurrence des événements n'a pas d'impact sur les résultats de l'analyse. Or de nombreuses études de systèmes physiques nécessitent la prise en compte des aspects temporels au niveau des événements et des fonctions associées [13]. Les travaux de Dugan et de son équipe de l'université de Virginie aux États-Unis [14] ont néanmoins permis de répondre à ce problème en intégrant la dimension dynamique dans les AdD à l'aide des chaînes de Mar-

Phase	**vol à vue en zone non habitée**
Mode de panne	Perte du contrôle manuel en zone non habitée
Cause possible	Calculateur KO (1), liaison bord-sol KO (2), changement de mode involontaire par le pilote (3)
Effet local	Actionneurs bloqués dans la dernière position commandée
Effet sur système	Maintien du drone sur la dernière position stable ; (4) mouvement du drone en automatique
Détection	– (1), (2), (3) Pas de réaction du drone aux ordres du pilote (observateur extérieur) ; – (1) Commande erronée en sortie calculateur ; – (3) Données de la liaison non rafraîchies
Action remède	– (1) Basculer sur le redondant (s'il existe) ; – (3) Basculer en mode automatique ; – (2) Basculer dans un mode sûr automatique minimal avec retour au point de départ ou autre point de repli sûr préenregistré dans le fichier mission pour un atterrissage automatique

TABLEAU 1.3 – Exemple d'un tableau AMDEC pour le système de contrôle d'un drone.

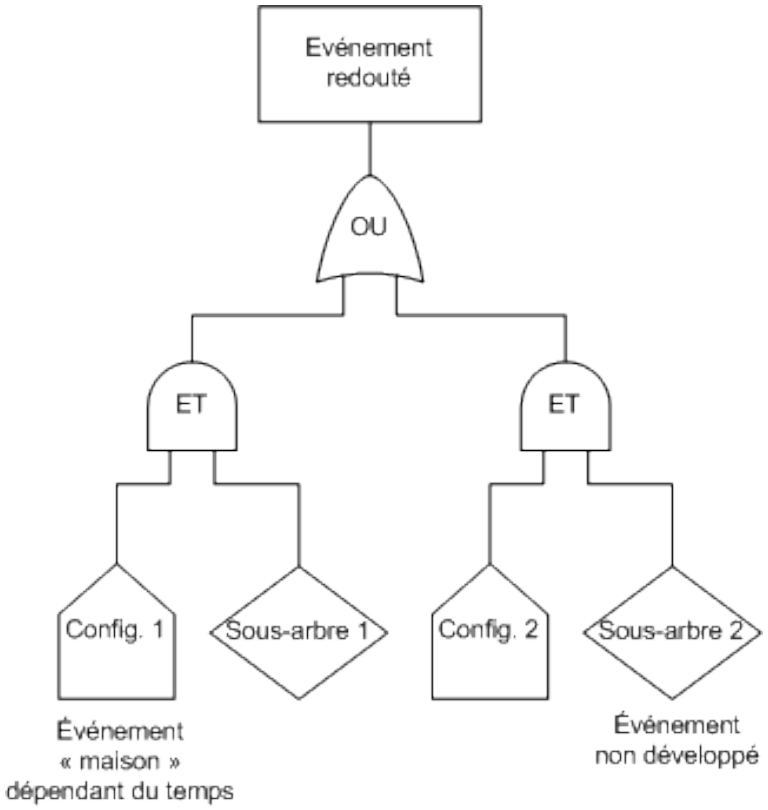

FIG. 1.3 – Arbre de défaillance dynamique de Cepin et Mavko.

khov. C'est également l'approche adoptée par [15] avec le nouveau formalisme BDMP (Boolean logic Driven Markov Processes" ®) qui combine la puissance des graphes de Markov (moins l'explosion combinatoire) à la simplicité de la modélisation par arbres de défaillances. Les travaux de Cepin et Mavko [16] offrent quant à eux une alternative en proposant un arbre dit dynamique basé sur la "composition" de sous arbres statiques correspondant à des configurations du système à différents instants donnés (figure 1.3).

Les méthodes heuristiques et probabilistes telles que la simulation de Monte Carlo et les modèles de Markhov sont également très répandues pour l'analyse de sécurité des systèmes dynamiques. Ces techniques sont très pratiques dans l'évaluation quantitative pour la fiabilité et la disponibilité d'une fonction ou d'un système.

1.4 Processus de conception

L'analyse de sécurité repose sur un processus et des méthodes d'évaluation qui permettent de comprendre comment les fautes apparaissent, se propagent et doivent être tolérées pour tenir les objectifs de sécurité applicables à un système. Elle accompagne un processus de conception qui doit lui-même être sûr

afin de limiter les défauts de conception. Ainsi pour les systèmes embarqués, une approche descendante ("top-down", en anglais) synthétique du processus de conception conduit à spécifier des exigences principales dites de "haut-niveau" caractérisant la finalité, la mission et les objectifs du système. D'abord, ces exigences décrivent l'aspect *"besoins-attentes"* qui permet d'expliciter les services et les contraintes, exprimés sous forme de fonctions.

Ensuite, les fonctions sont soit allouées à des sous-systèmes lorsqu'ils existent, soit décomposées, et au final on obtient une architecture fonctionnelle permettant de caractériser le fonctionnement interne du système, puis une architecture organique en constituants. Dans le cadre de systèmes embarqués, cet aspect *"architecture"* permet de décomposer les exigences de haut niveau en exigences du logiciel et en exigences du matériel.

Après le développement des constituants, l'étape d'intégration constitue une longue phase de V & V du système.

Cependant, les activités de V & V sont également menées tout au long du processus de conception pour :

- vérifier la conformité par rapport aux spécifications fonctionnelles nominales et par rapport aux hypothèses de comportements défaillants ;
- valider les solutions envisagées sous forme d'architectures.

Leur nature dépend des objets analysés. Ainsi, en phase amont, des spécifications rédigées en langage naturel sont validées par relecture. Lorsqu'un prototype de système existe, des tests permettent de le valider. Les spécifications formelles constituent une classe particulière d'objets analysables par simulation ou analyse statique : modèles formels pour des systèmes dynamiques en Lustre [17], ou l'aide de réseaux de Petri [18]. Event-B est l'outil retenu pour notre étude. En effet il est employé pour spécifier formellement un comportement et valider des propriétés intrinsèques à des systèmes complexes. Fondé sur la logique des prédicats du premier ordre et la théorie des ensembles, Event-B permet de modéliser et de vérifier des architectures définies par des propriétés génériques (comme l'existence de redondances d'un certain type) et pas uniquement une instance particulière d'architectures. Une description détaillée des modèles et de l'outil Event-B sera présentée dans le chapitre 3. De plus, son usage suppose un certain nombre d'hypothèses à établir pour une représentation du système réel, "juste nécessaire". Ainsi, il revient au concepteur de vérifier la cohérence et la correction de la modélisation, en présence de ces hypothèses.

En bref La SdF traduit un besoin de confiance sur les systèmes développés. Cette confiance est apportée par de nombreuses méthodes et pratiques en cours qui associent des formalismes mathématiques à des connaissances a priori du comportement attendu. Des architectures dédiées à la SdF sont alors développées pour anticiper la gestion des défaillances auxquelles tout système artificiel est sujet.

Chapitre 2

La conception des architectures de SdF

Résumé

Ce chapitre présente les mécanismes de sécurité et les patrons d'architectures qui enrichissent l'architecture fonctionnelle nominale du système pour rendre ce dernier plus sûr. En effet, il convient de bien définir l'architecture complète du système ainsi que ses propriétés pour garantir la pertinence des mécanismes de SdF mis en place. C'est d'ailleurs dans cette optique que l'on se propose d'étudier une architecture en couches fonctionnelles adaptées à la représentation des systèmes autonomes de contrôle.

« *L'architecture est art de suggestion.* »
Daniel Pennac - La Petite Marchande de prose

En ingénierie des systèmes [19], la résolution d'un problème passe par une analyse fonctionnelle qui permet d'identifier les fonctions appropriées. C'est la phase de décomposition du problème. Puis, cette analyse conduit nécessairement à élaborer une architecture qui traduit les relations entre ces fonctions. L'avantage de cette démarche, inscrite dans le cycle en V de développement d'un système, est de pouvoir s'assurer à chaque étape de la pertinence des choix.

De même, en SdF, il existe des fonctions spécifiques de tolérance aux fautes (section 2.1) que sont la détection, l'isolation, l'identification et la reconfiguration. Dans ce chapitre, un aperçu de ces diagnostics est proposé dans le cas d'une approche stochastique (sous-section 2.2.1), d'une approche continue (sous-section 2.2.2) et d'une approche discrète (sous-section 2.2.3). Il est certain que la présentation de la tolérance aux fautes en variant les approches met en évidence des architectures fonctionnelles particulières que l'on ne détaillera pas. Néanmoins, cela apporte des pistes de développement des mécanismes de sécurité encore plus abstraits considérés par la suite dans notre étude.

Par ailleurs, l'analyse du système impose également de définir une architecture physique des éléments constitutifs. Le passage de l'architecture fonctionnelle à l'architecture physique est fortement facilité par l'usage régulier de patrons d'architectures éprouvées et validées (sous-section 2.1.1). Parmi ces architectures éprouvées, on a fait le choix d'une architecture en couches fonctionnelles plus générique et à échelle plus important pour représenter un système autonome de contrôle (sous-section 2.3.1). On identifie également des propriétés (sous-section 2.3.2) permettant de caractériser cette architecture adoptée que l'on instancie dans le cas de l'étude d'un drone capable autonomie (sous-section 2.3.4).

2.1 Principes des architectures de SdF

Par définition, l'architecture est "l'organisation de divers éléments constitutifs d'un système, en vue d'optimiser la conception de l'ensemble pour un usage déterminé". En prenant en compte uniquement les aspects de SdF, l'architecture globale se réduit à quelques composants qui mettent en œuvre des mécanismes destinés au respect des propriétés de SdF. Mais de quels mécanismes a-t-on besoin pour garantir la sécurité d'un système ? Comme on ne peut empêcher une faute imprévisible d'avoir lieu, il est indispensable d'établir des fonctions permettant de mitiger la propagation des défaillances. Pour cela, certaines études portent sur la recherche des conditions d'occurrence des fautes pour tenter de les éliminer tout en sachant que l'exhaustivité des résultats n'est pas réaliste pour des systèmes embarqués critiques fonctionnant en permanence. Aussi ces études sont-elles complétées par une conception d'architectures permettant de

tolérer des fautes. La structure de ces architectures comprend des redondances et des composants spécifiques permettant de réaliser les trois fonctionnalités suivantes :

- détection : observer une incohérence dans le comportement d'un composant ;
- identification (et isolation) : déterminer la cause de la défaillance et la confiner pour éviter sa propagation ;
- reconfiguration : réaliser des actions correctrices améliorant le comportement (voire retour à un comportement sans faute).

Ces fonctionnalités forment le mécanisme communément appelé détection de fautes, identification / isolation et reconfiguration (en anglais, Fault Detection Isolation and Reconfiguration, FDIR). En pratique, les deux premières fonctionnalités, la détection de fautes et l'identification, constituent le principe du diagnostic de pannes, la reconfiguration étant souvent considérée séparément.

Dans la section suivante, nous verrons quelques motifs génériques mettant en œuvre des mécanismes de reconfiguration. Les motifs de sécurité offrent une méthode simple de conception d'architectures physiques.

Mais avant cela, pour comprendre l'intérêt d'une architecture spécifique de tolérance à une défaillance, revenons à notre problème de défaillance byzantine et détaillons la solution algorithmique proposée qui impose une architecture physique particulière [8]. On rappelle que l'objet du problème est de faire en sorte que tous les généraux loyaux se mettent d'accord sur un plan de bataille, que parmi ces généraux, certains sont des traîtres, et que les généraux communiquent entre eux par envoi de messagers. On démontre que l'envoi d'un message oral n'est fiable seulement si le nombre total n de généraux est supérieur à trois fois le nombre m de généraux traitres. On suppose que les généraux utilisent la même méthode de combinaison des informations pour prendre une décision. La décision est définie par une fonction de majorité sur les valeurs $v(1), \ldots, v(n)$. Un algorithme performant $OM(m)$ pour résoudre ce problème dans le cas de l'envoi de messages oraux, avec au plus m traîtres, est décrit sous forme récursive (algorithme 2.1 et algorithme 2.2).

Algorithme 2.1 Algorithme OM(0)

1: Le général envoie sa valeur à chaque lieutenant.
2: Chaque lieutenant transmet la même décision tirée de la valeur reçue.

Une illustration de cet algorithme est représentée dans la figure 2.1 avec un général et six lieutenants. Le général et le lieutenant $L6$ sont des traîtres. Les messages envoyées sont "A" pour *Attaque*, "R" pour *Retraite* et "x" indiquant l'un ou l'autre des messages précédents. D'après l'algorithme 2.2, le lieutenant $L1$ reçoit :

OM(2) : A

OM(1) : 2R 3A 4R 5A 6R

Algorithme 2.2 Algorithme OM(m)

Input: $m \geq 1$

1: Le général envoie une valeur à chaque lieutenant.
2: Soit v_i la valeur reçue par le lieutenant i. Chaque lieutenant i agit comme un général dans l'algorithme $OM(m-1)$ et envoie la valeur v_i à chacun des $n-2$ autres lieutenants.
3: Pour chaque i et chaque $j \neq i$, soit v_j la valeur reçue par le lieutenant i de la part du lieutenant j dans l'étape 2 (en utilisant l'algorithme $OM(m-1)$). Le lieutenant i transmet la décision tirée de la valeur $majorite(v_1, \ldots, v_{n-1})$.

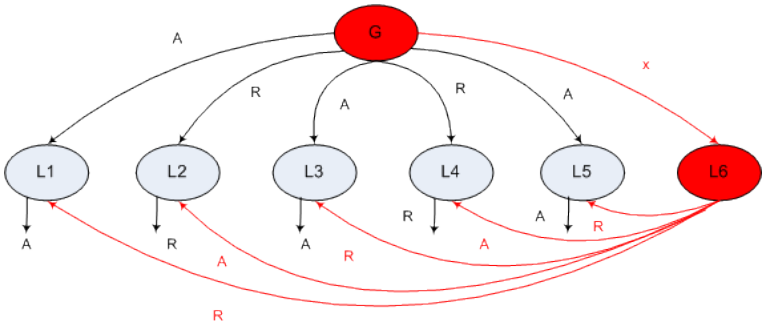

FIG. 2.1 – Exemple de scénario n=7, m=2.

OM(0) :	$2\{1A$		$3A$	$4R$	$5A$	$6A\}$
	$3\{1A$	$2R$		$4R$	$5A$	$6R\}$
	$4\{1A$	$2R$	$3A$		$5A$	$6A\}$
	$5\{1A$	$2R$	$3A$	$4R$		$6R\}$
	$6\{1x$	$2x$	$3x$	$4x$	$5x$	$\}$

Le message reçu par $L1$ pour $OM(0)$, complété par $OM(1)$, est identique pour chaque lieutenant autre que $L6$ (à l'exception de la ligne correspondant aux messages envoyés par $L6$). Ainsi, tous les lieutenants loyaux prennent la même décision, *"Attaque"* issue de $majorite(A, R, A, R, A, -)$ (la valeur "$-$" indiquant une non prise en compte de cette sixième colonne liée à l'incohérence de ces données).

Par analogie avec des réseaux de calculateurs, l'intérêt de ce problème de généraux byzantins réside dans la maîtrise des défaillances de calculateurs. Des calculateurs défectueux risquent de corrompre tout un système en envoyant des informations incohérentes aux autres calculateurs. L'application de l'algorithme de [8] permet alors de déterminer le nombre de calculateurs redondants ainsi que le protocole de communication nécessaires pour rendre le système le plus sûr possible : $3m+1$ calculateurs pour m calculateurs défaillants. Cependant, cet argument sur le nombre de calculateurs exploités peut constituer un frein dans l'utilisation de cet algorithme. Un autre inconvénient est également son coût en temps d'exécution.

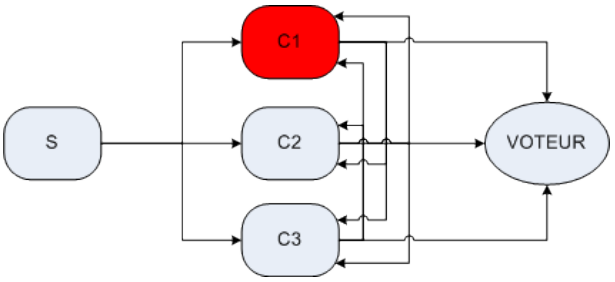

FIG. 2.2 – Exemple de scénario avec un réseau de calculateurs. Les calulateurs C1, C2, C3 sont redondants.

2.1.1 Patrons d'architectures

En conception de systèmes complexes, l'usage est d'utiliser des solutions génériques simples et éprouvées qui rendent accessibles les architectures à des utilisateurs non experts. Ainsi, les patrons d'architectures visent à favoriser la compréhension et la réutilisation d'architectures récurrentes dans le but de faciliter l'analyse de systèmes tout en réduisant leurs coûts de conception et de développement [20]. Les patrons d'architectures forment une sorte de librairie de modèles se rapportant à des larges domaines d'application.

Dans le cadre l'analyse de SdF, les motifs ou patrons développés s'intéressent aux mécanismes de propagation de défaillances [20]. Les motifs de SdF sont construits à partir des éléments de base indispensables pour une interaction avec le reste de l'architecture : les interfaces entrées/sorties, les fonctions de service et les fonctions de contrôle/test.

Des exemples de motifs concernent principalement la redondance matérielle s'appliquant à des équipements critiques tels que capteurs, calculateurs ou circuits hydrauliques. Deux types de redondance matérielle sont en particulier présentés dans [20] :

- la redondance froide ("cold redundancy") caractérise une architecture contenant un équipement actif et un ou plusieurs équipements redondants inactifs et non alimentés ; en cas de défaillance de l'équipement actif, un dispositif permet de déconnecter ce dernier et de mettre en marche un équipement redondant après une initialisation (inconvénient lié à l'échec d'une initialisation) ;
- la redondance chaude ("hot redundancy") traduit une architecture dans laquelle les équipements redondants sont en permanence actifs et ils reçoivent les mêmes informations que l'équipement primaire ; la reconfiguration liée à une défaillance de l'équipement primaire est plus aisée et plus rapide.

Une forme particulière de redondance est la triplication ou la redondance triple. En général, trois équipements identiques reçoivent des informations similaires et indépendantes et effectuent le même traitement en utilisant des logiciels

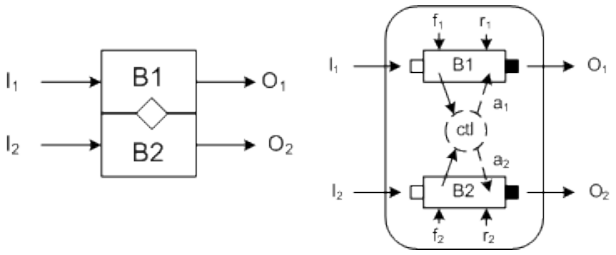

FIG. 2.3 – Motif de redondance.

dissimilaires pour plus de sûreté. Les sorties de ces équipements peuvent être exploitées de différentes façons. Par exemple, elles peuvent servir d'entrées à un voteur qui sélectionne la valeur médiane parmi les trois données produites.

Une spécification plus détaillée du comportement des composants d'un patron d'architecture de sécurité peut être trouvée dans [21]. [21] fournit un cadre de description formelle de la stratégie de reconfiguration basée sur des transitions de modes opérationnels associés aux objectifs (ou fonctions primaires) du système. Plus concrètement, [22] préconise des fonctions particulières chargées de la surveillance et du contrôle, et appelées respectivement *détecteurs* et *correcteurs*. Ces fonctions sont apparentées indifféremment à des éléments logiciels ou matériels :

- des exemples de *détecteurs* : alarmes, watchdogs, codes détecteurs d'erreurs ;
- quelques exemples de *correcteurs* : voteurs, codes correcteurs d'erreurs.

Nous détaillerons dans la section suivante quelques mécanismes de tolérance aux fautes introduits précédemment. De même, des descriptions plus précises de ces mécanismes sont données dans [21].

En bref Les architectures dédiées SdF s'appuient sur des mécanismes de tolérance aux fautes qui mettent en œuvre des fonctionnalités de diagnostic et de reconfiguration. Il existe alors des patrons d'architectures permettant un développement plus aisé d'architectures complexes par composition. Cependant, on doit s'assurer de maintenir les propriétés globales de sécurité lors de cette composition.

2.2 Tolérance aux fautes

Tout système artificiel en fonctionnement permanent, qu'il soit critique ou non essentiel, est susceptible d'être soumis à des défaillances que l'on cherche à

mieux maîtriser et à appréhender au plus tôt. Comme indiqué précédemment, ces défaillances proviennent de fautes connues ou inconnues, mais imprévisibles. Dans la suite, on ne s'intéressera qu'à des fautes connues ayant des caractéristiques énoncées ci-après.

Définition 2.4 : Hypothèses sur les fautes

- Une faute est un événement non désiré, non observable, dont l'occurrence est aléatoire.
- Une faute est active dans le sens où elle produit une erreur.
- Une faute est permanente lorsque les raisons de son occurrence ne sont pas ponctuelles ni limitées dans le temps. Les actions de reconfiguration peuvent corriger les effets de la faute, mais ses causes restent inchangées sans une réparation.
- Les fautes sont indépendantes lorsqu'elles sont attribuées à des causes distinctes.

■

Définition 2.5 : Tolérances aux fautes

La tolérance aux fautes est la capacité du système à fonctionner correctement en présence de fautes. Les fautes sont alors masquées du point de vue du fonctionnement du système. Le masquage peut être total ou partiel. Dans le cas d'un masquage partiel, le système a un fonctionnement dégradé en présence de fautes. Certaines propriétés du système ne sont plus vérifiées, mais la fonction principale du système est préservée. Concernant le masquage total, l'intégrité fonctionnelle du système est maintenue. ■

D'après [23] [22], la tolérance aux fautes est rendue possible en règle général grâce à l'application du mécanisme de FDIR. La reconfiguration agit sur le comportement du système et le fait évoluer afin d'atteindre un état de fonctionnement amélioré et stable par rapport à l'état de dysfonctionnement avec la panne. En ce qui concerne le diagnostic, la technique est étroitement liée à la méthode (classique déterministe ou stochastique) et à l'approche (continue ou discrète) de détection utilisées. Le diagnostic repose sur une connaissance a priori des fautes ou des classes de fautes ainsi que sur une méthode de discrimination de ces fautes dans le cas où plusieurs fautes ont le même effet.

Par conséquent, afin de déterminer si un système remplit correctement ses objectifs, il est nécessaire de surveiller de manière précise son fonctionnement à l'aide de capteurs positionnés stratégiquement. Des techniques de placement ou de sélection de capteurs sont alors utilisées pour déterminer le nombre et la localisation des capteurs à placer sur le système. La fonction de surveillance permet de détecter le passage du système en fonctionnement anormal. Elle récupère les informations issues des capteurs et les transforme en indicateurs de

défaillance à partir d'une référence illustrant le fonctionnement normal ou anormal du système [24].

Dans [24], lorsque ces indicateurs de défaillance révèlent un fonctionnement anormal, ils deviennent des symptômes. Les symptômes traduisent les effets observables des défaillances.

Définition 2.6 : Symptôme

Un symptôme est l'effet ou la conséquence visible d'une défaillance. ■

Le problème du diagnostic est de déterminer pour un système donné, à partir d'une référence et d'un ensemble d'indicateurs fournis par la fonction de surveillance, les fautes étant apparues sur le système. Le diagnostic est défini de la manière suivante.

Définition 2.7 : Diagnostic

Le diagnostic est l'identification de la cause probable de la (ou des) défaillance(s) à l'aide d'un raisonnement logique fondé sur un ensemble d'informations provenant d'une inspection, d'un contrôle ou d'un test. ■

Cette définition résume les deux tâches essentielles du diagnostic : l'observation des symptômes de la défaillance et l'identification de la cause de la défaillance à l'aide d'un raisonnement logique fondé sur des observations du système. Les deux étapes d'une méthode de diagnostic sont donc la localisation et l'identification des fautes sur les équipements responsables d'une ou plusieurs défaillances du système. L'étape de localisation permet d'isoler les équipements en panne, c'est-à-dire dans lesquels une faute est apparue. L'étape d'identification détermine le type de faute apparue. Une fois le type de faute identifié et selon la connaissance disponible sur le système, il est parfois possible de propager les effets d'une faute sur les équipements du système afin de prédire les conséquences de ces défaillances.

2.2.1 Diagnostic probabiliste

Plusieurs algorithmes sont utilisés pour la détection d'une faute. L'algorithme le plus simple et le plus courant consiste à comparer la valeur de sortie d'un dispositif du système à une valeur de référence ou à un seuil. On définit ainsi un domaine de valeurs où le système est dans un état nominal et un domaine de valeurs pour lequel une faute est survenue. Cette technique de détection basée sur la cohérence de valeurs est réalisable soit par une méthode classique déterministe soit par une méthode stochastique [25] [26]. Dans le cas d'un diagnostic probabiliste, on considère n mesures d'un système ou n valeurs empiriques $(x_1, x_2, \ldots, x_n)$ de la variable aléatoire ξ, issue d'une loi de

probabilité P entièrement ou partiellement inconnue. Soit $\delta(x_1, x_2, \ldots, x_n)$ une fonction de ces valeurs, δ est une variable aléatoire que l'on cherche à estimer avec $\widehat{\delta}(x_1, x_2, \ldots, x_n)$ un estimateur de δ. En convenant d'une certaine probabilité de l'estimateur, nous acceptons un risque de probabilité α d'une erreur d'estimation. On peut donc déterminer la valeur vraie de δ dans un intervalle $[\delta - h_1, \delta + h_2]$ (h_1 et h_2 sont des paramètres fixant le domaine des valeurs) avec une probabilité $1 - \alpha$ de $\widehat{\delta}$. Supposons deux hypothèses alternatives H_0 et H_1 qui respectivement considèrent le système en situation nominale et en situation de défaut. α définit alors le seuil de confiance associé aux régions déterminées par ces hypothèses :

$$\begin{aligned} P\{choisir\ H_0 | H_0\ vraie\} &= P\{(\delta - h_1) \leq \overline{x} \leq (\delta + h_2) | \overline{x} = \delta_0\} \\ &= 1 - \alpha \end{aligned} \tag{2.1}$$

Par exemple, H_0 peut désigner une valeur moyenne des mesures $\overline{x} = \delta_0$ et H_1 désigne $\overline{x} \neq \delta_0$. Considérons les densités de probabilité des hypothèses H_0 et H_1 connues et égales respectivement à $p(x|H_0)$ et $p(x|H_1)$ (figure 2.4). Pour des valeurs $x > x_{1-\alpha}$, la probabilité que H_0 soit vraie n'est pas nulle et correspond à la surface α sous la densité $p(x|H_0)$ à droite de la droite $x = x_{1-\alpha}$ (avec $x_{1-\alpha}$ la valeur de la variable aléatoire telle que $P\{\xi < x_{1-\alpha}\} = 1 - \alpha$). Cette probabilité correspond aux cas où l'on décide que H_1 est vraie alors que le fonctionnement réel est correct. Il s'agit alors d'une fausse alarme, notée P_F :

$$P_F = \int_{E_1} p(x|H_0)dx \quad avec\ E_1 =]x_{1-\alpha}, +\infty[\tag{2.2}$$

Par symétrie, la surface β sous la densité $p(x|H_1)$ à gauche de la droite $x = x_{1-\alpha}$ correspond aux cas où l'on décide que H_0 est vraie alors que le fonctionnement réel est défaillant. Il s'agit alors d'une non détection ou d'un manque, notée P_M :

$$P_M = \int_{E_0} p(x|H_1)dx \quad avec\ E_0 =]x_{1-\alpha}, +\infty[\tag{2.3}$$

Hypothèse acceptée / **Décision**	H_0 **vraie**	H_1 **vraie**
H_0 retenue H_1 rejetée	Décision correcte $P = 1 - \alpha$	Défaut manqué $P = P_M = \beta$
H_0 rejetée H_1 retenue	Fausse alarme $P = P_F = \alpha$	Décision correcte $P = P_D = 1 - \beta$

TABLEAU 2.1 – Test d'hypothèses et décision binaire

Définition 2.8 : Test statistique

Le test statistique de choix (ou règle de décision) vise à détecter la modification observée sur les caractéristiques statistiques de la variable aléatoire

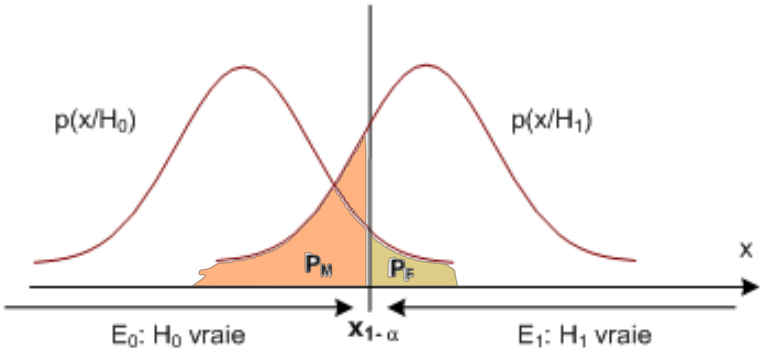

FIG. 2.4 – Exemple de test à deux hypothèses pour une valeur de seuil de confiance fixée.

testée. Il convient alors de minimiser les erreurs de première espèce ou fausses alarmes P_F et les erreurs de seconde espèce ou non détection P_M. ■

Définition 2.9 : Probabilité de détection

La probabilité de détection est liée à l'erreur de seconde espèce par : $P_D = 1 - P_M = 1 - \beta$. ■

Algorithme 2.3 Test d'hypothèses

Input: un ensemble d'échantillons de mesures
Input: un modèle statistique donnant les densités de probabilité de l'espace d'observations
1: définir H_0
2: établir la statistique mesurant l'écart entre les estimations et le résultat attendu sous H_0
3: choisir le seuil de confiance α
4: définir les régions d'acceptation E_0 et critique E_1
5: calculer la valeur de probabilité correspondant à l'expérience
6: conclure à l'acceptation ou au rejet

L'intérêt d'un système de diagnostic probabiliste est double car ce système permet de déterminer si des mesures sont correctes ou non, et il évalue le résultat retourné en fournissant une décision d'acceptation ou de rejet en fonction du principe de fausse alarme et de non détection de fautes. Le schéma synoptique décrit l'architecture de ce système de diagnostic (figure 2.5).

Il existe également d'autres approches de détection reposant sur une représentation continue et sur une représentation discrète du système. L'approche continue est basée sur les techniques utilisées dans le domaine automatique telles que la redondance analytique ou le filtre de Kalman, tandis que l'approche discrète s'intéresse aux techniques logiques utilisées par le domaine informatique.

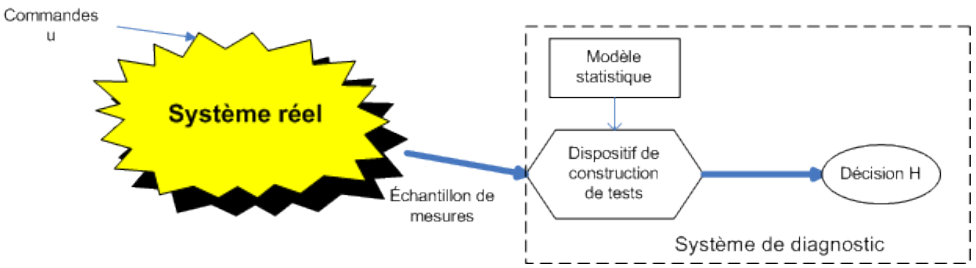

FIG. 2.5 – Système de diagnostic probabiliste.

Une troisième approche hybride permet de combiner ces deux domaines [27] [28] [29] [30].

2.2.2 Diagnostic de systèmes continus

La redondance analytique est une méthode de diagnostic en ligne basée sur la représentation d'états continus du système.

Définition 2.10 : Redondance analytique

La redondance analytique est l'analyse de résidu ou écart entre des grandeurs estimées à partir d'un modèle mathématique du système et des grandeurs réelles (des mesures). ■

Considérons un modèle d'état d'un système linéaire continu pour la synthèse d'un générateur de résidus :

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) + D_x d(t) + F_x f(t) \\ y(t) = Cx(t) \end{cases} \tag{2.4}$$

Dans cette équation, x représente le vecteur d'état, u le vecteur de commande, y le vecteur de sortie. On ajoute sur l'état des entrées perturbatrices : d représentant des incertitudes de modélisation et f représentant des défauts. La représentation par matrices de transfert donne :

$$\begin{aligned} y(s) &= C(sI - A)^{-1}Bu(s) + C(sI - A)^{-1}Dd(s) + C(sI - A)^{-1}Ff(s) \\ &= G_u(s)u(s) + G_d(s)d(s) + G_f(s)f(s) \end{aligned} \tag{2.5}$$

Notion de résidu Un résidu correspond à une différence entre le comportement prédit par le modèle de référence et le comportement observé du système. Il peut se définir de la manière suivante.

Définition 2.11 : Résidu

Un résidu est associé à une différence résultant de la comparaison de mesures de capteurs à des valeurs calculées analytiquement de la variable considé-

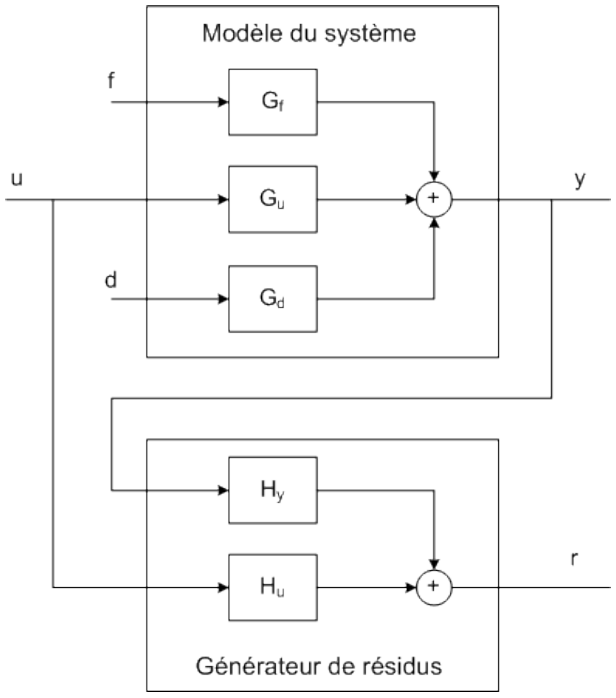

FIG. 2.6 – Dispositif de diagnostic par génération de résidus.

rée dans le système modélisé. ■

On cherche donc à construire un générateur de résidus permettant d'élaborer un vecteur de résidus r à partir de u et de y (figure 2.6). Ainsi dans la figure 2.6, le choix des matrices H_u et H_y permet de définir un résidu répondant aux propriétés de diagnostic des défauts :

$$\begin{cases} r(t) = 0 & si\ f(t) = 0 \\ r(t) \neq 0 & si\ f(t) \neq 0 \end{cases} \tag{2.6}$$

En pratique, les résidus sont des signaux qui, à l'image des mesures, peuvent être considérés soit de manière déterministe en appliquant des techniques d'automatique comme les approches par espace de parité ou à base d'observateurs, soit de manière stochastique en appliquant la technique de diagnostic probabiliste présentée précédemment. Les incertitudes et les erreurs liées au traitement imposent de définir un seuil de confiance autour de la valeur nulle dans le cas d'une situation nominale sans faute. Puis, en considérant un vecteur de défauts, on établit une matrice des signatures des résidus permettant d'identifier simplement les fautes apparaissant.

Plus de détails sur cette méthode sont fournis dans [27] [24].

2.2.3 Diagnostic de systèmes discrets

Une méthode hors ligne basée sur la modélisation des systèmes à événements discrets (SED) apporte une propriété intéressante sur la diagnosticabilité d'un système.

Définition 2.12 : Système à événements discrets (SED)

Un SED est défini formellement par un quadruplet $G = (X, \Sigma, \mu, x_0)$ tel que :

- X est un ensemble d'états,
- Σ est un ensemble d'événements,
- $\mu \in X \times \Sigma \rightarrow 2^X$ est un ensemble fini de transitions,
- x_0 est l'état initial.

■

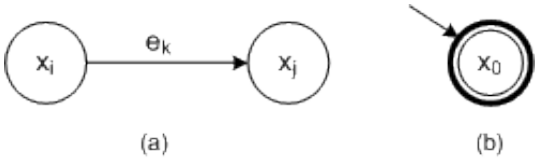

FIG. 2.7 – Représentation graphique d'une transition (a) et d'un état initial (b).

Dans cette approche, le comportement dynamique d'un SED est traduit par une séquence d'événements (également appelée *chemin*) représentée par une séquence de symboles auxquels on applique la théorie des langages. On considère alors qu'un modèle M de SED contient un ensemble d'événements "observables" Σ_o comme des commandes d'un contrôleur ou des mesures issues de capteurs, et des événements "non observables" Σ_{no} dont un ensemble d'événements de fautes Σ_f. Le problème de diagnostic consiste alors à déterminer pour une séquence donnée d'observations s'il existe des chemins cohérents avec cette séquence, et dans ce cas, à répondre à la question : ces chemins sont-ils nominaux (sans faute) ou défaillants (avec occurrence de fautes) ? Pour cela, on utilise les techniques basées sur la recherche opérationnelle pour explorer l'ensemble des chemins possibles. Une réduction du modèle du système par décomposition en modèles de composants permet d'améliorer le calcul d'exploration qui peut s'avérer être assez chronophage [30]. Une autre méthode s'appuyant sur l'outil d'optimisation SAT (satisfiabilité propositionnelle) améliore la détection des chemins qui sont compatibles avec les observations et qui sont soit nominaux soit défaillants, à l'aide d'une écriture du problème sous forme de contraintes SAT [31]. Cependant, l'inconvénient de cette méthode concerne les performances du diagnostic qui repose grandement sur les formulations d'un nombre important de contraintes non triviales. L'auteur propose néanmoins des formulations de contraintes génériques permettant de traduire le modèle du système, les observations et les questions de présence de fautes dans les chemins cherchés. Mais,

on ne dispose pas d'outils ni de règles permettant de valider la pertinence et la correction de ces contraintes.

Ainsi, les motivations de la présentation de ces méthodes de diagnostic sont de fournir un aperçu de ce qui peut être concrètement fait pour diagnostiquer une défaillance. Par la suite, notre approche va consister à tenir pour acquis ces méthodes et à ne considérer qu'une vision abstraite des mécanismes de diagnostic dans une modélisation particulière (voir chapitre 4). La méthode de diagnostic SED est certainement celle qui pourrait être rattachée plus aisément à notre modélisation, bien qu'une hybridation de nos modèles avec les modèles de diagnostic de systèmes continus est également envisageable.

Après le diagnostic du système, la reconfiguration est le mécanisme actif permettant de mitiger l'impact de la faute et de rendre la défaillance invisible du point de vue de l'utilisateur. La reconfiguration consiste à mettre en œuvre différentes actions correctrices. Les principales actions, non réparatrices de la faute, considèrent des moyens palliatifs de fonctionnement dégradé ou de fonctionnement nominal en présence d'une faute. Les moyens utilisés s'appuient essentiellement sur le principe de la redondance [32]. Par ailleurs, le principe de reconfiguration fait essentiellement appel à des propriétés liées à l'architecture et à des motifs de sécurité (voir section 2.1.1).

2.3 Architecture adoptée

2.3.1 Architecture en couches

D'après [19], le concept d'architecture permet de mieux appréhender et de mieux maîtriser un système complexe. Le choix d'une architecture en couches est souvent motivé par une composition du système en éléments de nature différente, se distinguant les uns des autres par leurs fonctionnalités ou leurs aspects temporels sous-jacents [33]. Dans le cas des systèmes embarqués autonomes de contrôle (correspondant à la plupart des systèmes embarqués critiques), l'architecture générique est constituée de 3 couches fonctionnelles : opération, fonction, équipement (voir la partie gauche de la figure 2.8). La couche équipement concerne les fonctions remplies par des équipements du système de contrôle à savoir les actionneurs et capteurs utilisés pour la maîtrise des mouvements éventuels. Au-dessus, on trouve la couche fonction contenant les fonctions de commande bas niveau telles que les différentes fonctions de pilotage et la fonction de navigation d'un drone. La dernière couche de notre architecture est la couche opération regroupant principalement les applications d'aide à la décision, de gestion d'opérations liées à la mission fixée et des applications de supervision globale du système.

Concernant les aspects temporels de ces couches, on peut constater que le temps de réponse augmente avec le niveau de la couche concernée. Ainsi, pour

répondre aux sollicitations de l'environnement ou à des phénomènes physiques évoluant rapidement comme la variation de vitesses de vent ou les turbulences, les capteurs et actionneurs doivent avoir des temps de réponse très faibles (de l'ordre de 0,3s [1]). De même, les fonctions de commande connectées à ces équipements réalisent les calculs d'asservissement avec un temps de réponse un peu plus élevé, incluant le temps de réponse des équipements tout en garantissant la stabilité du contrôle (de l'ordre de quelques secondes). Enfin, le temps de réponse de la couche opération prend en compte des horizons temporels d'analyse plus importants pour définir les actions ou les opérations à réaliser (de l'ordre de quelques dizaines de secondes). Du point de vue de l'autonomie décisionnelle, cette architecture en couches met également en évidence une hiérarchie fonctionnelle entre les couches. La couche supérieure de gestion des opérations coordonne et supervise les activités des fonctions de commande de la couche adjacente inférieure. De même, la couche fonction contrôle et surveille les activités des équipements.

Dans la littérature, de nombreux travaux sur la conception rigoureuse de systèmes tolérants aux fautes font état d'une architecture en couches [34] [35] [36] [37] [38] [39] [40] [41]. L'approche abordée par [36] consiste à considérer la couche de plus niveau comme étant une abstraction de l'application envisagée. La modélisation est ensuite raffinée en ajoutant progressivement le comportement des couches inférieures. De plus, on peut également intégrer à cette modélisation des modes opérationnels pour prendre en compte d'éventuelles défaillances [42].

Dans notre approche, des barrières de sécurité en termes de détection de fautes, d'isolation et de reconfiguration (FDIR) sont établis dans chaque couche. Une faute non détectée ou non résolue au niveau équipement sera traitée par la couche fonction. En cas d'échec du traitement au niveau fonction, la couche opération peut décider de modifier le mode opération en basculant :

- soit vers un mode dégradé (*Back*) incluant un possible repli vers un point géographique préétabli,
- soit vers un mode avorté (*Abort*) qui écourte la phase en cours pour effectuer la suivante,
- soit vers un mode annulé (*Canc*) faisant tomber le drone à l'endroit où il se trouve.

Au niveau des deux premières couches, les modes de défaillance considérés concernent principalement des modes erronés (*Erroneous*) et perdus (*Lost*) indiquant respectivement que l'équipement ou la fonction retournent des valeurs erronées, et que l'équipement ou la fonction ne retournent plus de données en sortie [43]. La partie droite de la figure 2.8 représente les automates de modes de défaillances associés à chaque entité située dans la partie gauche suivant la couche considérée.

[1] Les caractéristiques temporelles données correspondent aux évaluations faites sur le drone étudié

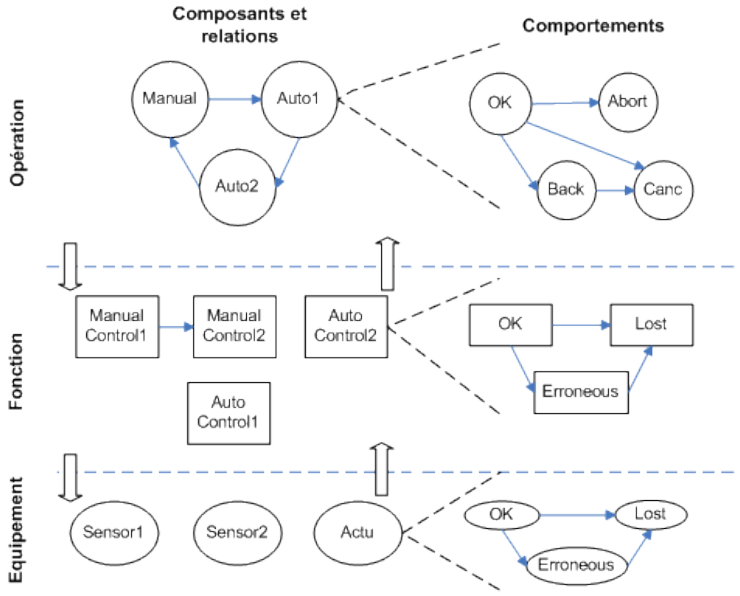

FIG. 2.8 – Architecture en couches d'un système autonome.

Cependant, une architecture ne se caractérise pas uniquement par des éléments considérés individuellement, mais également par les règles ou les propriétés qui sous-tendent cette structuration. Dans la sous-section suivante, on présente les propriétés de sécurité à vérifier par notre système en s'appuyant sur une architecture en couches.

2.3.2 Exigences de sécurité primaires

Le processus d'analyse de sécurité permet de créditer le système d'une plus grande confiance sur son innocuité. Cette analyse repose sur des propriétés ou exigences de sécurité à satisfaire par le système. D'abord, on identifie des événements redoutés à cause de leurs effets sur des personnes, sur des biens, ou sur le système lui-même. Une classification de ces événements en fonction de leurs effets permet de définir des niveaux de criticité associés à ces événements. Ce principe général est par exemple raffiné dans le cas de l'aéronautique par le standard ARP 4754A (voir section 1.3.1). Dans notre cas d'étude, on considère par exemple comme catastrophique le crash du drone dans une zone habitée.

De ce fait, on déduit deux principales propriétés de sécurité à satisfaire, l'une étant qualitative [2.7], l'autre quantitative [2.8] :

Exigence qualitative : « Une faute simple ne doit pas conduire à l'événement redouté de crash dans une zone habitée » (2.7)

Exigence quantitative : « La probabilité d'occurrence de l'événement redouté de crash doit être inférieure à 10^{-x} *par heure de vol* [2] *»* (2.8)

Cependant, ces exigences de sécurité primaires sont nécessairement traduites en exigences fonctionnelles permettant d'assurer un comportement sûr et tolérant aux fautes. Dans notre architecture, les exigences fonctionnelles correspondent essentiellement à des propriétés caractérisant les composants à l'intérieur des couches, ainsi que les relations entre les couches.

2.3.3 Spécification des mécanismes de sécurité

Pour tenir les exigences de sécurité assignées au système global, l'architecture en couches doit mettre en œuvre au moins trois types de mécanismes de sécurité.

Tout d'abord, des redondances fonctionnelles ou matérielles doivent être prévues. Des éléments pourront être considérés comme redondants s'il y a *"équivalence"* comportementale entre eux. De plus, pour pouvoir montrer qu'une paire de composants redondants permet de tolérer une faute, il faut avoir des garanties sur leur *"indépendance"* intrinsèque : deux noms de composants distincts correspondent bien à deux objets physiquement différents et même dissemblables dans leur mode de conception. Il restera ensuite à montrer que la conception de l'architecture n'introduit pas des dépendances non souhaitées. Par la suite, la propriété de *"redondance sûre"* est définie comme étant la conjonction des propriétés d'indépendance et d'équivalence.

Puis, des fonctions de sécurité élémentaires doivent être introduites dans chaque couche pour surveiller l'état de santé du système et activer des fonctions de secours en cas de détection de panne. Ces fonctions exploitent en particulier l'*"observabilité"* de l'état des composants, i.e. la possibilité d'accéder à la connaissance de l'état de santé des composants.

Enfin, des services de sécurité doivent coordonner les redondances et fonctions de sécurité élémentaires. Sur le plan fonctionnel, un service sera donc défini :

- statiquement par la *"sélection"* d'évènements élémentaires coordonnés et
- dynamiquement par les *"contraintes temporelles"* qui synchronisent le déclenchement des évènements du service avec les autres évènements de la couche [44] [45].

De plus, dans une architecture en couches, les services sont supposés être les seuls moyens d'échange entre couches. Ainsi, toutes les activités des éléments d'une couche ne sont pas des services : seules les activités qui sont visibles par la couche supérieure sont des services. Un service peut donc être vu comme une machine abstraite qui génère une fonctionnalité spécifique pour la couche supérieure en coordonnant des activités des éléments de la couche et des services de plus bas niveau (si nécessaire).

[2]x dépendant de la densité de population de la zone survolée dans le cas des drones.

FIG. 2.9 – Drone RMAX de l'ONERA.

2.3.4 Présentation du système étudié

Le cadre applicatif de cette étude concerne le système de contrôle avionique du drone autonome RMAX développé par l'ONERA (figure 2.9). Cette avionique implémente trois modes de pilotage (dans le bloc *Pilot*, figure 2.10) qui sont sélectionnés manuellement ou de manière autonome, un module de guidage (*Guidance*) et un module de navigation (*Navigation*) autorisant une navigation intégrant comme charge utile une caméra (*Payload*). Le premier mode de pilotage correspond à un pilotage manuel (aussi appelé par la suite *MP*). Le second mode est relatif à un pilotage automatique grossier *AP1*, tandis que le troisième mode *AP2* est associé à un pilotage automatique plus fin lié à des capteurs spécifiques tels que le GPS. Ces trois modes de pilotage communiquent directement avec des capteurs et actionneurs afin de contrôler le drone. De plus, pour assurer une plus grande autonomie décisionnelle, des fonctions dites de "haut niveau" telles que la planification (*Decision/Planning*) et la supervision (*Supervision Unit*) gèrent les opérations liées à une mission donnée. Ainsi, le système de contrôle du drone est organisé au sein d'une architecture logique hiérarchique, en couches fonctionnelles. Dans la représentation figure 2.10, les données échangées entre les blocs ne correspondent pas aux variables réellement échangées ; elles indiquent le type de relations mis en place entre les blocs ou composants fonctionnels.

La mission de ce drone est constituée de plusieurs phases opérationnelles combinées aux différents modes de pilotage et à des zones survolées : décollage manuel, vol d'avancement à vue en zone non habitée, vol autonome hors vue en

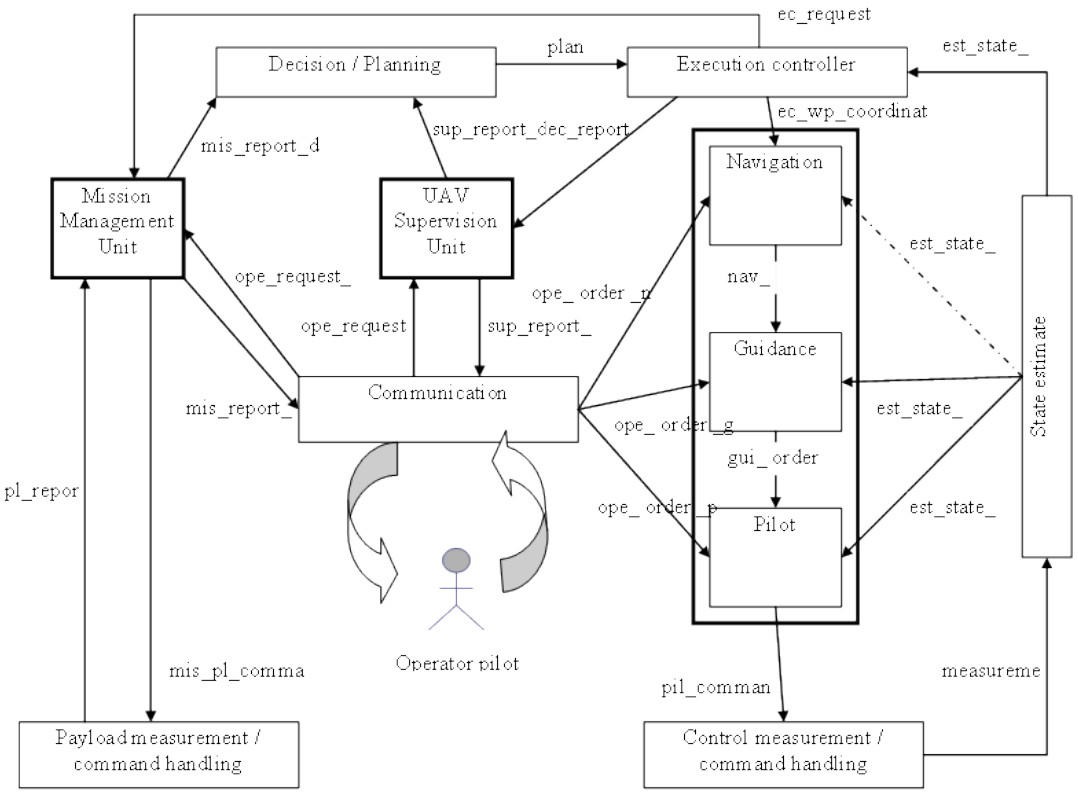

FIG. 2.10 – Architecture du système de contrôle du drone RMAX.

zone habitée, atterrissage manuel (figure 2.11). Pour notre étude, on suppose que le décollage se déroule avec zéro panne détectée.

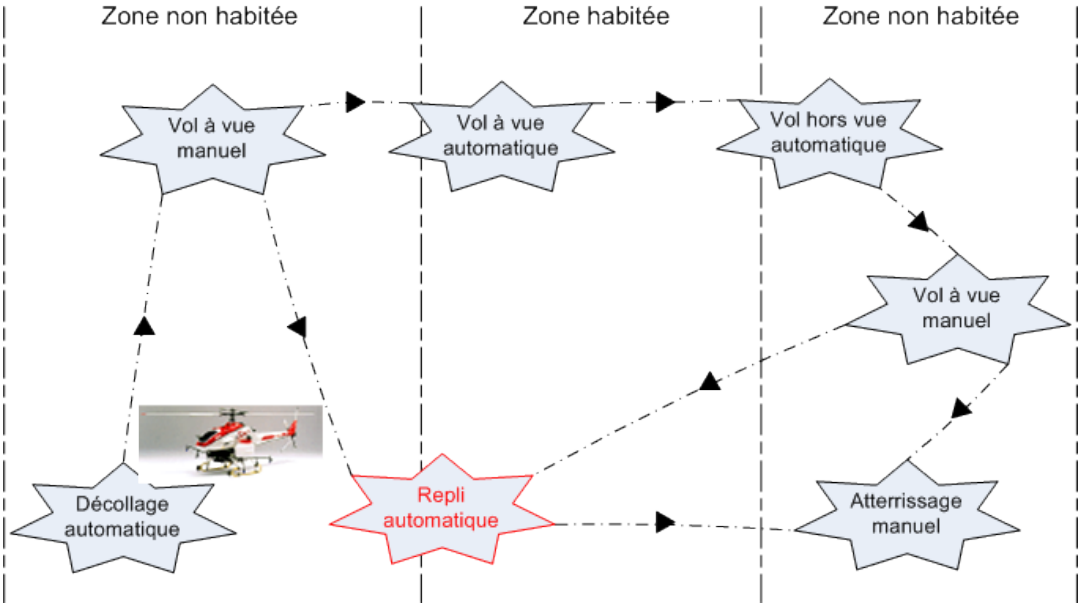

FIG. 2.11 – Un scénario opérationnel du drone.

En bref L'architecture en couches fonctionnelles est une solution étudiée pour structurer la gestion de la sécurité du système de contrôle d'engins autonomes. Elle comporte trois couches abstraites dont les spécificités autorisent une tolérance aux fautes graduelle par des mécanismes de sécurité intrinsèques aux couches. Ainsi, certaines fautes apparaissant au niveau équipement sont traitées à ce même niveau, tandis que d'autres sont propagées aux niveaux supérieurs. Une formalisation des ces procédés au sein de l'architecture générique adoptée est apportée dans le chapitre 4. Cette formalisation est appliquée au drone RMAX dans ce document et est applicable à d'autres systèmes autonomes de contrôle, tels que les systèmes de contrôle-commande de satellites.

Chapitre 3

La modélisation en Event-B

Résumé
Ce chapitre rappelle tout d'abord les notions de logique des prédicats du premier ordre utilisée dans Event-B avant d'introduire le formalisme Event-B. Puis, il montre comment le principe de vérification par preuves de Event-B permet d'améliorer la confiance dans les modèles développés. La logique des prédicats du premier ordre et la théorie des ensembles permettent de modéliser des systèmes et leur dynamique avec plus ou moins de simplicité. Event-B spécialise et étend ce cadre théorique pour faciliter la modélisation et le raisonnement sur des systèmes à événements discrets.

« L'idée sans le mot serait une abstraction ; le mot sans l'idée serait un bruit ; leur jonction est leur vie. »
Victor Hugo - Post-Scriptum de ma vie

Les chapitres précédents ont introduit de façon informelle la problématique des architectures de systèmes complexes et ont spécifié leur comportement vis-à-vis de la sécurité. Cette première étape a mis alors en évidence des difficultés à extraire puis à traiter des propriétés nécessaires aux objectifs attendus. Les étapes suivantes vont consister à spécifier formellement ces propriétés pour les rendre plus claires et moins ambiguës.

En effet, la spécification formelle est l'expression en langage formel d'un ensemble de propriétés qu'un système doit satisfaire. En d'autres termes, la spécification formelle est une partie de la spécification standard qui s'appuie sur une *notation mathématique* pour exprimer certaines exigences sous forme de propriétés. On se base sur des théories mathématiques stables et prouvées, telles que la théorie des ensembles et la logique des prédicats, pour modéliser le système et pour décrire son comportement attendu. Par ailleurs, les progrès de l'informatique permettent de mettre à disposition des équipes de spécification autrement spécialisées des outils et des techniques informatisés cohérents. On parle alors de *méthode formelle* dès lors qu'il existe un cadre rationnel d'outils et de règles de *modélisation* et de *raisonnement* s'appliquant aussi bien à la spécification qu'à la conception de systèmes.

L'intérêt des mathématiques réside dans la qualité de ses expressions (claires, concises, précises) et dans les traitements qu'elles peuvent subir, contrairement à une spécification informelle en langage naturel. Cependant, les inconvénients d'une telle spécification concernent :

- un premier obstacle à franchir lorsqu'il s'agit de maîtriser une notation faisant appel à des compétences en mathématiques ;
- la modélisation qui peut faire ressortir des difficultés inhérentes à la technique utilisée. Elle peut dans certains cas conduire à retarder le développement du système.

A vrai dire, un modèle ne constitue pas la (ou une) réponse au problème soulevé. La modélisation, et par conséquent la spécification formelle, permet principalement d'identifier les propriétés du système et de prouver que ces propriétés sont présentes dans le système final à construire.

Ce chapitre rappelle tout d'abord les notions de logique des prédicats du premier ordre et de logique de Hoare utilisées dans la méthode formelle Event-B avant d'introduire le formalisme Event-B. Puis, il montre comment le principe de vérification par preuves de Event-B permet d'améliorer la confiance dans les modèles développés.

3.1 Logique des prédicats du premier ordre et logique de Hoare

Une logique est un *langage* muni d'un mécanisme de déduction. Elle fournit une syntaxe pour énoncer des *propriétés* et une *sémantique*. La logique des prédicats du premier ordre permet de parler et de *raisonner* en tenant compte non seulement de la structure en phrase des énoncés mais aussi des *constituants* des énoncés.

Dans cette section, on verra quels sont les constituants qui interviennent dans les énoncés, les connecteurs logiques introduits pour prendre en compte les constituants et d'une manière systématique, on définira formellement le langage de cette logique. Puis, on verra comment donner un sens aux constituants et comment le sens d'un énoncé est défini à partir du sens de ces constituants, i.e. la sémantique à donner au langage. Enfin, nous verrons comment raisonner sur cette logique dans différents systèmes déductifs.

3.1.1 Langage de la logique

▶ Concepts

Informellement, si l'on tente de décomposer une proposition ou un énoncé, on met en évidence deux types de composants : les noms de *prédicats*, qui énoncent quelque chose (une propriété, une relation) au sujet d'*objets*, désignés par d'autres noms.

On notera que les objets peuvent être déterminés, ou indéterminés. Les noms déterminant directement un objet sont appelés *constantes* alors que les noms dénotant des objets indéterminés sont appelés *variables*.

Les prédicats peuvent porter sur un, deux, n objets. On dit alors que les prédicats sont d'*arité* 1, 2, ... n ou *monadiques*, *dyadiques*, ... *n-adiques* ou *unaires*, *binaires*, ... *n-aires*. Les prédicats d'arité 1 expriment des propriétés alors que les prédicats d'arité > 1 dénotent des relations.

On notera que ce qui est intéressant, ce n'est pas nécessairement de raisonner sur des objets bien déterminés mais plutôt sur *tous les objets* ou *un certain objet*. Pour exprimer ce type de concept, on introduit de nouveaux connecteurs logiques, les quantifieurs universel $\forall$ et existentiel $\exists$ qui, appliqués à une variable x se lisent respectivement $\forall x$: "pour tout x" et $\exists x$: "il existe x tel que".

Si l'on reprend l'exemple de l'énoncé suivant "Si les humains sont mortels et Socrate est humain, alors Socrate est mortel", on peut dénoter par :

- la variable x : un individu indéterminé,
- la constante S : Socrate,
- le prédicat monadique H : la propriété "être humain",
- le prédicat monadique M : la propriété "être mortel"

L'énoncé devient alors :

$$(\forall x(H(x) \Rightarrow M(x)) \wedge H(S)) \Rightarrow M(S)$$

Les *fonctions* permettent de désigner des objets indirectement à partir d'autres objets.

▶ Alphabet

Définition 3.13 : Alphabet

L'alphabet de la logique des prédicats du premier ordre est constitué de :

a) l'alphabet de la logique propositionnelle c'est-à-dire :
   - les noms de propositions : P_1, P_2, ..., P_n, ...,
   - les connecteurs logiques : $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$,
   - les signes de ponctuation : "(" et ")"

b) d'un nombre infini dénombrable de constantes : a_1, a_2, ..., a_n, ...,

c) d'un nombre infini dénombrable de variables : x_1, x_2, ..., x_n, ...,

d) d'un nombre infini dénombrable de prédicats k-adiques : P_{k1}, P_{k2}, ..., P_{kn}, ...,

e) d'un nombre infini dénombrable de symboles de fonction d'arité k : F_{k1}, F_{k2}, ..., F_{kn}, ...,

f) du quantifieur universel $\forall$ et du quantifieur existentiel $\exists$.

■

Remarque

Lorsqu'il n'y aura pas d'ambiguïté, on omettra l'indice indiquant l'arité d'un prédicat ou d'une fonction. On se permettra aussi d'utiliser d'autres noms de constantes, variables, propositions, prédicats ou fonctions dans un souci de lisibilité.

▶ Syntaxe

On donne un ensemble de règles pour définir les formules bien formées du langage. Au préalable, on définira les notions de *termes* et de *formules atomiques*.

Terme

On a vu que les constituants du langage étaient de deux types, les objets et les prédicats. On a vu qu'il y avait plusieurs manières de désigner un objet. Les

termes sont des dénotations d'objets correctement construites. Formellement, les termes de la logique des prédicats du premier ordre sont définis inductivement de la manière suivante.

Définition 3.14 : Termes

a) Les variables et les constantes sont des termes de la logique des prédicats du premier ordre.

b) Si F est une fonction d'arité k et $t_1, \ldots, t_k$ sont k termes du langage, alors F($t_1, \ldots, t_k$) est un terme de la logique des prédicats du premier ordre.

■

Formule atomique

Les formules atomiques sont les énoncés primitifs construits en appliquant un nom de prédicat k-adique à k termes. Formellement, les formules atomiques de la logique des prédicats sont définies inductivement de la manière suivante.

Définition 3.15 : Formules atomiques

a) Les propositions sont des formules atomiques de la logique des prédicats du premier ordre.

b) Si P est un prédicat k-adique et $t_1, \ldots, t_k$ sont k termes, alors P($t_1, \ldots, t_k$) est une formule atomique de la logique des prédicats du premier ordre.

■

Formule bien formée

Les *formules bien formées* ou fbf de la logique des prédicats du premier ordre sont alors définies par les règles suivantes.

Définition 3.16 : Formules bien formées

a) Toute formule atomique de la logique des prédicats du premier ordre est une fbf.

b) Si A, B sont des fbf et x une variable arbitraire alors $\neg A$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$, $\forall x\ A$, $\exists x\ A$ sont des fbf.

c) Seules les formules vérifiant a) et b) sont des fbf.

■

► **Variables libres et liées**

La *portée* d'un quantificateur est la suite de symboles auxquels il s'applique. Dans la formule "$\forall x$ *A*", "x A" est la portée de $\forall$ et A est la portée de $\forall x$. On a une définition analogue pour le quantificateur existentiel.

Une *occurrence* d'une variable x est *liée* dans une fbf ssi (si et seulement si) elle suit immédiatement un quantificateur ou est sous la portée d'un quantificateur immédiatement suivi d'une autre occurrence de x. Une occurrence d'une variable x qui n'est pas liée dans une fbf est dite *libre* dans cette fbf.

Exemple

Dans $\forall x\ (\exists y\ P(x,y) \wedge Q(x,z)) \wedge R(x)$, les occurrences 1, 2 et 3 de x sont liées, tandis que l'occurrence 4 de x est libre ; les occurrences 1 et 2 de y sont liées ; l'occurrence de z est libre.

Cette notion de variables libres ou liées est importante pour plusieurs raisons et en particuliers *pour savoir si on peut donner un sens à une fbf ou pas.*

Considérons le cas de la formule atomique suivante M(x) où apparaît une occurrence libre de la variable x. Comme la variable n'est pas dans le champ d'un quantificateur, elle désigne un objet totalement indéterminé. La formule atomique exprime alors que la propriété désignée par M ("être mortel" par exemple) est vérifiée par un objet indéterminé. Il est alors impossible de dire s'il est vrai ou faux que la propriété M est vérifiée par x, objet indéterminé. Par contre, les formules $\forall x$ M(x) et $\exists x$ M(x) où les occurrences de x sont liées, peuvent être évaluées. Les questions "est-ce que tous les objets sont mortels ?" et "est-ce qu'il existe un objet mortel ?" ont un sens. Les variables placées dans la portée d'un quantificateur prennent un sens déterminé.

Une formule est *fermée* (ou *close*) si elle ne contient pas de variables libres. Sinon elle est ouverte.

Une *substitution* de variables dans une expression E est le résultat simultané du remplacement de toutes occurrences libres des variables à substituer dans E par leur terme associé. $E\{p \backslash q\}$ est appelé *instance* de E où toutes les occurrences de p dans E sont remplacées par q.

3.1.2 Sémantique

Nous allons à présent étudier comment on donne un sens aux fbf du langage. Une fbf est un énoncé qui normalement est soit vrai, soit faux. Dans cet énoncé peuvent figurer des composants (variables, constantes, termes, noms de prédicats). Comment donne-t-on un sens à ces composants ? Comment ces sens se composent-t-ils pour donner un sens à des formules atomiques ou à des fbf ? C'est ce que nous étudierons dans cette section.

▶ Concepts

Comment relier objets et prédicats à une réalité

Une constante désigne un objet précis dans l'univers dont parle l'énoncé où la constante figure. On appelle cet univers, univers du discours ou *domaine de discours D.*

Exemples

S désigne Socrate, zéro désigne le nombre 0. On voit donc que D peut être hétérogène et que l'on met a priori sur le même plan des hommes et des nombres.

Une variable va, elle aussi, référencer D ; elle désigne n'importe quel objet de D : homme, nombre, autre, ...

Pour arriver à distinguer les objets du domaine, on dispose des prédicats monadiques. H désigne l'ensemble des objets de D qui sont humains, M caractérise les mortels.

Plus généralement, un prédicat n-adique désigne l'ensemble des n-uplets de D qui satisfont la relation n-aire désignée par le prédicat. Cet ensemble est appelé l'*extension* du prédicat.

Exemple

Si Egal est le nom du prédicat dénotant la relation d'identité entre objets du domaine, alors Egal désigne l'ensemble des couples (d,d) pour tout objet d de D.

Dans tous les cas, le concept de domaine est fondamental et le principe de désignation d'un objet ou d'un ensemble de t-uple d'objets du domaine permet de donner dans un cadre unifié un sens (*interprétation*) aux deux types de constituants logiques, objet et prédicat.

Comment donner un sens aux formules à partir de l'interprétation des objets et des prédicats

En plus, on peut établir un lien direct entre les *fonctions d'interprétation* qui vont donner un sens aux prédicats et aux termes et la *fonction de distribution de valeurs de vérité* applicables aux fbf.

Exemple

Considérons par exemple la formule atomique M(S). Cette formule sera vraie ssi l'objet désigné par S appartient à l'ensemble désigné par le prédicat M.

On notera au passage qu'on ne peut donner un sens (vrai ou faux) à la formule atomique M(x) sans *assigner* un sens à x puisqu'on ne peut pas raisonnablement décider si un objet indéterminé appartient ou pas à un ensemble. Par contre, on peut donner un sens à $\forall x$ M(x) puisque l'on peut décider si tous les objets du domaine appartiennent ou pas à l'ensemble des objets désignés par M. De manière analogue, on peut décider s'il existe ou pas un élément de

D appartenant à l'extension de M, autrement dit, on peut décider si l'extension de M est vide ou pas. On retrouve l'importance du concept de variable libre ou liée et l'on va voir comment le prendre en compte rigoureusement.

▶ Structure d'interprétation

Définition 3.17 : Interprétation

Une interprétation (ou structure d'interprétation) M pour un langage L de la logique des prédicats du premier ordre est un couple M = (D,I) où
- D est un ensemble non vide appelé le domaine de la structure et
- I est une fonction appelée la fonction d'interprétation de la structure.

I permet d'interpréter constantes, fonctions et prédicats de la manière suivante :

a) pour toute constante c du langage L, I(c) est un élément de D
b) pour tout symbole de fonction F d'arité $k \geq 0$, I(F) : $D^k \rightarrow D$ est une fonction k-aire
c) pour tout symbole de prédicat P k-adique $k \geq 0$, I(P) : $D^k \rightarrow \{vrai, faux\}$ est la fonction caractéristique d'une relation k-aire de D.

■

On remarque qu'un prédicat d'arité 0 (sans argument) est une variable propositionnelle et l'on retrouve que l'interprétation d'une variable propositionnelle est une valeur de vérité. De même, une fonction d'arité 0 est une constante.

▶ Structure d'interprétation

On cherche un principe de distribution de valeur de vérité *compositionnel* : connaissant les interprétations des constantes, fonctions, prédicats, ..., on voudrait en déduire un principe de distribution de valeur de vérité aux atomes (ou formules atomiques, ou variables propositionnelles ou propositions atomiques) puis aux fbf. Problème : on ne peut donner un sens aux formules contenant des variables libres sans avoir assigner au préalable une valeur aux variables libres.

Définition 3.18 : Assignation

Soit L un langage de la logique des prédicats du premier ordre et M=(D,I) une structure sur L. Une *assignation* s est toute fonction $V \rightarrow D$ de l'ensemble V des variables de L dans D. Une *x-variante* d'une assignation $s : V \rightarrow D$ est une nouvelle assignation $s' : V \rightarrow D$ telle que $s'(y) = s(y)$ pour tout y différent de x et $s'(x) \neq s(x)$. ■

Définition 3.19 : Evaluation

L'*évaluation val(I,s) de termes* quelconques dans une interprétation I et une assignation s sur un domaine D est alors définie de la manière suivante :
- si x est une variable, $val(I,s)(x) = s(x) \in D$
- si c est une constante, $val(I,s)(c) = I(c) \in D$
- $val(I,s)(F(t_1,\ldots,t_k)) = I(F)(val(I,s)(t_1),\ldots,val(I,s)(t_k))$

L'*évaluation val(I,s) de formules* quelconques dans une interprétation I et une assignation s sur un domaine D est alors définie de la manière suivante :
- $val(I,s)(t_1 = t_2)$ ssi $val(I,s)(t_1) = val(I,s)(t_2)$
- $val(I,s)(P(t_1,\ldots,t_k)) = vrai$ ssi $I(P)(val(I,s)(t_1),\ldots,val(I,s)(t_k)) = vrai$
- $val(I,s)(\neg A) = vrai$ ssi $val(I,s)(A) = faux$
- $val(I,s)(A \wedge B) = vrai$ ssi $val(I,s)A$ et $val(I,s)B$
- $val(I,s)(\forall x\ A(x)) = vrai$ ssi pour tout s' x-variante de s, $val(I,s')(A(x)) = vrai$
- $val(I,s)(\exists x\ A(x)) = vrai$ ssi il existe s' x-variante de s, telle que $val(I,s')(A(x)) = vrai$

■

► **Satisfiabilité, validité**

On dit qu'une interprétation I et une assignation s sur un domaine D $satisfont$ une formule A ssi $val(I,s)(A) = vrai$. On note $I, s \models A$.

Une formule A est dite $satisfiable$ s'il existe une interprétation I et une assignation s sur un domaine D satisfaisant A.

Une structure M=(D,I) est appelé *modèle de A* ssi pour toute assignation s sur D, $I, s \models A$. On note $I \models A$.

Une formule A est dite *valide* ssi toute structure M est un modèle de A. On note $\models A$.

Une formule A *implique logiquement* une formule B ssi $\models A \Rightarrow B$. Deux formules A et B sont *logiquement équivalentes* ssi $\models A \Leftrightarrow B$. On note $A \equiv B$.

Une formule B est une *conséquence logique* de A ssi tout modèle de A est un modèle de B. On note $A \models B$.

3.1.3 Système déductif de jugement

On a vu dans les sections sur la syntaxe et la sémantique de la logique des prédicats du premier ordre comment donner un sens "vrai" ou "faux" à des formules de cette logique. On a vu qu'il y a des formules toujours vraies (va-

lides) et d'autres qui peuvent être falsifiables (fausses dans certaines structures d'évaluation).

Plusieurs techniques ont été développées pour étudier la validité de formules logiques. Toutes reposent sur l'idée suivante. Comme une formule valide est vraie dans toute structure d'évaluation, *la vérité de la formule ne découle pas d'éléments externes à la formule mais des caractéristiques de ses composants de bases* : les termes, les formules atomiques et la façon dont ils sont reliés par les connecteurs logiques. Les différentes techniques de preuve de validité s'emploient à *décomposer les formules complexes* selon des stratégies particulières afin de rendre explicite les composants qui induisent la validité (ou la falsifiabilité) d'une formule.

Ainsi, le jugement, ou inférence, est une opération logique qui consiste à tirer une conclusion vraie (de valeur de vérité vraie) à partir d'une série de propositions reconnues comme vraies (appelées *axiomes*) et d'un petit ensemble de règles permettant de produire toutes les vérités (appelées *règles d'inférence*).

Dans cette section, on étudiera la méthode définie par le système de séquents de Gentzen G pour la logique des prédicats du premier ordre. On peut voir un séquent comme une structure de données permettant de distinguer les formules vraies et les formules fausses. Le système de séquents est un ensemble de règles de décomposition préservant la validité des ensembles de formules décomposées.

► Syntaxe et sémantique d'un séquent

Un *séquent* est défini par deux ensembles de formules bien formées de la logique des prédicats du premier ordre, Γ et Δ, reliés par le symbole $\vdash$. On le note $\Gamma \vdash \Delta$. Γ est appelé *antécédent* (ou *hypothèse*) du séquent, Δ est appelé *conséquent* (ou *conclusion*).

Un séquent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ est satisfiable (resp. falsifiable) ssi il existe une structure M=(D,I) et une assignation s, telles que : $val(I, s)(A_1 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee \ldots \vee B_m) = vrai$ (resp. $faux$).

Un séquent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ est valide ssi $A_1 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee \ldots \vee B_m$ est une formule valide c'est-à-dire $\models A_1 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee \ldots \vee B_m$. On note $\models (A_1, \ldots, A_n \vdash B_1, \ldots, B_m)$.

► Système de preuve

Un *axiome* est un séquent $A_1, \ldots, C, \ldots, A_n \vdash B_1, \ldots, C, \ldots, B_m$ pour lequel une formule C apparaît dans l'antécédent et le conséquent du séquent.

Une *règle d'inférence* entre séquents est de la forme :

Elle s'interprète de la manière suivante : si les séquents de la prémisse sont valides, alors le séquent conclusion est aussi valide.

Prémisse : $\Gamma_1 \vdash \Delta_1 \quad \ldots \; \Gamma_n \vdash \Delta_n$
Conclusion : $\Gamma \vdash \Delta$

Un *système S de preuve* de séquents est défini par un ensemble fini de schéma d'axiomes et de règles d'inférence.

Un *arbre de déduction* pour un système donné S est un arbre dont :
- la racine est un séquent
- chaque nœud est soit une feuille, soit a un ou deux fils tels que le nœud père peut être dérivé des nœuds fils par application d'une des règles d'inférence du système S.

Un *arbre de preuve* pour un système donné S est un arbre de déduction de S dont les feuilles sont des séquents axiomes. On dit que la racine de l'arbre de preuve est un *séquent prouvable.*

$\wedge$-gauche : $\frac{H,P,Q \vdash R}{H,P\wedge Q \vdash R}$	$\wedge$-droite : $\frac{H \vdash P \quad H \vdash Q}{H \vdash P\wedge Q}$
$\vee$-gauche : $\frac{H,P \vdash R \quad H,Q \vdash R}{H,P\vee Q \vdash R}$	$\vee$-droite : $\frac{H \vdash P}{H \vdash P\vee Q}$ ou $\frac{H \vdash Q}{H \vdash P\vee Q}$
$\Rightarrow$-gauche : $\frac{H,Q \vdash R \quad H \vdash R,P}{H,P\Rightarrow Q \vdash R}$	$\Rightarrow$-droite : $\frac{H,P \vdash R,Q}{H \vdash R,P\Rightarrow Q}$
$\neg$-gauche : $\frac{H \vdash R,P}{H,\neg P \vdash R}$	$\neg$-droite : $\frac{H,P \vdash R}{H \vdash R,\neg P}$
$\forall$-gauche : $\frac{H,P\{t\backslash x\},\forall x \; P \vdash R}{H,\forall x \; P \vdash R}$	$\forall$-droite : $\frac{H \vdash R,P\{t\backslash x\}}{H \vdash R,\forall x \; P}$
$\exists$-gauche : $\frac{H,P\{t\backslash x\} \vdash R}{H,\exists x \; P \vdash R}$	$\exists$-droite : $\frac{H \vdash R,P\{t\backslash x\},\exists x \; P}{H \vdash R,\exists x \; P}$

TABLEAU 3.1 – Règles d'inférence de séquents (sans égalité).

3.1.4 Logique de Hoare

Dès les années 40, Turing montre dans ses travaux que le raisonnement sur des programmes séquentiels devient plus aisé si des annotations relatives aux propriétés sur l'état de la mémoire sont introduites à des endroits particuliers dans ces programmes [46]. Plus tard, des mathématiciens en logique, comme Hoare, proposent un cadre méthodologique pour prouver la cohérence entre les programmes et les propriétés encadrant des instructions [47] [48] [49]. Ils définissent ainsi la spécification formelle. La logique de Hoare est alors un système de déduction pour la vérification de programmes dont l'idée est de considérer un programme comme une transformation des propriétés sur l'état de la mémoire [50]. L'état de la mémoire est caractérisé par les valeurs associées aux

variables du programme. La définition syntaxique de la logique de Hoare est donnée comme suit.

Définition 3.20 : Logique de Hoare

La logique de Hoare est un triplet $< L, Ax, R >$ où :
- L définit une collection de formules de la forme pPq avec p et q deux propriétés exprimées sous forme de prédicats et P un programme,
- Ax est un ensemble d'axiomes qui s'écrivent pour toute instruction d'affectation de l'expression e à la variable d'état x : [Aff] $\{[x := e]p\}x := e\{p\}$,
- R est l'ensemble des règles de déduction comprenant :

$$[Seq] \quad \frac{\{p\}A_1\{q'\}, \{q'\}A_2\{q\}}{\{p\}A_1; A_2\{q\}}$$

$$[Pré] \quad \frac{\{p'\}A\{q\}, p \Rightarrow p'}{\{p\}A\{q\}} \qquad [Post] \quad \frac{\{p\}A\{q'\}, q' \Rightarrow q}{\{p\}A\{q\}}$$

$$[Et] \quad \frac{\{p\}A\{q\}, \{p\}A\{q'\}}{\{p\}A\{q \wedge q'\}} \qquad [Ou] \quad \frac{\{p\}A\{q\}, \{p'\}A\{q'\}}{\{p \vee p'\}A\{q\}}$$

■

Lors de l'exécution d'une instruction voire d'un programme, notés A, la sémantique de la formule $\{p\}A\{q\}$ signifie que, sous l'hypothèse que A soit correct si les variables d'état de A satisfont p avant l'exécution de A (p définit une pré condition par rapport à l'exécution de A), alors après exécution, ces variables satisfont q (q est appelé une post condition liée à A ; $A\{q\}$ est le prédicat résultant égale à q dans lequel les variables ont été modifiées par A). Autrement dit, l'exécution de A transforme n'importe quel état mémoire respectant p en un état mémoire satisfaisant q. Par conséquent, en partant d'une spécification définissant les propriétés attendues q à partir des propriétés p de l'environnement, on détermine le meilleur programme A comme étant celui qui vérifie $\{p\}A\{q\}$. Pour avoir une expression axiomatique $\{p\}A\{q\}$ la plus efficace et la plus précise possible, on recherche une pré condition p la plus faible possible (modèle d'interprétation des variables d'état large) pour un programme A donné et une post condition q que l'on veut la plus forte possible (modèle d'interprétation réduit). La plus faible pré condition (*weakest precondition*, en anglais) est notée *wp(A, q)*. Dans le cas d'une affectation ou substitution S, $wp(S, q) = S\{q\}$ (axiome *[Aff]*). Cette recherche arrière, depuis la post condition jusqu'à la pré condition spécifiée, peut se généraliser à l'ensemble des instructions d'un programme pour former une annotation cohérente de ce dernier. Des recherches récentes ont étayé cette technique par des indications sur les variables d'état provenant d'analyse propre à l'automatique comme les ellipses invariantes de Lyapunov [51].

On verra dans la section suivante comment les règles de déduction de Event-B sont liées à la logique de Hoare.

3.2 Présentation Event-B

On a vu que la logique des prédicats du premier ordre est adaptée à l'analyse d'énoncés logiques. Cependant, dans l'étude du comportement de systèmes, on a besoin de techniques et de méthodes plus sophistiquées pour modéliser plus facilement ce comportement. Ainsi, Event-B correspond à une technique de modélisation de systèmes à événements discrets basée sur la logique des prédicats du premier ordre et la théorie des ensembles[1]. Event-B est un formalisme de spécification formelle de systèmes complexes s'appuyant sur la méthode B classique [52] et sur la modélisation de systèmes à événements discrets [53]. La méthode B a été initialement conçue comme une méthode de développement de logiciels depuis l'étape de spécification abstraite jusqu'à l'étape de conception détaillée directement implémentable dans un langage de programmation. L'introduction des événements $EVENTS$ dans Event-B à la place des opérations $OPERATIONS$ de la méthode B permet de prendre en compte les aspects comportementaux d'un système décomposé en éléments en interaction.

Le contenu de cette section portera essentiellement sur la description des éléments constitutifs d'un modèle de base Event-B, ainsi que sur les règles d'obligation de preuves liées à un modèle fondant sa pertinence et sa correction. Ces règles sont dérivées de la sémantique de Event-B et les obligations générées seront à prouver à l'aide de règles d'inférence pour la logique des prédicats et la théorie des ensembles. Par la suite, on abordera également le principe de raffinement qui rend la méthode de modélisation plus progressive.

3.2.1 Modèle Event-B de base

Définition 3.21 : Event-B

Un modèle Event-B est défini formellement par un n-uplet $M = (c,P,v,I,K,E)$ où :

- c est un ensemble de constantes caractérisant le système ;
- v représente l'ensemble des variables d'état ;
- $P(c)$ est une collection d'axiomes relatifs aux constantes ;
- $I(c,v)$ désigne une collection de propriétés invariantes spécifiant un comportement sûr du système ;
- E est un ensemble d'événements ;

[1] Théorie des ensembles de Nicolas Bourbaki : certains pensent que cela a inspiré le nom de la méthode B

– K est un événement particulier d'initialisation affectant des valeurs aux variables d'état.

■

D'un point de vue pratique, un modèle Event-B est composé de deux entités de base, à savoir le "*Contexte*" et la "*Machine*". Par rapport à la définition donnée ci-dessus, le contexte regroupe toutes les caractéristiques statiques du système telles que les éléments constants et les axiomes (voir tableau 3.2). Il permet de spécifier le domaine des données qui fixe le périmètre de l'évolution du système.

Par ailleurs, la machine complète la définition du modèle Event-B en décrivant une abstraction du comportement dynamique du système à l'aide d'un ensemble de variables d'état, d'un ensemble d'invariants et d'un ensemble d'événements agissant sur ces variables (voir tableau 3.3). La machine abstraite permet de fixer un ensemble de propriétés attendues pour le comportement.

CONTEXT Nom_Contexte
SETS
Déclaration des noms des ensembles d'objets du système
CONSTANTS
Déclaration des noms des constantes du système
AXIOMS
Déclaration des noms des axiomes (ou théorèmes) du système
END

TABLEAU 3.2 – Structure générique d'un contexte.

De façon plus détaillée, les variables d'état de la machine déclarées dans la clause $VARIABLES$ sont contraintes par les propriétés déclarées dans la clause $INVARIANTS$. Dans cette dernière clause, les variables sont en effet typées en fonction des ensembles définis dans la clause $SETS$ du contexte associé. On peut distinguer trois types d'ensembles :

– le type abstrait : représente un ensemble fini d'objets ou d'individus ayant des caractéristiques communes non définies explicitement. Par exemple : $SENSOR$, $FUNCTION$;
– le type énuméré : correspond à un ensemble d'éléments distincts bien identifiés. Par exemple : $E\_FLAG = \{active, idle, spare, off\}$;

MACHINE Nom_Machine
SEES Nom_Contexte
VARIABLES
Déclaration des noms des variables d'état du modèle
INVARIANTS
Déclaration des invariants (ou théorèmes) du système
VARIANT
Déclaration du variant
EVENTS
Initialisation
Définition des valeurs initiales des variables
Event *Nom_Evenement* $\widehat{=}$
Définition des conditions et des actions d'un événement
END

TABLEAU 3.3 – Structure générique d'une machine abstraite.

– le type standard : concerne les ensembles classiques tels que les entiers naturels $\mathbb{N}$ ou les entiers relatifs $\mathbb{Z}$ ou les intervalles $i..j$ ou l'ensemble de booléens $BOOL$.

Ensuite, la clause $AXIOMS$ permet d'exprimer des prédicats sur les ensembles et les constantes, tandis que dans la clause $EVENTS$, on trouve des événements particuliers d'initialisation (clause *Initialisation*) ainsi que des événements caractéristiques du système.

▶ Evénement Event-B

Les événements en Event-B prennent trois formes représentées dans le tableau 3.4 où G(c,v), G(p,c,v) représentent les gardes des événements, et A(c,v,v') représente les actions générées par l'événement :

Evénement simple. La garde de l'événement est toujours vraie. C'est généralement l'écriture de l'événement particulier d'initialisation *Initialisation* qui n'est déclenchée qu'une seule fois (tableau 3.4-a).

Evénement gardé. L'écriture de l'événement fait apparaître une garde. L'événement se déclenche seulement lorsque la garde est satisfaite (tableau 3.4-b).

Evénement multiple. C'est un événement gardé particulier paramétré par une variable locale utile pour décrire des événements correspondants à une valeur de la variable locale dans un domaine spécifié dans la garde : $\exists p \cdot G(p, c, v)$ (tableau 3.4-c).

(a)	(b)	(c)
Evt $\widehat{=}$ **begin** A(c,v,v') **end**	*Evt* $\widehat{=}$ **when** G(c,v) **then** A(c,v,v') **end**	*Evt* $\widehat{=}$ **any** p **where** G(p,c,v) **then** A(p,c,v,v') **end**

TABLEAU 3.4 – Différents formes d'événements.

On notera que, lorsque la garde d'un événement est satisfaite, l'action associée n'est pas nécessairement déclenchée. Par ailleurs, si plusieurs événements d'un modèle ont leurs gardes satisfaites au même instant, leurs actions ne sont pas déclenchées simultanément : il y a un choix arbitraire d'un des événements. De plus, la durée d'exécution de l'action est considérée comme instantanée.

Enfin, un événement traduit un changement d'état gardé qui fait évoluer les valeurs d'un ensemble de variables. L'action $A(c, v, v')$ peut décrire :

- soit une action inchangée notée *skip* qui maintient toutes les variables à leurs mêmes valeurs : $v' = v$;
- soit une affectation entièrement déterministe de la forme $v := E(c, v)$ où $E(c, v)$ est une expression attribuée aux variables v après modification ;
- soit une affectation non déterministe (partiellement ou totalement) s'écrivant $v :\mid Q(c, v, v')$ où $Q(c, v, v')$ est un prédicat dont le modèle d'interprétation spécifie les valeurs possibles de v après modification.

▶ Obligation de preuves

Comment prouver que le modèle réalisé se comporte correctement et qu'il répond aux propriétés spécifiées ? Pour cela, une collection d'obligations de preuves est associée au formalisme Event-B lors de la modélisation d'un système et ces règles d'obligations de preuves doivent être déchargées afin de garantir la correction du modèle. Parmi ces règles, on trouve la règle de préservation des invariants (notée INV) (tableau 3.5) exprimant qu'un événement ne viole pas les invariants après modification des variables d'état, ainsi que la règle de fai-

sabilité (ou *feasibility rule*, notée FIS) (tableau 3.6) appliquée aux événements ayant des actions non-déterministes :

$P(c) \wedge I(c,v) \wedge G(c,v) \wedge A(c,v,v') \Rightarrow I(c,v')$	**INV**

TABLEAU 3.5 – Règle de préservation des invariants.

$P(c) \wedge I(c,v) \wedge G(c,v) \Rightarrow \exists v' \cdot A(c,v,v')$	**FIS**

TABLEAU 3.6 – Règle de faisabilité.

Notez que ces règles s'appliquent également aux événements d'initialisation. La règle de préservation des invariants et la règle de faisabilité sont exprimées respectivement dans la propriété (tableau 3.7), notée INI_INV et la propriété (tableau 3.8), notée INI_FIS.

$P(c) \wedge K(v) \Rightarrow I(c,v)$	**INI_INV**

TABLEAU 3.7 – Règle de préservation des invariants pour un événement d'initialisation.

De plus, il est parfois utile pour certains systèmes embarqués de prouver que le système ne bloque pas i.e. qu'il fonctionne en permanence. Il s'agit alors de montrer, qu'il existe toujours un événement déclenchable quel que soit l'état du système. Cela est obtenu en stipulant que la disjonction des gardes du modèle est toujours satisfaite sous l'hypothèse de satisfaction des axiomes et des invariants. Cette règle optionnelle s'appelle la règle de non blocage (ou *DeadLock Freedom rule*, notée DLF) (tableau 3.9).

Enfin, Event-B dispose d'un autre atout majeur : son principe de raffinement rigoureux permettant une modélisation formelle progressive et sûre pour traiter de la complexité d'un système.

3.2.2 Raffinement

En règle générale, on définit un raffinement comme une technique incrémentale visant à transformer un modèle abstrait en un modèle plus concret contenant plus de détails. Ces détails ajoutés peuvent concerner des variables, des invariants ou des événements [54], [55].

Définition 3.22 : Raffinement

Le raffinement étant une relation de préordre partielle monotone sur l'ensemble des modèles notée $\sqsubseteq$, il est défini de la manière suivante. Soient M_a et

$P(c) \Rightarrow \exists v \cdot K(v)$	**INI_FIS**

TABLEAU 3.8 – Règle de faisabilité pour un événement d'initialisation.

$P(c) \wedge I(c,v) \Rightarrow G_1(c,v) \vee \ldots \vee G_n(c,v)$	**DLF**

TABLEAU 3.9 – Règle de faisabilité pour un événement d'initialisation.

M_r deux machines. On dit que M_a est une abstraction de M_r ou que M_r raffine M_a si et seulement si pour tout événement e_a de M_a, il existe un événement e_r de M_r qui le raffine. En considérant deux événements $e_a \in G_a(c,v) \rightarrow A_a(c,v,v')$ et $e_r \in G_r(c,w) \rightarrow A_r(c,w,w')$ associés aux invariants sur les variables abstraites *I(c,v)* et aux invariants dits de "collage" entre les variables abstraites et les variables concrètes *J(c,v,w)*, une définition formelle de la construction $e_a \sqsubseteq e_r$ (signifiant que "e_a est raffiné par e_r") est spécifiée dans les propriétés établies dans le tableau 3.10. ■

$P(c) \wedge I(c,v) \wedge J(c,v,w) \wedge G_r(c,w) \Rightarrow G_a(c,v)$	**GRD_REF**
$P(c) \wedge I(c,v) \wedge J(c,v,w) \wedge G_r(c,w) \wedge A_r(c,w,w') \Rightarrow \exists v' \cdot (A_a(c,v,v') \wedge J(c,v',w'))$	**INV_REF**

TABLEAU 3.10 – Règles de pertinence d'un raffinement.

La propriété notée GRD_REF, correspond au renforcement de la garde abstraite par la garde concrète. Autrement dit, un événement concret ne pourrait être déclenché si la condition d'activation de son événement abstrait ne l'était pas. La propriété notée INV_REF correspond au cas général d'événements abstrait et concret non déterministes pour lesquels on spécifie que chaque modification des variables concrètes w par l'événement concret est associée à au moins une modification des variables abstraites v par l'événement abstrait correspondant. En d'autres termes, l'exécution de l'événement concret ne contredit pas l'événement abstrait, en validant les invariants de collage avant et après exécution. De plus, toutes les exécutions concrètes ont un pendant abstrait, ce qui n'est pas le cas inversement. On aboutit ainsi à une réduction du non déterminisme par le raffinement. En notant rel, l'ensemble des relations binaires entre les variables d'état avant exécution de l'événement et après exécution de l'événement, on obtient une transformation de la propriété en [3.1] :

$$rel(e_r) \subseteq rel(e_a) \tag{3.1}$$

Exemple

Considérons par exemple un raffinement permettant une réduction du non déterminisme d'un événement :

$$\begin{aligned} e_a &\hat{=} x :\in 0..2 \\ e_r &\hat{=} x := 1 \end{aligned}$$

Le non déterminisme de l'événement abstrait implique que tous les choix sont possibles : (x,0), (x,1) et (x,2). Donc, on établit bien un raffinement correct d'un événement car $rel(e_r) \subseteq rel(e_a)$, comme $\{(x,1)\} \subseteq \{(x,0),(x,1),(x,2)\}$.

Exemple

Soit un raffinement renforçant la garde d'un événement :

$$\begin{aligned} e_a &\hat{=} x = 0 \mid x := 1 \\ e_r &\hat{=} x = 0 \wedge y = 5 \mid x := 1 \end{aligned}$$

Le renforcement de la garde étant assuré par l'événement concret, on en déduit que le raffinement est correct.

Par conséquent, le raffinement sert souvent à réduire le non déterminisme d'un événement ou bien à renforcer la condition d'activation d'un événement. Le raffinement suggère que l'utilisation de M_r à la place de M_a est imperceptible du point de vue d'un observateur ou d'un utilisateur.

De même, dans le cas d'un raffinement, on étend les propriétés de préservation des invariants pour un événement d'initialisation INI_INV_REF et de non blocage relatif DLF_REF (tableau 3.11).

$P(c) \wedge K(v) \wedge N(w) \Rightarrow J(c,v,w)$	**INI_INV_REF**
$P(c) \wedge I(c,v) \wedge J(c,v,w) \wedge (G_{a1}(c,v) \vee \ldots \vee G_{an}(c,v)) \Rightarrow (G_{r1}(c,w) \vee \ldots \vee G_{rn}(c,w))$	**DLF_REF**

TABLEAU 3.11 – Autres obligations de preuves d'un raffinement.

Dans un modèle raffiné, l'ajout d'un nouvel événement raffine l'événement *skip* dans le modèle abstrait. Pour ce nouvel événement, la règle de préservation des invariants est encore valable pour justifier que cet événement est correct. La règle de non blocage relatif impose que si le modèle abstrait est non bloqué alors le non blocage du modèle raffiné implique que les nouveaux événements ne sont pas activables indéfiniment (ce qui correspond à des événements *skip* activés à l'infini dans le modèle abstrait ; ce qui est contradictoire avec le non blocage du modèle abstrait). Pour éviter la divergence des nouveaux événements, on établit un variant pour tous les nouveaux événements et on s'assure de la décroissance de ce variant à chaque activation d'un nouvel événement (tableau 3.12).

$P(c) \wedge I(c,v) \wedge J(c,v,w) \wedge G_r(c,w) \Rightarrow Var(c,w) \in \mathbb{N}$	**VAR_REF1**
$P(c) \wedge I(c,v) \wedge J(c,v,w) \wedge G_r(c,w) \Rightarrow Var(c,w') < Var(c,w)$	**DLF_REF**

TABLEAU 3.12 – Règle de non divergence des nouveaux événements.

Maintenant, revenons quelques instants sur le principe de vérification et de validation (V & V) mis en place dans la méthode Event-B et basé sur les règles d'obligation de preuves, afin de comprendre notre intérêt pour cette méthode et notre utilisation par la suite.

3.2.3 Positionnement de Event-B par rapport à la logique de Hoare

On a vu dans la sous-section précédente les rudiments de Event-B tirés en grande partie de la méthode B dédiée aux développements logiciels. De fait, le principe de V & V de Event-B est basé sur des méthodes et techniques développées en génie logiciel. Partant de ce constat qu'un programme implanté dans une machine d'exécution peut être très complexe et critique, il est évident de définir des règles strictes garantissant la fiabilité et la sécurité du programme et par extension, du système [50]. Les principales règles de V & V sont issues de la logique de Hoare.

Ainsi, [50] explique comment le langage formel B respecte la logique de Hoare et facilite la construction de modèles corrects. Par extension, Event-B apporte les mêmes garanties pour une modélisation correcte et pertinente sous l'hypothèse du choix des bonnes propriétés à vérifier. Le principe de la logique de Hoare donne lieu à différentes règles d'obligation de preuves à respecter en Event-B lors de la construction d'un modèle. La première de ces règles est la règle de préservation des invariants. Un invariant étant une propriété satisfaite avant et après exécution de tout événement d'un modèle, il peut être soit une propriété de typage des variables d'état, soit une propriété spécifiée que l'on souhaite voir respectée en permanence. D'après l'axiome *[Aff]*, si l'ensemble des invariants I est une pré condition d'une substitution S constituant l'action d'un événement, étant donné que $S\{I_i\}$ est la plus faible pré condition pour un invariant I_i donné en post condition, on a alors : $I \Rightarrow S\{I_i\}$. En étendant cette formule à un modèle avec une substitution gardée (la garde est notée *G(c,v)* où v représente l'ensemble des variables d'état) dans un environnement contraint défini par des propriétés sur les constantes c (en Event-B ces propriétés sont appelées des axiomes P - voir section 3.2), on obtient la formule généralisée : $P(c) \wedge I(c,v) \wedge G(c,v) \Rightarrow S\{I_i(c,v)\}$. En écrivant le prédicat avant-après sous la forme $v' = E(c,v)$ avec v' les variables modifiées par l'événement, on a : $P(c) \wedge I(c,v) \wedge G(c,v) \Rightarrow I_i(c, E(c,v))$. On retrouve ainsi la règle de préservation

d'invariants présentée dans la section 3.2.1 où la substitution est sous forme de prédicat *A(c,v,v')*. Dans le cas d'une initialisation par affectation de valeurs $v' = K(c)$, cette formule se réduit à : $P(c) \wedge I(c, v) \Rightarrow I_i(c, K(c))$. La seconde règle d'obligation de preuves concerne les événements provoquant une modification d'état non déterministe $v :| A(c, v, v')$ qui doivent être faisables et non inertes. Cette règle de faisabilité (voir section 3.2.1) indique que dans l'hypothèse où la pré condition de l'événement non déterministe est remplie, il existe une valeur possible que l'on peut affectée à v.

D'autres règles d'obligation de preuves notamment en rapport avec le raffinement en Event-B renforcent les mécanismes de vérification de la correction et de la pertinence de la modélisation.

3.3 Vérification et validation

La stratégie de vérification de modèles Event-B repose sur une double démarche :

- l'une plus factuelle et moins outillée consiste à corroborer les modèles développés avec des modèles éprouvés existants ou des avis d'experts ;
- l'autre s'appuie sur la validation des règles d'obligation de preuves déchargées par les prouveurs de la plateforme Rodin [2].

Le premier point est une méthode courante en modélisation (figure 3.1). Il tend à enrichir la base des connaissances a priori sur le système étudié. Le regard du concepteur sur ces connaissances est essentiel dans un premier temps pour y puiser des informations pertinentes. Puis, le concepteur synthétise un modèle permettant de compléter ou de modifier les informations initiales. Cette analyse aboutissant à des modifications éventuelles est itérative et récursive, et elle est menée en permanence en tâche de fond lors de la modélisation. Des techniques et des outils servant à juger la pertinence des informations récoltées existent pour aider le concepteur dans cette démarche. Cependant, l'objet de notre étude ne portant pas sur ces techniques, notre démarche a consisté à réaliser par itération plusieurs modèles Event-B à partir d'un cahier des charges et de modèles fonctionnels existants en Simulink. L'un de ces modèles, celui présenté dans ce manuscrit, a été retenu comme il semble le plus pertinent (voir chapitre 4).

La seconde démarche de vérification est plus rigoureuse car elle s'appuie sur les concepts de la modélisation Event-B et sur les performances de validation de la plateforme Rodin (figure 3.2). La plateforme Rodin est en quelques mots un support libre de la méthode Event-B développé par des universitaires et des industriels en Europe. La plateforme Rodin intègre de nombreux outils interfacés dont des prouveurs de théorèmes permettant de valider les règles d'obligation de preuves intrinsèques à la modélisation. Certaines règles sont déchargées au-

[2] `http ://www.event-b.org/`

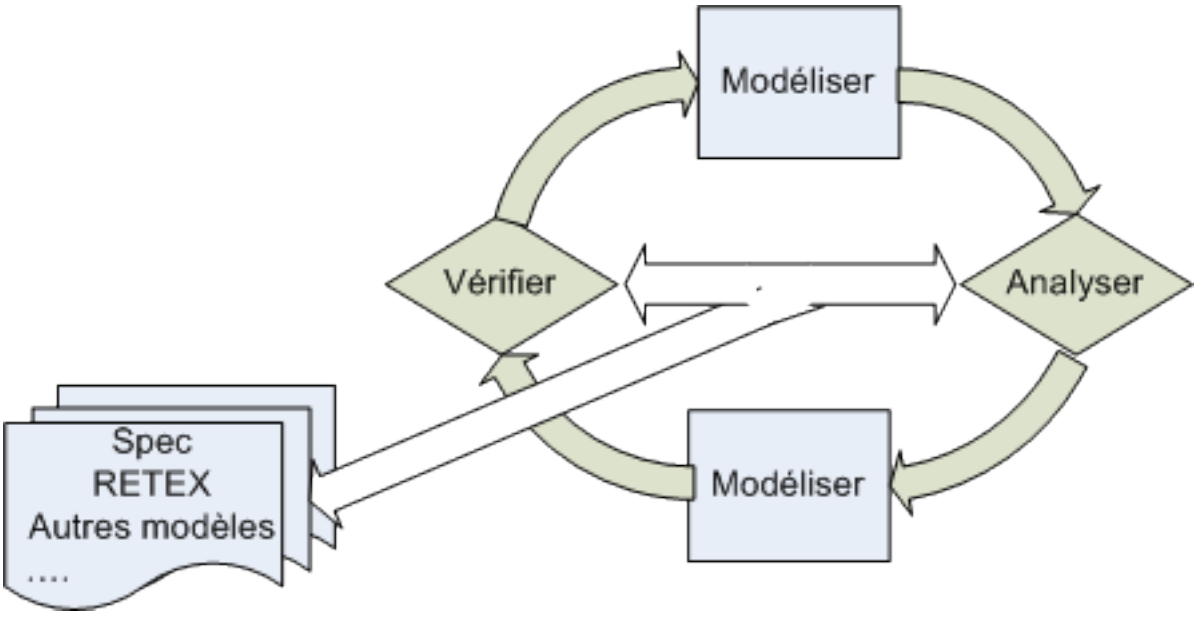

FIG. 3.1 – Méthodologie de modélisation.

tomatiquement lors de l'enregistrement d'un modèle. D'autres nécessitent l'utilisation d'un environnement permettant de réaliser des preuves interactives. Les décharges de preuves interactives peuvent être plus ou moins longues et rébarbatives. Pour diminuer le nombre de ces preuves interactives et alléger la charge du concepteur en vérification, il convient d'utiliser à bon escient les mécanismes de décomposition et de raffinement mis en place en Event-B. En effet, ces mécanismes réduisent la complexité des modèles et mettent à disposition des règles à décharger les hypothèses ou les propriétés utiles pour leurs preuves. Il s'agit alors de rendre les modèles compréhensibles et de faciliter l'analyse de sécurité, par exemple.

Par la suite, notre stratégie a consisté à imbriquer ces deux démarches au niveau des processus de modélisation et de validation de notre cas d'étude. La première démarche intervient principalement au niveau de la compréhension du fonctionnement du système étudié. La seconde démarche est surtout mise en œuvre pour prendre en compte les exigences de sécurité à respecter dans les modèles développés. D'un point de vue didactique, la compréhension du système signifie savoir si celui-ci tel qu'il est envisagé ou connu a priori répond aux spécifications du cahier des charges par rapport à ses caractéristiques fonctionnelles. Le schéma de la figure 3.1 représente la première démarche de vérification permettant de porter un jugement sur nos modèles par rapport au comportement attendu.

Par conséquent, l'apport principal de la stratégie de modélisation et de validation adoptée réside dans la confiance accordée aux modèles développés

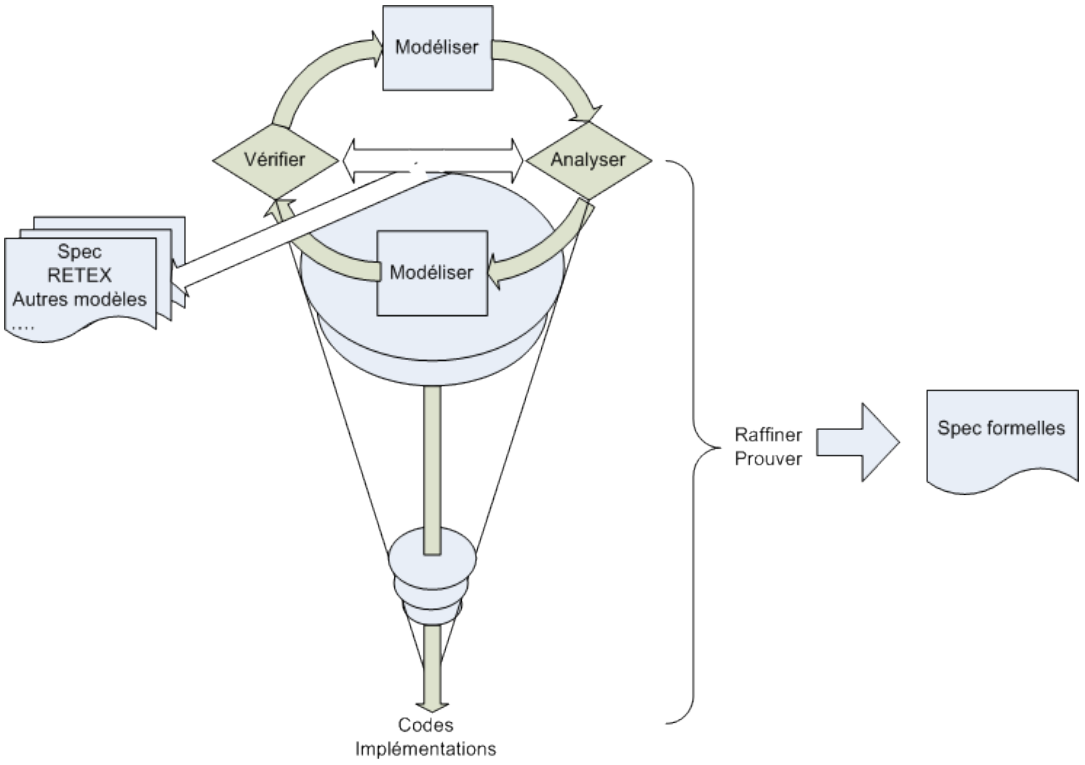

FIG. 3.2 – Méthodologie de modélisation en Event-B.

pour bien mener l'analyse de sécurité. Dans le chapitre 4 suivant, on verra plus en détails les modèles développés ainsi que les vérifications effectuées.

Chapitre 4

La modélisation de l'architecture en couches

Résumé
La modélisation en Event-B fournit une représentation abstraite du comportement d'un système sous la forme d'un ensemble d'objets et d'événements caractéristiques du système et des propriétés que ces éléments doivent respecter. Les preuves réalisées sur ce type de modèle visent à s'assurer que le comportement est correct vis-à-vis des propriétés. Dans le cas de notre étude, ce principe est appliqué aux différentes couches proposées pour structurer l'architecture des systèmes autonomes. Le raffinement est utilisé à la fois verticalement pour établir les relations entre couches, et horizontalement pour préciser le contenu de chaque couche du point de vue de la sécurité.

« L'abstraction, c'est la capacité à accepter de ne pas comprendre tout de suite pour comprendre après. »
David Constant

Le paradigme de l'observation d'une ville en parachute illustre bien la notion d'*abstraction*. La première vue depuis le point de largage, dessine le contour du panorama en mettant en évidence des bâtiments, des routes et des véhicules circulant sur ces routes. A mi-parcours, on aperçoit des individus entrant et sortant des bâtiments et des véhicules. Enfin, proche du sol, les détails sont plus importants, et on arrive à mieux distinguer les accès aux bâtiments, les feux de signalisation sur les routes, les feux des véhicules, les interactions entre les individus. En se rapprochant, on voit donc plus d'événements et le comportement des différentes entités est plus précis. Cependant les vues rapprochées ne contredisent pas les vues lointaines qui offrent une image grossière du site. Par conséquent, il ressort de ce paradigme qu'une première difficulté de la représentation concerne l'estimation de la *granularité* de l'abstraction finale afin de ne pas être submergé par les détails de ce que l'on observe.

Par une approche système, la modélisation en Event-B cherche à spécifier la représentation juste nécessaire du système. La modélisation en Event-B s'abstient d'expliciter une (ou la) solution à un problème. Elle vise plutôt à s'assurer que les réalisations tirées des modèles développés seront jugées correctes grâce aux propriétés validées.

Ainsi, notre architecture en couches du système de contrôle a subi un processus de modélisation *itérative* et *progressive* en Event-B. On a mis en œuvre deux types de raffinement [56] :

- un raffinement "horizontal" qui apporte des détails à l'intérieur des couches ;
- un raffinement "vertical" qui concerne des détails sur les relations inter couches.

Dans ce chapitre, on présente d'abord l'approche de modélisation adoptée (section 4.1). Puis, on détaille la modélisation de l'architecture en décrivant les caractéristiques et les propriétés couche par couche (section 4.2). Enfin, on évoque comment les propriétés de sécurité sont prouvées dans notre modélisation (section 4.6).

4.1 Approche relative à l'architecture

L'intérêt de Event-B dans notre étude réside dans sa modélisation permettant d'exprimer formellement des propriétés validées par preuves pendant la conception des modèles du système, mais également dans son principe de raffinement que l'on utilise d'une part pour formaliser les relations entre couches de l'architecture, et d'autre part pour affiner progressivement les définitions des couches.

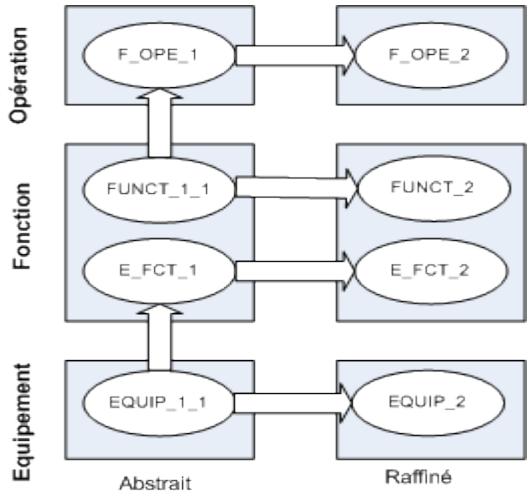

FIG. 4.1 – Mécanisme de raffinement adopté.

Dans notre approche, on considère que les propriétés définies précédemment (voir sous-section 2.3.3) sont primordiales pour traduire les relations entre les entités des couches de l'architecture. On a vu que parmi ces relations figurent des "services" génériques fournis par une couche inférieure, coordonnés par la couche supérieure pour réaliser une fonction spécifique. Du point de vue de la modélisation Event-B, la notion de "services" suggère alors un raffinement dont la machine abstraite est le modèle de la couche de niveau inférieur. En effet, dans une relation entre deux couches adjacentes, la couche de niveau inférieur offre un degré de liberté plus grand du point de vue fonctionnel, que seul le comportement de la couche de niveau supérieur permet de contraindre par raffinement. Ce raffinement se poursuit jusqu'à obtenir une implémentation correcte de la relation entre ces couches, vue de la couche de niveau supérieur. Ainsi, les hypothèses faites au niveau fonction contraignent l'activité des équipements ; de même les conditions ou propriétés à satisfaire au niveau opération permettent de raffiner l'exécution des fonctions. La figure 4.1 met en évidence l'approche de modélisation Event-B adoptée pour spécifier les relations entre les couches. Les modèles *EQUIP_1_1* et *FUNCT_1_1* subissent un premier raffinement dit "vertical" précisant les relations entre couches (*E_FCT_1*, *F_OPE_1*), puis ces derniers sont à leur tour raffinés plus classiquement pour renforcer ces relations. Les raffinements classiques "horizontaux" sont poursuivis jusqu'à considérer, par exemple, le cas concret du système de contrôle du drone autonome RMAX. De plus, l'application d'un mécanisme de fusion de modèles (fusion implicite entre les modèles *FUNCT_1* et *E_FCT_1*) permet de consolider la spécification de notre architecture en couches comme indiqué par Abrial [57] et Butler [58].

Ce chapitre du mémoire présente une formalisation en Event-B de l'architecture générique en couches d'un système de contrôle en décrivant le comportement des couches équipement, fonction et opération. Les principes de l'approche adoptée s'appuient sur un processus de modélisation basé sur des raffinements

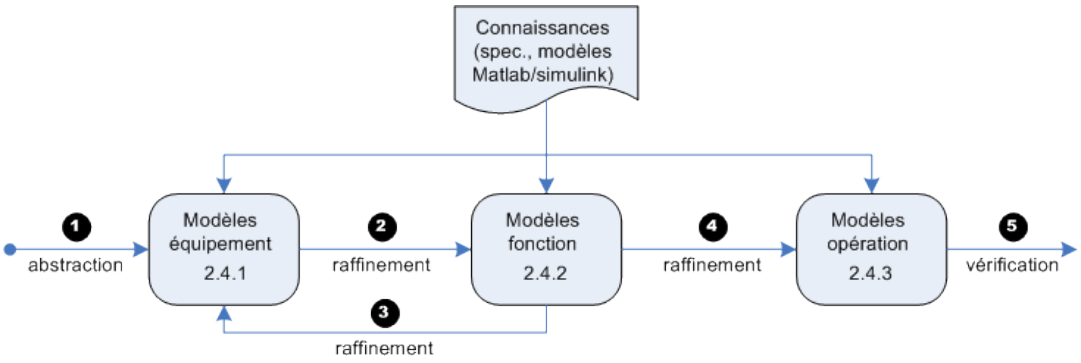

FIG. 4.2 – Principes de modélisation.

successifs (figure 4.2). La première étape (figure 4.2 ❶) concerne une focalisation du concepteur sur des aspects essentiels du système représentant son comportement de base par rapport à ses interactions avec l'environnement (des mesures et des actions physiques vis-à-vis du milieu environnant). On réalise alors une abstraction du fonctionnement du système. L'étape suivante (figure 4.2 ❷) traduit une amélioration de cette première modélisation par une association du comportement des équipements à des fonctions permettant de leur donner un sens. La troisième étape (figure 4.2 ❸) complète l'interaction équipement-fonction en acceptant d'agir sur la configuration des équipements à la suite d'analyses effectuées au niveau fonction. Cela contribue à favoriser l'accomplissement des objectifs de sécurité. Du point de vue de la modélisation, l'étape quatre (figure 4.2 ❹) conclut par la prise en compte globale du comportement nominal ou défectueux pour un scénario de mission. Enfin, la dernière étape de vérification (figure 4.2 ❺) fait l'objet de la section 4.6.

4.2 Caractéristiques globales des modèles

4.2.1 Description de l'architecture

Une description générique de l'architecture met en évidence des familles de composants qui sont étroitement liées à la définition des couches. Elles sont nommées *EQUIPMENT*, *FUNCTION* et *OPERATION* et sont caractérisées par les attributs suivants :

- Le *mode de fonctionnement* permet d'identifier pour un composant donné l'état de fonctionnement courant. Suivant la famille de composant, on désigne soit l'ensemble des modes de fonctionnement d'un équipement (*E_MODE*), soit l'ensemble des modes de fonctionnement d'une fonction (*F_MODE*), soit l'ensemble des modes opération-

nels (*O_MODE*). Par exemple, les ensembles *E_MODE*, *F_MODE* et *O_MODE* pourraient contenir respectivement des éléments de la forme "id_nom-équipement_mode", "id_nom-fonction_mode" et "id_nom-opération_mode".

- *L'état de santé* caractérise une surveillance de la qualité du fonctionnement. De même, on désigne deux ensembles, à savoir, l'ensemble des états de santé associé aux équipements (*E_SURVEILLANCE*) et l'ensemble des états de santé associé aux fonctions (*F_SURVEILLANCE*). La couche opération étant la couche ultime de supervision du système, on ne dispose plus de moyens de récupérer une défaillance et on fait donc l'économie de la modélisation de son état de santé. De même, les ensembles *E_SURVEILLANCE* et *F_SURVEILLANCE* pourraient contenir respectivement des éléments de la forme "id_nom-équipement_statut" et "id_nom-fonction_statut".
- Les *ressources d'observation* permettent d'identifier les données produites ou consommées par certains composants. Parmi l'ensemble des ressources d'observation, on distingue un ensemble d'observations attribuées aux équipements (*E_OBSERVATION*) et un ensemble d'observations attribuées aux fonctions (*F_OBSERVATION*). De même, on suppose dans notre modélisation qu'il n'y a pas d'observations spécifiques aux opérations. Les informations éventuellement intéressantes sont implicitement suggérées dans le mode opérationnel. Par exemple, les ensembles *E_OBSERVATION* et *F_OBSERVATION* pourraient contenir respectivement des éléments de la forme "id_nom-équipement_obs" et "id_nom-fonction_obs".

Par ailleurs, des domaines de valeurs sont associés à ces attributs et sont différenciés suivant les familles de composants (les attributs et les domaines de valeurs associés sont présentés dans le tableau 4.1 récapitulatif).

Concernant l'attribut de mode de fonctionnement, on décrit un ensemble énuméré d'états fonctionnels distincts pour des équipements, noté *E_FLAG* :

active	état signifiant que l'équipement est actif (par exemple, un actionneur réalise un mouvement ou un capteur envoie des mesures) ;
off	état "off" d'un équipement ; cet état caractérise principalement des mises hors service suite à des défaillances ;
idle	état signifiant que l'équipement est en attente (pour un actionneur, l'attente d'une nouvelle commande) ;
spare	état d'un équipement dit "de secours" ; on considère ici des redondances tièdes pour lesquelles l'équipement "de secours" n'est pas actif mais est alimenté.

De même, on décrit un ensemble énuméré d'états fonctionnels distincts caractérisant des fonctions, noté *F_FLAG* :

		Attribut		
		Mode de fonctionnement	**Etat de santé**	**Ressources d'observation**
EQUIPMENT	nom	E_MODE	E_SURVEILLANCE	E_OBSERVATION
	valeurs	E_FLAG	E_STATUS	VALUE BOOL
FUNCTION	nom	F_MODE	F_SURVEILLANCE	F_OBSERVATION
	valeurs	F_FLAG	F_STATUS	VALUE BOOL
OPERATION	nom	O_MODE		
	valeurs	MODE_OP		

TABLEAU 4.1 – Caractéristiques statiques de l'architecture en couches étudiée

f_active	état signifiant que la fonction est activée par une requête ; dans cet état la fonction réalise des acquisitions de ressources ou pas ;
f_idle	état signifiant que la fonction est en attente d'une nouvelle requête d'activation ;
f_executing	état de la fonction en cours d'exécution ;
f_off	état de la fonction hors service.

On stipule également un ensemble énuméré de modes opérationnels distincts, noté *MODE_OP* :

m_ok	correspondant au mode opérationnel nominal ;
m_backup	ce mode caractérise une phase de repli en mode dégradé ;
m_aborted	représente l'interruption d'une phase suite à une défaillance et la poursuite de la mission initiale ;
m_cancelled	arrêt de la mission symbolisant une perte de l'engin.

En ce qui concerne l'attribut d'état de santé, on représente un ensemble énuméré de statuts de surveillance pour des équipements, noté *E_STATUS* :

ok	l'état de santé de l'équipement est nominal ;
erroneous	l'équipement renvoie des valeurs erronées dans le domaine des valeurs spécifiées ;
lost	l'équipement ne renvoie pas de valeurs ou renvoie des valeurs hors du domaine des valeurs spécifiées.

De même, on définit un ensemble énuméré de statuts de surveillance pour des fonctions, noté *F_STATUS* :

f_ok	l'état de santé est nominal ;
f_erroneous	la fonction renvoie des valeurs erronées dans le domaine des valeurs spécifiées ;
f_lost	la fonction ne renvoie pas de valeurs ou renvoie des valeurs hors du domaine des valeurs spécifiées.

Enfin, l'attribut de ressources d'observation est évalué par un ensemble abstrait de valeurs, appelé simplement *VALUE*. On associe également à cet attribut un ensemble de booléens indiquant la disponibilité des données d'observation.

4.2.2 Description du comportement nominal

Les paramètres caractéristiques de l'architecture évoluent après exécution d'activités ou de services des différentes couches. Plus concrètement, on considère que l'exécution d'un service peut être constituée des 4 types d'évènements suivants :

- *L'activation* permet de sélectionner une fonctionnalité associée à un composant d'une couche. Par exemple, l'événement *a_activate* représente

l'activation d'un actionneur dans la couche équipement (la sortie d'un état de veille). De même, l'événement *select_ab* correspond à l'activation d'un mode avorté (*Abort*) dans la couche opération (idem pour les modes normal (*ok*), dégradé (*bu*), annulé (*ca*)).
- *L'acquisition* renforce les liens entre différents composants de l'architecture par l'attente de ressources, essentiellement des données utiles à l'étape suivante (*f_acquire_r*). Par contre, l'événement *f_acquire* indique seulement le passage d'une activation vers une étape d'exécution.
- *L'exécution* correspond au traitement lié à la fonction du composant. Elle peut engendrer des données à échanger comme l'envoi de mesures par un capteur de la couche équipement (*s_send*) ou l'envoi des résultats d'une action réalisée par un actionneur (*a_return*). Dans la couche fonction, ce type d'événements produit ou non des ressources (*f_execute_r*, *f_execute*). De plus, les événements *send_cmd_t* et *send_cmd_f* traduisent la génération et l'envoi de commandes (qui sont des observations communes aux équipements et aux fonctions) destinées aux actionneurs. De façon similaire, dans la couche opération, les événements *send_rq_t* et *send_rq_f* traduisent la génération et l'envoi de requêtes (qui sont des observations communes aux opérations et aux fonctions) destinées à des fonctions.

Suivant la couche considérée, ces types d'événements sont plus ou moins représentés. De plus, un autre type d'événements se rapporte à la communication entre les couches en traduisant la disponibilité des données ou des observations à échanger et le protocole d'échange. Le tableau 4.2 donne un aperçu des événements élaborés pour symboliser le comportement nominal du système dans une architecture en couches.

4.2.3 Description de comportements fautifs

Le comportement en présence de fautes se manifeste par des modifications du statut de santé des différents composants de l'architecture. La variable d'état qui permet cette surveillance est une fonction *e_status* pour la couche équipement, ou bien *f_status* pour la couche fonction. L'évolution des valeurs des statuts de santé d'un instant à un autre se traduit par des événements de défaillance associés au mécanisme de détection de défauts :
- *detect_err*[1] considère le passage à l'état erroné suite à la détection d'un défaut conduisant à l'envoi de valeurs erronées ;
- *detect_lost*[1] exprime la perte d'un composant sous l'effet d'un défaut.

Un autre mécanisme lié à la reconfiguration due à une faute est représenté par un événement *recover* qui dans le cas des équipements engendre l'activation d'un équipement redondant et met hors service l'équipement perdu en veillant à

[1]Un préfixe *f_* est ajouté pour distinguer des événements de détection lié à la couche fonction

		Classes d'évènements nominaux			
		Activation	**Acquisition**	**Exécution**	**Communication**
EQUIPMENT	actuator	a_activate		a_return	enable_obs
	sensor			s_send	
FUNCTION	function	f_activate	f_acquire f_acquire_r	f_execute f_execute_r send_cmd_t send_cmd_f	receive_obs op_enable_obs
OPERATION	mode	select_ok select_ab select_bu select_ca		send_rq_t send_rq_f	op_receive_obs

TABLEAU 4.2 – Événements pour le comportement nominal

diminuer le nombre d'équipements redondants restant. Dans la couche fonction, un événement *f_recover* met hors service la fonction défaillante et diminue le nombre de fonctions équivalentes à cette fonction défaillante. Par ailleurs, deux événements particuliers traitent de reconfiguration par rapport à des défaillances dans des échanges de données. Le premier événement *frecover_f* concerne des échanges de données erronées entre fonctions pour lesquelles on choisit d'interrompre les échanges en rendant indisponible l'accès à ces données. Le second événement *frecover_e* concerne des échanges de données erronées (évaluation négative de ces données) provenant d'un équipement pour lequel on choisit d'envoyer une commande faisant passer l'état de cet équipement incriminé dans l'état erroné bien qu'aucune faute n'ait été identifiée. Ce changement effectif de l'état de l'équipement est réalisé par l'événement *recover_err*, dans la couche équipement.

Le tableau 4.3 récapitule les événements modélisés pour un comportement en présence de fautes dans notre architecture en couches.

4.3 Modélisation de la couche Equipement

Précisions à présent comment ces principes de modélisation généraux sont spécialisés par couche.

	Classes d'évènements de gestion des défaillances	
	Détection	**Reconfiguration**
EQUIPMENT	detect_err detect_lost	recover recover_err
FUNCTION	f_detect_err f_detect_lost propagate_err propagate_lost	f_recover frecover_f frecover_e
OPERATION		

TABLEAU 4.3 – Événements pour le comportement défectueux

Comme énoncé précédemment, le rôle de la couche équipement est de fournir des services à la couche fonction. Ces services correspondent à des activités liées à un fonctionnement nominal ou à un dysfonctionnement. Dans ce qui suit, on considère le modèle équipement effectuant la liaison avec les fonctions comme un modèle abstrait de départ. Dans la pratique, d'autres modèles plus abstraits ont été développés pour faciliter notre modélisation. Les sous-sections suivantes décrivent les hypothèses de cette modélisation, le processus mis en œuvre et les modèles réalisés.

4.3.1 Hypothèses de modélisation

La couche "équipement" se focalise sur la modélisation des fonctions liant le système autonome à son environnement. Il s'agit principalement des fonctions de :

- acquisition de grandeurs physiques ; ces fonctions sont généralement remplies par des capteurs ;
- action sur l'environnement ; ces fonctions sont généralement remplies par des actionneurs.

Elle comprend aussi les fonctions permettant de lier la couche "équipement" à la couche supérieure i.e. des fonctions de communication.

Par conséquent, pour une abstraction du fonctionnement du système par rapport aux interactions avec son environnement, on suppose certaines restrictions :

- on ne considère dans les équipements que les capteurs et les actionneurs ; les calculateurs sont implicitement évoqués lors de la description des fonctions : des redondances fonctionnelles avec dissimilarité pour

plus d'indépendance, imposent des implémentations sur des calculateurs distincts ;
- on ne prend pas en compte les effets de défauts d'alimentation qui peuvent constituer des situations critiques non négligeables en termes de probabilités menant à l'événement redouté ;
- on exclut également les autres équipements matériels annexes n'intervenant pas directement dans la réalisation du contrôle de l'engin : par exemple, des relais, des équipements et des réseaux de communication, la charge utile.

4.3.2 Processus de modélisation

D'après l'approche de modélisation énoncée plus haut, la modélisation des équipements constitue le point de départ de notre formalisation. On commence par un modèle abstrait, très simple, des capteurs et des actionneurs qui permet de poser les grands ensembles abstraits et les séquences d'états fonctionnels : activation des actionneurs, exécution des capteurs et des actionneurs, et communications amont-aval (section 4.2.2).

Ainsi, les capteurs et actionneurs produisent des données utiles pour des traitements ultérieurs en aval. Ces données concernent principalement des mesures dans le cas des capteurs, et des indications sur les effets des actions réalisées par des actionneurs.

Puis, des raffinements [2] répétés conduisent à une description plus détaillée du comportement non nominal des équipements en décrivant les mécanismes de FDIR locaux, à savoir les détections et les reconfigurations, propres à cette couche. Bon nombre des raffinements servent principalement à faciliter les décharges de règles d'obligations de preuves (voir section 4.6 suivante). Ces raffinements correspondent aux raffinements horizontaux mentionnés plus haut.

La figure 4.4 traduit les étapes de la modélisation des équipements avec :

1. la production des données utiles pour le contrôle qui est modélisée avec $EQUIP\_0$;
2. l'introduction de la communication avec la couche fonction qui passe par la définition de l'événement *enable_obs* dans $EQUIP\_1$;
3. l'intégration des mécanismes de sécurité avec les événements *detect_err*, *detect_lost*, *recover* dans $EQUIP\_1\_1$; ce modèle constitue le modèle complet des équipements qui sera détaillé par la suite ;
4. la précision des mécanismes de sécurité avec $EQUIP\_2$;
5. le raffinement de la reconfiguration qui est détaillée avec l'événement *recover* dans $EQUIP\_3$.

[2] Dans les raffinements, les événements non modifiés ne sont pas représentés volontairement

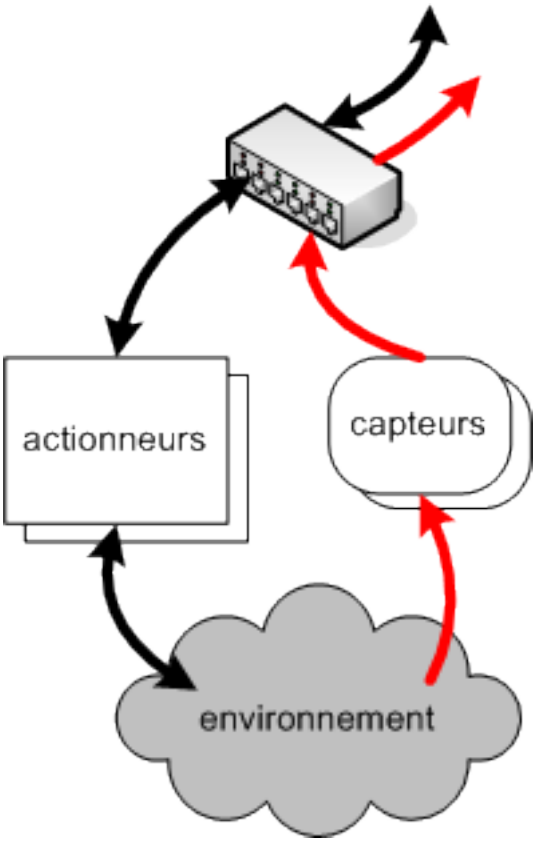

FIG. 4.3 – Principes des capteurs et actionneurs.

La sous-section suivante donne plus de détails sur les étapes ci-dessus.

4.3.3 Caractérisation des équipements

▶ Caractéristiques statiques

Le "Contexte" du modèle abstrait depuis $EQUIP\_0$ jusqu'à $EQUIP\_1\_1$ décrit les paramètres statiques liés à la couche équipement. Il définit ainsi des ensembles d'objets abstraits et des relations entre certains éléments de ces ensembles, qui sont des caractéristiques statiques de la couche considérée [3]. Par exemple :

- On définit un ensemble abstrait *EQUIPMENT* composé des sous-ensembles disjoints *SENSOR* et *ACTUATOR*.
- On identifie un ensemble abstrait global de ressources noté *RESOURCE* comprenant un ensemble d'attributs d'observations attribuées aux équipements ($E\_OBSERVATION$).

[3] Les modèles de "contexte" et de "machines" développés sont donnés en annexe A.3

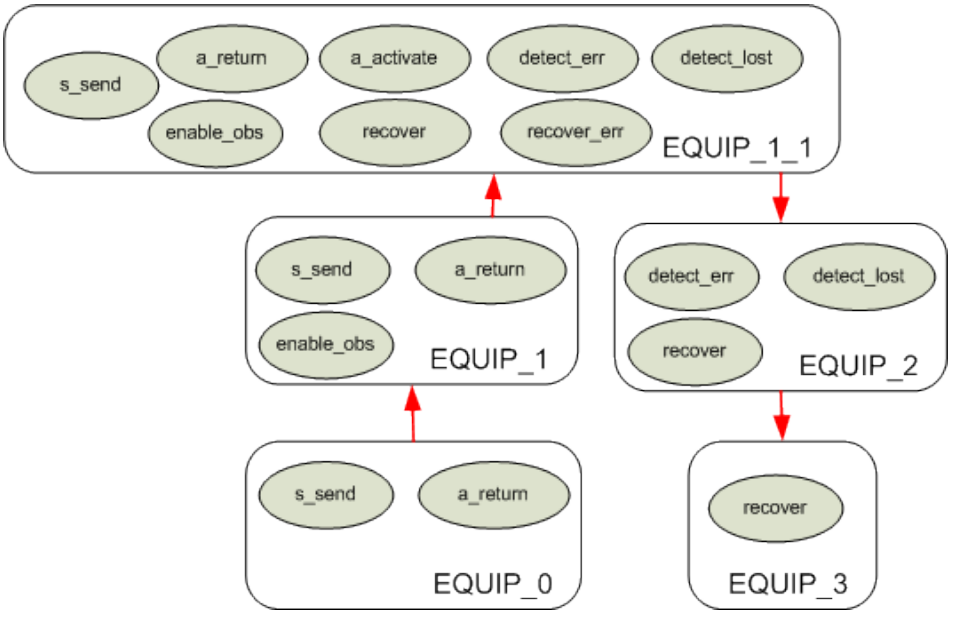

FIG. 4.4 – Etapes de la modélisation des équipements.

- Un ensemble de valeurs $VALUE$ est déclaré afin d'être associé aux observations produites lors de la description du comportement des équipements.
- On définit des ensembles abstraits E_MODE et $E\_SURVEILLANCE$ représentant respectivement des attributs décrivant l'état de fonctionnement et l'état de santé des équipements.
- De plus, un ensemble abstrait $DEFAULT$ caractérise l'ensemble des fautes que peut subir le système.
- On décrit un ensemble énuméré E_FLAG caractérisant des états fonctionnels distincts (section 4.2.1).
- On décrit un ensemble énuméré E_STATUS représentent les valeurs des attributs de surveillance de chaque équipement (section 4.2.1).

De même, des fonctions mathématiques représentent les liens entre les éléments des ensembles ci-dessus.

- La fonction bijective $eqt\_surv \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow E\_SURVEILLANCE$ [4] exprime que tous les équipements possèdent un attribut de surveillance dont la valeur donne leur état de santé.
- La fonction bijective $eqt\_mode \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow E\_MODE$ stipule que tous les équipements possèdent un attribut dont la valeur donne leur état de fonctionnement.
- La fonction bijective $eqt\_obs \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow E\_OBSERVATION$ exprime que tous les équipements possèdent un

[4] $\rightarrowtail\!\!\!\!\!\rightarrow$ est l'opérateur définissant une fonction bijective

attribut d'observation dont la valeur est un vecteur d'observables du système.

- La fonction partielle $e\_fault\_class \in DEFAULT \nrightarrow EQUIPMENT$ [5] identifie les défauts attribuables à chaque équipement. Certaines fautes peuvent affecter les couches supérieures de l'architecture du système.
- La fonction totale $eqt\_cat \in EQUIPMENT \rightarrow SS\_EQUIPMENT$ [6] indique que chaque équipement appartient à une catégorie d'équipements équivalents *SS_EQUIPMENT* contenant soit l'équipement seul soit plusieurs équipements redondants.

Propriétés Les propriétés relatives aux caractéristiques statiques sont principalement des axiomes de typage et de relation entre éléments d'ensembles abstraits tels que les déclarations de fonctions bijectives précédentes.

▶ Caractéristiques dynamiques

La "Machine" *EQUIP_1_1* de ce modèle décrit les caractéristiques dynamiques intégrales de la couche *équipement*. On y décrit trois événements traduisant le fonctionnement nominal des équipements, un événement concernant les échanges avec la couche supérieure et quatre événements représentant des situations de défaillance.

Activation

L'événement *a_activate* précise simplement l'activation d'un actionneur. On a comme paramètre un actionneur *act*. L'actionneur est activé à condition qu'il soit préalablement dans un état d'attente et dans un état nominal ou erroné.

Exécution

L'événement *a_return* indique le retour d'un actionneur *act* à un régime permanent du point de vue de l'automatique après une action ou un mouvement commandé, accompagné d'un indicateur *v*. Cet événement est donc consécutif à un événement d'activation d'un actionneur. On a comme paramètres un actionneur *act* et une valeur indicatrice *v*. Ce retour est effectif à condition que l'actionneur soit préalablement dans un état actif, et que l'actionneur soit dans un état nominal ou erroné. De même, l'état d'un buffer d'observations *buffer_obs* passe à *TRUE* signalant ainsi une mise à disposition des observations de cet actionneur.

[5] $\nrightarrow$ est l'opérateur définissant une fonction partielle
[6] $\rightarrow$ est l'opérateur définissant une fonction totale

L'événement *s_send* spécifie l'envoi d'une mesure par un capteur, indépendamment des sollicitations d'une quelconque fonction. On considère en paramètres un capteur *sen* et une valeur de mesure v donnés. La mise à jour de la mesure du capteur dépend alors en permanence des états de fonctionnement et de dysfonctionnement de l'équipement : le capteur est actif dans le sens où il peut mesurer des grandeurs physiques, mais il peut être dans un état nominal ou erroné. En outre, l'état d'un buffer d'observations $buffer\_obs$ correspondant à un protocole d'échanges entre les équipements et les fonctions du niveau supérieur, passe à la valeur *TRUE* pour une mise à disposition des observations de ce capteur lors de cet événement. Le terme *buffer* sous-entend également un éventuel besoin de stockage de données lié aux aspects temporels entre les couches.

Communication

L'événement *enable_obs* permet de préparer un buffer pour de prochaines observations (passage de *TRUE* à *FALSE*).

Détection

L'événement *detect_err* est un événement de détection d'une anomalie conduisant à l'envoi de valeurs erronées pour un équipement donné. Les paramètres de cet événement sont un équipement *eqt* et une faute *fault*. La faute est détectée et l'équipement passe dans l'état erroné.

L'événement *detect_lost* est un événement de détection d'une anomalie conduisant à la perte d'un équipement donné. Les paramètres de cet événement sont un équipement *eqt* et une faute *fault*. La faute est détectée et l'équipement passe dans l'état perdu.

Reconfiguration

L'événement *recover* constitue l'étape de reconfiguration suite à une faute. On a comme paramètres un équipement *eq* et un équipement redondant distinct associé *eq_red*. L'événement est activé à condition que l'équipement concerné soit dans un état actif, que l'équipement redondant soit inactive. L'action engendrée concerne une activation de l'équipement redondant et une mise hors service de l'équipement perdu.

L'événement *recover_err* réalise un changement d'état vers un état erroné pour un équipement initialement dans un état nominal.

Algorithme 4.1 s_send et a_return (EQUIP_1_1)

event *s_send*
any sen, v **where**
$flag(eqt\_mode(sen)) = active$
$e\_status(eqt\_surv(sen)) \neq lost$
$buffer\_obs(eqt\_obs(sen)) = FALSE$
...
then
$value(eqt\_obs(sen)) := v$
$buffer\_obs(eqt\_obs(sen)) := TRUE$
end

event *a_return*
any act, v **where**
$flag(eqt\_mode(act)) = active$
$e\_status(eqt\_surv(act)) \neq lost$
$buffer\_obs(eqt\_obs(act)) = FALSE$
...
then
$value(eqt\_obs(act)) := v$
$buffer\_obs(eqt\_obs(act)) := TRUE$
$flag(eqt\_mode(act)) := idle$
end

Algorithme 4.2 a_activate et recover (EQUIP_1_1)

event *a_activate*
any *act* **where**
$act \in ACTUATOR$
$flag(eqt\_mode(act)) = idle$
$e\_status(eqt\_surv(act)) \neq lost$
then
$flag(eqt\_mode(act)) := active$
end

event *recover*
any eq, eq_red **where**
$flag(eqt\_mode(eq)) = active$
$flag(eqt\_mode(eq\_red)) = spare$
...
then
$flag := flag \Leftarrow \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto off\}$ [7]
end

Propriétés Les principales propriétés structurantes de notre modèle sont décrites par des invariants au niveau de la composante "Machine" du modèle. Dans la machine *EQUIP_1_1*, seules des invariants définissant le type de variables d'état sont précisés pour faciliter les preuves.

4.3.4 Description détaillée

▶ Caractéristiques dynamiques

La première machine *EQUIP_2* raffinant ce modèle abstrait d'équipements stipule le comportement des équipements en présence de fautes. On trouve donc dans ce modèle le raffinement des deux événements de détection et l'événement de reconfiguration consécutive à une défaillance diagnostiquée.

Détection

Les conditions d'activation de l'événement *detect_err* sont renforcées par les prédicats sur l'état actif et nominal de l'équipement considéré et sur l'appartenance de la faute considérée à la classe de fautes impactant cet équipement.

[7] $\Leftarrow$ est un opérateur de surchage d'une fonction qui, pour $f \Leftarrow \{x \mapsto y\}$, force dans la fonction f un couple antécédent-image défini par $\{x \mapsto y\}$

De façon similaire, les conditions d'activation de l'événement *detect_lost* sont accentuées par les prédicats sur l'état actif et nominal ou erroné de l'équipement considéré et sur l'appartenance de la faute considérée à la classe de fautes impactant cet équipement.

Reconfiguration

L'événement *recover* est modifié en précisant que son déclenchement a lieu à condition que l'équipement concerné soit dans un état perdu, que l'équipement redondant distinct fasse partie d'une catégorie d'équipements redondants dont le nombre d'éléments est strictement supérieur à un. Les actions engendrées concernent une activation de l'équipement redondant et une mise hors service de l'équipement perdu et une diminution du nombre d'équipements redondants restant.

Algorithme 4.3 recover et detect_lost (EQUIP_2)

event *recover*
any eq, eq_red **where**
$eq\_red \in eqt\_cat(eq)$
$e\_status(eqt\_surv(eq)) = lost$
$flag(eqt\_mode(eq)) = active$
$flag(eqt\_mode(eq\_red)) = spare$
$nb(eqt\_cat(eq)) > 1$
...
then
$flag := flag \Leftarrow \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto off\}$
$nb(eqt\_cat(eq)) := nb(eqt\_cat(eq)) - 1$
end

event *detect_lost*
any $fault, eqt$ **where**
$fault \mapsto eqt \in e\_fault\_class$
$e\_status(eqt\_surv(eqt)) \neq lost$
$flag(eqt\_mode(eqt)) = active$
then
$e\_status(eqt\_surv(eqt)) := lost$
end

Propriétés Les propriétés structurantes de ce modèle spécifient que les catégories d'équipements redondants sont non vides, disjointes et distinguent les actionneurs des capteurs (tableau 4.4). Ces propriétés sont nécessaires pour prouver les exigences de tolérance à un nombre de fautes.

En outre, les propriétés invariantes relatives aux caractéristiques dynamiques de ce modèle concernent la définition du nombre *nb* des équipements redondants ou équivalents opérationnels (sous-ensemble $SS\_EQUIPMENT$) en fonction des états de fonctionnement de ces équipements (tableau 4.5). En effet, un nombre *nb* supérieur à 1 indique qu'il y a un équipement redondant équivalent de secours ($flag(eqt\_mode(b)) = spare$) et qu'un autre équipement est en activité (état *active* ou *idle*). Un nombre *nb* égale à 1 implique qu'un équipement de cette catégorie est hors service (état off), tandis qu'un autre est opérationnel (état différent de off). Par contre, un nombre *nb* égale à 0 signifie que tous les équipements de cette catégorie sont hors service (état off).

axm1 : $\forall sse \cdot (sse \in SS\_EQUIPMENT \Rightarrow ((sse \cap SENSOR \neq \varnothing \wedge sse \cap ACTUATOR = \varnothing) \vee (sse \cap SENSOR = \varnothing \wedge sse \cap ACTUATOR \neq \varnothing)))$

axm2 : $\forall sse1, sse2 \cdot (sse1 \in SS\_EQUIPMENT \wedge sse2 \in SS\_EQUIPMENT \Rightarrow (sse1 \neq sse2 \Leftrightarrow sse1 \cap sse2 = \varnothing))$

TABLEAU 4.4 – Propriétés statiques

inv1 : $\forall sse \cdot ((sse \in SS\_EQUIPMENT \wedge nb(sse) > 1) \Rightarrow (\exists a, b \cdot (a \in sse \wedge b \in sse \wedge a \neq b \wedge flag(eqt\_mode(b)) = spare \wedge (flag(eqt\_mode(a)) = active \vee flag(eqt\_mode(a)) = idle))))$

inv2 : $\forall sse, b \cdot ((sse \in SS\_EQUIPMENT \wedge b \in sse \wedge nb(sse) = 1) \Rightarrow (\exists a \cdot (a \in sse \wedge flag(eqt\_mode(a)) \neq off \wedge (a \neq b \Rightarrow flag(eqt\_mode(b)) = off))))$

inv3 : $\forall sse, a \cdot ((sse \in SS\_EQUIPMENT \wedge a \in sse \wedge nb(sse) = 0) \Rightarrow flag(eqt\_mode(a)) = off)$

TABLEAU 4.5 – Propriétés invariantes

4.4 Modélisation de la couche Fonction

La couche Fonction étant une couche intermédiaire entre les couches Equipement et Opération, sa modélisation fait apparaître un double comportement relatif aux interactions à la fois avec la couche inférieure et avec la couche supérieure. Le développement de ces deux modèles de comportement se fait en parallèle. Une fusion de ces modèles pourrait par la suite être envisagée en considérant les procédés de composition [57] et [58].

4.4.1 Hypothèses de modélisation

Les hypothèses prises pour les équipements sont étendues aux fonctions. Par conséquent, on suppose que :

- les fonctionnalités étudiées concernent principalement le contrôle d'un système ; cela implique que certaines fonctions sont chargées d'envoyer des commandes aux actionneurs ;
- les ressources nécessaires à l'exécution de certaines fonctions sont limitées à l'acquisition de données utiles soit provenant d'une autre fonction, soit provenant d'un équipement ;
- toutes les fonctions considérées sont en permanence actives, à moins d'être mises hors service.

4.4.2 Processus de modélisation

La modélisation des fonctions fait apparaître un double comportement relatif aux interactions à la fois avec la couche équipement et avec la couche opération. De même que pour les équipements, les fonctions sont modélisées par leurs activités spécifiques. Ces activités fonctionnelles régulières concernent l'acquisition de données, l'activation et l'exécution d'un traitement.

Dans un premier temps, on s'intéresse en priorité au comportement lié aux équipements en détaillant le protocole de communication pour le transfert des observations des équipements vers des fonctions. On exprime également la nécessité de l'envoi d'une commande vers des actionneurs dans le cas de certaines fonctions.

Dans un second temps, on décrit le comportement des fonctions vis-à-vis des opérations. On marque les dépendances de fonctions relatives aux ressources nécessaires pour leurs activités. Puis, on stipule leur comportement non nominal en décrivant les mécanismes de FDIR propres à cette couche. Ces mécanismes de FDIR censés être plus performants que ceux de la couche équipement, permettent de détecter des valeurs erronées provenant des équipements. Une reconfiguration à destination de l'équipement incriminé est alors envisageable.

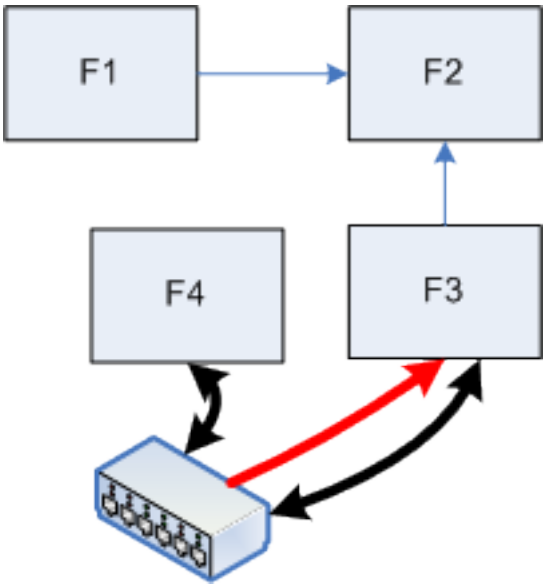

FIG. 4.5 – Principes des fonctions.

La figure 4.6 décrit les étapes de la modélisation de fonctions par deux branches parallèles. La branche de gauche spécifie le comportement des fonctions lié aux équipements :

1. l'établissement du protocole de communication avec la couche équipement et la mise en œuvre de l'envoi de commande ; ces processus sont principalement décrits dans le modèle raffiné [8] E_FCT_1 ;
2. le raffinement du modèle précédent propose une détection des valeurs erronées provenant des équipements pour une reconfiguration des équipements défaillants ; d'autre part, la propagation des défaillances liées à une erreur ou à une perte d'un équipement est exprimée dans le modèle E_FCT_2.

La branche de droite indique les activités de service des fonctions à destination de la couche opération :

1. le séquencement des activités de base est explicité dans le modèle $FUNCT\_0$;
2. on introduit dans le modèle $FUNCT\_1$ l'aspect de dépendance fonctionnelle par rapport à des ressources à travers l'événement $f\_acquire\_r$, ainsi que la production de ressources utiles à d'autres fonctions suggérée dans l'événement $f\_execute\_r$;
3. l'intégration des mécanismes de sécurité avec les événements f_detect_err, f_detect_lost, et $f\_recover$ et la préparation de la communication avec la couche opération à l'aide de l'événement op_enable_obs permettent de constituer un modèle complet en termes d'événements de cette branche, $FUNCT\_1\_1$;

[8] Dans les raffinements, les événements non modifiés ne sont pas représentés volontairement

4. le raffinement de la reconfiguration $f\_recover$ est réalisé dans $FUNCT\_2$.

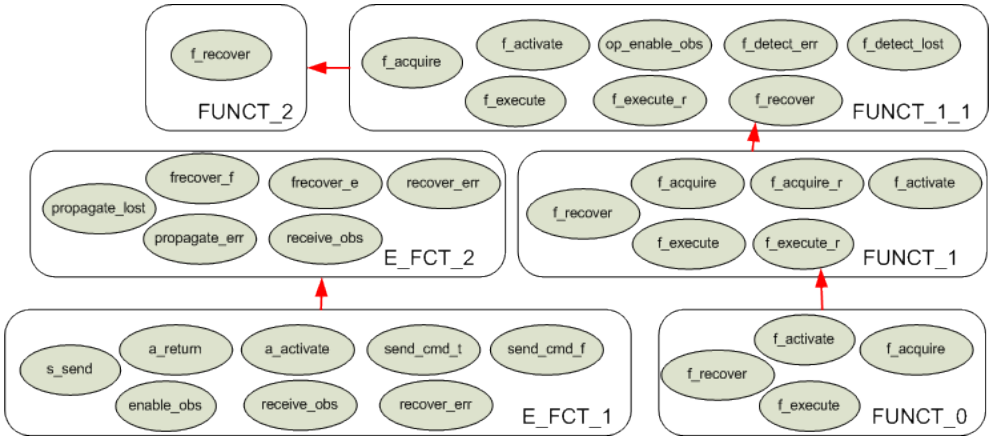

FIG. 4.6 – Etapes de la modélisation des fonctions.

4.4.3 Comportement lié aux équipements

Suite à la modélisation des équipements, le premier modèle de fonctions n'est pas vraiment abstrait car c'est le raffinement du comportement des équipements du point de vue de la couche fonction.

▶ Caractéristiques statiques

- Deux nouveaux ensembles abstraits sont identifiés $COMMAND$ et $FUNCTION$ définissant respectivement un ensemble non vide de commandes et un ensemble non vide de fonctions.
- On définit également une liste de fonctions concernées par l'étude L_FUNCT, ainsi qu'une sous-liste de ces fonctions traitant des commandes envoyées aux actionneurs SL_FUNCT.
- Deux fonctions bijectives permettent de faire le lien entre une fonction envoyant une commande et un actionneur recevant cette commande ($act\_cmd \in ACTUATOR \rightarrowtail\!\!\!\!\!\rightarrow COMMAND$ et $fct\_cmd \in SL\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow COMMAND$).
- Une relation binaire permet d'associer aux fonctions du système des ressources manipulées par ces fonctions ($SL\_FCT\_RES \in L\_FUNCT \leftrightarrow RESOURCE$[9]).

[9] $\leftrightarrow$ est l'opérateur définissant une relation binaire

▶ Caractéristiques dynamiques

La machine *E_FCT_1* raffinant le modèle abstrait d'équipements *EQUIP_1_1* spécifie les interactions entre les couches équipement et fonction. Cette machine complète les événements abstraits et introduit de nouveaux événements précisant les mises à jour des données véhiculées entre ces couches. Trois nouvelles variables d'état *e_cmd*, *f_res* et *fbuffer_obs* sont introduites. Il s'agit de fonctions assignant un booléen à différents attributs. Elles expriment respectivement la présence d'une nouvelle valeur de commande pour un actionneur, la disponibilité de ressources pour des fonctions et l'accessibilité à des observations provenant des équipements vers des fonctions. Ces variables sont alors utilisées dans les événements de ce modèle.

Activation et exécution (équipement)

Les événements d'exécution et d'activation de la couche équipement (*a_activate*, *a_return* et *s_send*) fournissent des services aux fonctions à travers de leurs observations. A ce titre, ils interviennent également dans la communication entre les couches équipement et fonction. Cette communication est modélisée en raffinant ces événements par renforcement de leurs gardes en y incluant des variables relatives à la communication (*e_cmd*, *f_res* et *fbuffer_obs*). La variable *e_cmd* intervient dans les gardes des événements relatifs aux actionneurs :

- Dans les gardes de l'événement *a_activate*, la condition $e\_cmd(act\_cmd(act)) = TRUE$ indique la nécessité de la disponibilité d'une commande actualisée pour l'activation d'un actionneur.
- Concernant l'événement *a_return*, on considère que la condition $e\_cmd(act\_cmd(act)) = FALSE$ correspond à une propriété de stabilité établie par la fonction de commande qui bloque l'envoi de nouvelles valeurs de commande.

Communication

De plus, le transfert d'observations vers les fonctions par l'intermédiaire des événements *a_return* et *s_send* est contraint par la variable *fbuffer_obs*. La valeur *FALSE* pour les buffers d'observation indique une disposition à une mise à jour de l'observation (du côté équipement) ou à une acquisition de l'observation (du côté fonction).

Ainsi, le protocole établi (figure 4.7) peut être assimilé à la procédure de manipulation d'un canal de communication. Au début, le canal est en position fermée à chaque extrémité (figure 4.7 ❶). Puis, pour une transmission de données, sous l'effet des événements *a_return* et *s_send*, il active un passage côté équipement. On autorise alors un transfert de plusieurs données pouvant être stockées dans le canal. On est dans l'état deux (figure 4.7 ❷). Depuis cet

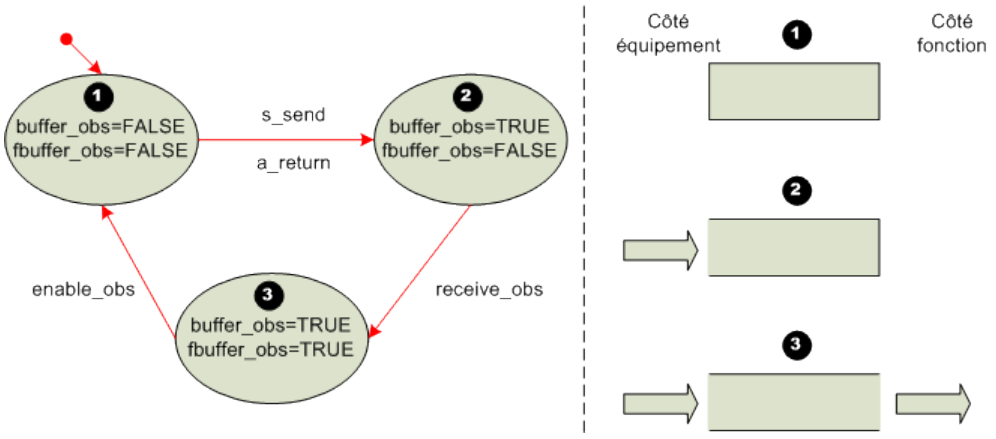

FIG. 4.7 – Protocole de communication pour l'envoi de données vers des fonctions.

état, on peut alors activer le passage côté fonction par l'événement *receive_obs* dédié uniquement à la communication. Le canal est entièrement ouvert, on est dans l'état trois (figure 4.7 ❸). Enfin, on revient dans l'état un (figure 4.7 ❶) en verrouillant le canal pour une nouvelle transmission contrôlée à l'aide d'un autre événement purement de communication *enable_obs*.

Exécution (fonction)

Deux nouveaux événements *send_cmd_t* et *send_cmd_f* relatifs à une fonction commande (ayant comme paramètre une fonction de la sous-liste des fonctions de commande des actionneurs) permettent d'autoriser l'envoi de commandes actualisées. Pour ces événements, on a fait le choix de ne pas détailler le protocole de transfert de commandes depuis la couche fonction vers la couche équipement, comme il s'agit de communication ne révélant pas des anomalies de fonctionnement. Le traitement d'éventuelles défaillances de fonctions de commande ne peut se faire qu'au niveau fonction et non au niveau équipement.

Algorithme 4.4 a_return et receive_obs (E_FCT_1)

event a_return
any act, v **where**
$flag(eqt\_mode(act)) = active$
$e\_status(eqt\_surv(act)) \neq lost$
$e\_cmd(act\_cmd(act)) = FALSE$
$buffer\_obs(eqt\_obs(act)) = FALSE$
$fbuffer\_obs(eqt\_obs(act)) = FALSE$
then
$flag(eqt\_mode(act)) := idle$
$value(eqt\_obs(act)) := v$
$buffer\_obs(eqt\_obs(act)) := TRUE$
end

event $receive\_obs$
any obs, SL_fct_o **where**
$SL\_fct\_o \times \{obs\} \subseteq SL\_FCT\_RES$
$buffer\_obs(obs) = TRUE$
$fbuffer\_obs(obs) = FALSE$
...
then
$f\_res :\in \{f\_res \lhd ((SL\_fct\_o \times \{obs\}) \times \{TRUE\}), f\_res \lhd ((SL\_fct\_o \times \{obs\}) \times \{FALSE\})\}$ [10]
$fbuffer\_obs(obs) := TRUE$
end

Propriétés Au-delà des propriétés de déclarations de constantes et de variables d'état, la principale propriété dynamique à retenir concerne le protocole de communication entre la couche équipement et la couche fonction (tableau 4.6). Cette propriété stipule que l'on ne peut avoir une mise à jour d'une

$$\texttt{inv4 :}\ \forall obs \cdot (obs \in E\_OBSERVATION \Rightarrow (buffer\_obs(obs) = TRUE \vee fbuffer\_obs(obs) = FALSE))$$

TABLEAU 4.6 – Propriété de communication entre équipements et fonctions

observation d'un équipement pendant la réception de celle-ci côté fonction. Elle correspond à l'exclusion de l'état associé à $buffer\_obs = FALSE$ et $fbuffer\_obs = TRUE$.

4.4.4 Propagations de défaillances entre couches

On s'intéresse au modèle E_FCT_2 raffinant E_FCT_1. Les propagations des défaillances des équipements vers des fonctions sont envisagées sous deux angles :

- soit la défaillance n'est pas détectée par l'équipement ou bien n'est pas maîtrisable par l'équipement, alors au niveau fonction, on détecte cette défaillance (réduite au cas de valeurs incohérentes), et on reconfigure l'équipement incriminé ;
- soit la défaillance est avérée mais non mitigée, alors ses effets sont propagés à certaines fonctions fixées par l'architecture considérée.

[10] $\times$ est l'opérateur de produit cartésien entre deux ensembles qui définit un ensemble de tous les couples, ou les pairs d'éléments, issus de ces ensembles

▶ Caractéristiques statiques

- On définit un nouvel ensemble abstrait d'attributs de surveillance $F\_ESURVEILLANCE$ manipulés par des fonctions mais correspondant à l'état de santé des équipements ($eqt\_fsurv \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow F\_ESURVEILLANCE$).
- L'ensemble abstrait $VALUE$ est décomposé en deux ensembles disjoints de valeurs $FAILED$ et $NOMINAL$ traduisant respectivement des valeurs spécifiées erronées consécutives à une défaillance et des valeurs correctes conformes à la spécification de l'équipement.
- On déclare un ensemble abstrait de défaillances $UP\_FAILURE$ ayant des impacts sur des fonctions.
- On définit également une relation constante ($f\_struct \in UP\_FAILURE \leftrightarrow \mathbb{P}_1(L\_FUNCT)$[11]) qui indique l'ensemble des fonctions impactées par une défaillance dans l'architecture considérée.
- De plus, on définit une fonction constante d'évaluation permettant de déterminer si une valeur est correcte ou non ($f\_eval \in VALUE \rightarrow BOOL$).

Propriétés Les principales propriétés statiques sont données dans le tableau 4.7. Ces propriétés permettent de caractériser de façon abstraite le do-

axm3 : $\forall v \cdot (v \in NOMINAL \Rightarrow f\_eval(v) = TRUE)$
axm4 : $\forall v \cdot (v \in FAILED \Rightarrow f\_eval(v) = FALSE)$

TABLEAU 4.7 – Propriétés de cohérence des valeurs

maine de valeurs pour la détection de fautes.

▶ Caractéristiques dynamiques

Communication

Dans la machine E_FCT_2, on introduit une nouvelle variable d'état ($cmd\_estatus \in F\_ESURVEILLANCE \rightarrow E\_STATUS$) permettant d'associer un état de santé à un attribut de surveillance d'un équipement à partir d'une fonction. L'événement raffiné *receive_obs* est explicité par le renforcement de sa garde qui contient maintenant une condition supplémentaire d'évaluation de l'observation devant être positive ($f\_eval(value(obs)) = TRUE$).

[11] $\mathbb{P}_1(S)$ est l'opérateur puissance d'ensembles qui définit l'ensemble non vide des parties (ou de tous les sous-ensembles) d'un ensemble S

Dans ce cas, l'indéterminisme de l'action dans E_FCT_1 est résolu et on attribue une disponibilité de la ressource au profit des fonctions concernées ($f\_res := f\_res \lhd\!\!+ ((SL\_fct\_o \times \{obs\}) \times \{TRUE\})$).

Détection

Deux nouveaux événements *propagate_err* et *propagate_lost* concernent les fonctions non perdues ($fp \mapsto fct\_set \in f\_struct$ et $fct\_set \times \{f\_lost\} \not\subseteq fct\_surv; f\_status$) et pouvant être affectées par la défaillance d'un équipement ou d'une fonction ($fp \in UP\_FAILURE$), et modifient l'état de ces fonctions en conséquence ($f\_status := f\_status \lhd\!\!+ (sf \times \{f\_lost\})$).

Reconfiguration

L'événement *frecover_f* raffine également l'événement abstrait *receive_obs* en précisant le cas où l'indéterminisme de celui-ci est levé en faisant l'attribution contraire qui consiste à déclarer la ressource indisponible pour les fonctions concernées ($f\_res := f\_res \lhd\!\!+ ((SL\_fct\_o \times \{obs\}) \times \{FALSE\})$. La garde est alors renforcée par une condition supplémentaire d'évaluation de l'observation négative ($f\_eval(value(obs)) = FALSE$).

Le nouvel événement *frecover_e* prend comme paramètre un équipement actif et paraissant nominal dans la couche équipement. Si la fonction chargée de la surveillance de cet équipement indique qu'il ne rend pas le service attendu, alors *frecover_e* envoie une commande faisant passer l'état de cet équipement dans l'état erroné bien qu'aucune faute n'ait été identifiée. Ce changement effectif de l'état de l'équipement est réalisé par l'événement *recover_err*.

Algorithme 4.5 propagate_err et receive_obs (E_FCT_2)

event *propagate_err*
any fp, fct_set, sf **where**
$fp \in UP\_FAILURE$
$fct\_set \in \mathbb{P}_1(L\_FUNCT)$
$sf \subset F\_SURVEILLANCE$
$fp \mapsto fct\_set \in f\_struct$
$fct\_set \times \{f\_lost\} \not\subseteq fct\_surv; f\_status$
$fct\_set \times sf \subset fct\_surv$
then
$f\_status := f\_status \lhd\!\!+ (sf \times \{f\_erroneous\})$
end

event *receive_obs*
any obs, SL_fct_o **where**
$SL\_fct\_o \times \{obs\} \subseteq SL\_FCT\_RES$
$buffer\_obs(obs) = TRUE$
$fbuffer\_obs(obs) = FALSE$
$f\_eval(value(obs)) = TRUE$
then
$f\_res := f\_res \lhd\!\!+ ((SL\_fct\_o \times \{obs\}) \times \{TRUE\})$
$fbuffer\_obs(obs) := TRUE$
end

4.4.5 Services pour la couche opération

Depuis la machine $FUNCT\_0$ à la machine $FUNCT\_1\_1$, le modèle de fonctions constitue le point de départ de la description du comportement des fonctions vis-à-vis des opérations.

▶ Caractéristiques statiques

Le "Contexte" de ce modèle abstrait définit :
- des ensembles abstraits relatifs aux fonctions *FUNCTION*, *F_MODE*, *F_SURVEILLANCE*, *F_OBSERVATION*,
- des ensembles abstraits globaux *VALUE*, *RESOURCE* et *DEFAULT*,
- des ensembles énumérés *F_FLAG* et *F_STATUS* (voir section 4.2.1).

De façon similaire aux équipements, on a des fonctions constantes :
- $fct\_surv \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow F\_SURVEILLANCE$,
- $fct\_mode \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow F\_MODE$,
- $fct\_obs \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow F\_OBSERVATION$,
- $f\_fault\_class \in DEFAULT \nrightarrow L\_FUNCT$.

▶ Caractéristiques dynamiques

La machine *FUNCT_1_1* de ce modèle décrit quatre événements traduisant le fonctionnement nominal des fonctions, un événement concernant les échanges avec la couche supérieure et trois événements représentant des situations de défaillance.

Activation

L'événement *f_activate* spécifie l'activation d'une fonction à condition qu'elle soit préalablement dans un état d'attente, et qu'elle soit dans un état nominal ou erroné.

Acquisition

L'événement *f_acquire* indique le passage d'une activation vers une étape d'exécution après acquisition éventuelle de ressources d'entrée. L'événement *f_execute* spécifie l'exécution d'une fonction avec comme résultat la production d'une observation modifiée ($fvalue(fct\_obs(fct)) := v$), l'indication de cette mise à jour par un buffer ($fbuffer\_h(fct\_obs(fct)) := TRUE$) et le passage à l'état *f_idle*. Ces deux événements sont complétés par deux événements supplémentaires traitant de la prise en compte de ressources lors de l'acquisition et à la fin de l'exécution. Ces ressources témoignent de l'interdépendance des fonctions et de leur dépendance vis-à-vis des équipements.

Communication

L'événement *op_ enable_ obs* permet de préparer un buffer pour l'envoi des observations vers la couche opération (passage de *TRUE* à *FALSE*).

Détection

Les événements suivants de diagnostic et de reconfiguration d'une faute sont indépendants de sollicitations d'une quelconque opération. L'événement *f_ detect_ err* est un événement de détection d'une faute conduisant à l'envoi de valeurs erronées pour une fonction donnée. La faute est détectée à condition que la fonction soit dans un mode d'exécution et que son état soit nominal.

L'événement *f_ detect_ lost* est un événement de détection d'une faute conduisant à la perte d'une fonction donnée. La faute est détectée à condition que la fonction soit activée ou qu'elle soit dans un mode d'exécution et que son état soit nominal ou erroné.

Reconfiguration

Enfin, l'événement *f_ recover* constitue l'étape de reconfiguration suite à une faute qui consiste à passer une fonction de l'état erroné à l'état perdu.

Algorithme 4.6 f_execute_r et f_detect_lost (FUNCT_1_1)

event $f\_execute\_r$
any fct, res, v **where**
$fct \mapsto res \in SL\_FCT\_RES$
$f\_flag(fct\_mode(fct)) = f\_executing$
$f\_res(fct \mapsto res) = FALSE$
$fbuffer\_h(fct\_obs(fct)) = FALSE$
...
then
$f\_flag(fct\_mode(fct)) := f\_idle$
$f\_res(fct \mapsto res) := TRUE$
$fvalue(fct\_obs(fct)) := v$
$fbuffer\_h(fct\_obs(fct)) := TRUE$
end

event f_detect_lost
any $fault, fct$ **where**
$fault \mapsto fct \in f\_fault\_class$
$f\_flag(fct\_mode(fct)) = f\_executing \vee$
$f\_flag(fct\_mode(fct)) = f\_active$
$f\_status(fct\_surv(fct)) \neq f\_lost$
...
then
$f\_status(fct\_surv(fct)) := f\_lost$
end

4.4.6 Modèle raffiné de fonction

► Caractéristiques statiques

On définit dans le contexte étendu un sous-ensemble de fonctions équivalentes ($fct\_cat \in L\_FUNCT \rightarrow SS\_FUNCTION$).

Propriétés La principale propriété statique est indiquée dans le tableau 4.8 et stipule que toute fonction définissant une catégorie de fonctions équivalentes appartient à cette catégorie.

axm6 : $\forall f \cdot (f \in dom(fct\_cat) \Rightarrow f \in fct\_cat(f))$

TABLEAU 4.8 – Propriété statique sur les classes de fonctions équivalentes.

▶ Caractéristiques dynamiques

Reconfiguration

La machine *FUNCT_2* traite en particulier l'événement de reconfiguration. L'événement *f_recover* est modifié en précisant que son déclenchement a lieu à condition que la fonction concernée ne soit pas dans un état nominal ($f\_status(fct\_surv(fct)) \neq f\_ok$), que la fonction équivalente distincte fasse partie d'une catégorie de fonctions équivalentes ($fct\_eq \in fct\_cat(fct)$) dont le nombre d'éléments est strictement supérieur à un ($f\_nb(fct\_cat(fct)) > 1$) et que la fonction équivalente soit dans un état nominal ($f\_status(fct\_surv(fct\_eq)) = f\_ok$). Les actions engendrées concernent la désactivation de la fonction défaillante et une diminution du nombre de fonctions équivalentes restant ($f\_nb(fct\_cat(fct)) := f\_nb(fct\_cat(fct)) - 1$).

Algorithme 4.7 f_recover (FUNCT_2)

event $f\_recover$
any fct, fct_eq **where**
$fct \in L\_FUNCT$
$fct\_eq \in L\_FUNCT$
$fct\_eq \in fct\_cat(fct)$
$fct\_eq \neq fct$
$f\_status(fct\_surv(fct)) \neq f\_ok$
$f\_status(fct\_surv(fct\_eq)) = f\_ok$
$f\_nb(fct\_cat(fct)) > 1$
$f\_flag(fct\_mode(fct)) \neq f\_off$
$f\_flag(fct\_mode(fct\_eq)) \neq f\_off$
then
$f\_nb(fct\_cat(fct)) := f\_nb(fct\_cat(fct)) - 1$
$f\_status(fct\_surv(fct)) := f\_lost$
$f\_flag(fct\_mode(fct)) := f\_off$
end

Propriétés Les propriétés invariantes relatives aux caractéristiques dynamiques de ce modèle concernent la définition du nombre f_nb des fonctions équivalentes opérationnelles en fonction des états de fonctionnement de ces fonctions (voir tableau 4.9). En effet, un nombre f_nb supérieur à 1 indique qu'il y a au moins deux fonctions équivalentes opérationnelles ($f\_flag(fct\_mode(a)) \neq f\_off$ et $f\_flag(fct\_mode(b)) \neq f\_off$). Un nombre f_nb égale à 1 implique qu'une fonction de cette catégorie est hors service (état off), tandis qu'une autre est opérationnelle (état différent de off). Par contre, un nombre f_nb égale à 0 signifie que toutes les fonctions de cette catégorie sont hors service (état off).

inv6 : $\forall ssf \cdot ((ssf \in SS\_FUNCTION \wedge f\_nb(ssf) > 1) \Rightarrow (\exists a, b \cdot (a \in ssf \wedge b \in ssf \wedge a \neq b \wedge f\_flag(fct\_mode(a)) \neq f\_off \wedge f\_flag(fct\_mode(b)) \neq f\_off)))$

inv7 : $\forall ssf, b \cdot ((ssf \in SS\_FUNCTION \wedge b \in ssf \wedge f\_nb(ssf) = 1) \Rightarrow (\exists a \cdot (a \in ssf \wedge f\_flag(fct\_mode(a)) \neq f\_off \wedge (a \neq b \Rightarrow f\_flag(fct\_mode(b)) = f\_off))))$

inv8 : $\forall ssf, a \cdot ((ssf \in SS\_FUNCTION \wedge a \in ssf \wedge f\_nb(ssf) = 0) \Rightarrow f\_flag(fct\_mode(a)) = f\_off)$

TABLEAU 4.9 – Propriétés sur le nombre de fonctions équivalentes.

4.5 Modélisation de la couche Opération

La couche opération est la couche ultime de notre architecture, dont le rôle est de gérer le comportement du système en fonction d'un scénario opérationnel nominal plus ou moins prédéfini, mais également en prenant en compte des aléas liés à des défaillances de services lors de l'accomplissement de la mission souhaitée.

4.5.1 Hypothèses de modélisation

Pour la couche opération, on considère les hypothèses suivantes :
- les opérations sont pré définies par le scénario de mission souhaité (figure 4.8) ;
- l'interaction avec un opérateur humain n'est pas prise en compte dans la thèse ;

– pour pouvoir évaluer la robustesse des couches fonction et équipement en l'absence d'hypothèses sur l'opérateur humain qui pourrait récupérer d'éventuelles défaillances de la couche opération, il n'y a pas de panne associée à la gestion des opérations.

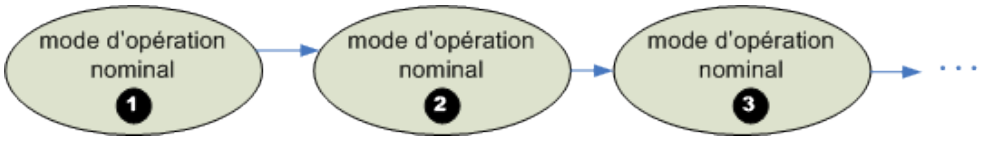

FIG. 4.8 – Scénario de mission.

4.5.2 Processus de modélisation

La couche opération spécifie principalement la gestion des modes d'opérations relatifs à un scénario de mission. Un mode d'opération coordonne un ensemble de fonctions pour répondre aux besoins spécifiques posés par une phase de la mission (figure 4.9). Le passage d'un mode au suivant se fait pour réaliser l'étape suivante de la mission ou pour adapter les performances de l'étape courante à l'état de santé du système.

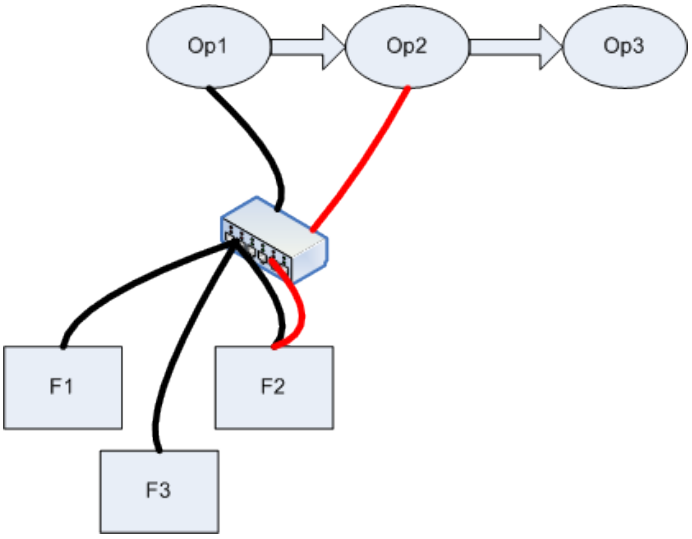

FIG. 4.9 – Principes des opérations.

Dans la modélisation de cette couche (figure 4.10), on identifie deux étapes importantes :

1. la communication avec la couche fonction par envoi de requêtes et par réception des observations qui est mise en place dans le modèle F_OPE_1 ;
2. l'établissement du mécanisme de gestion des modes opérationnels spécifié dans les modèles F_OPE_2 et F_OPE_3.

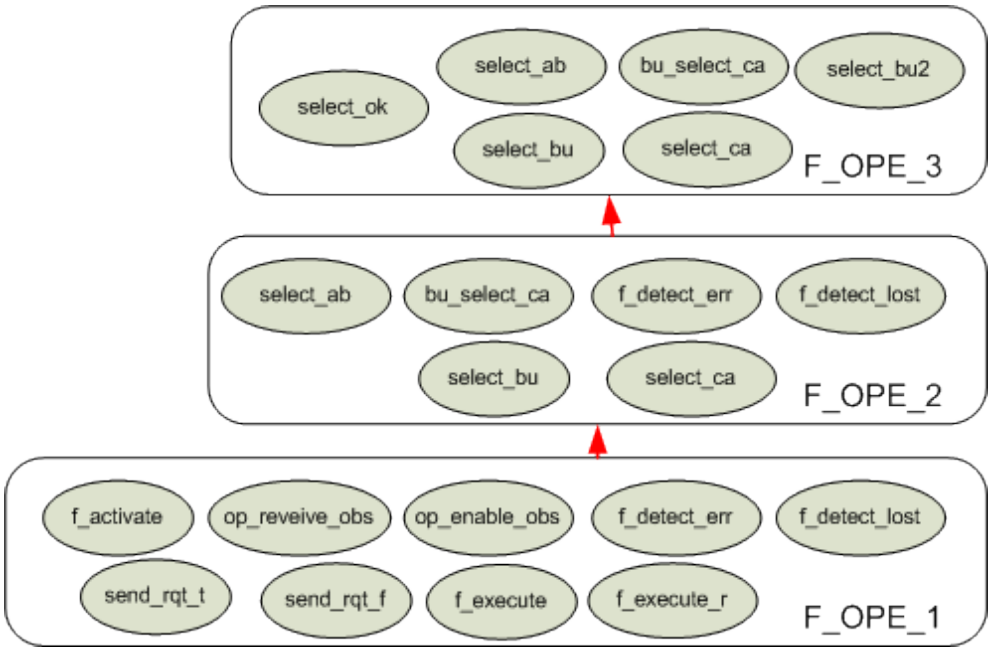

FIG. 4.10 – Etapes de la modélisation des opérations.

4.5.3 Communication en lien avec les fonctions

En lien avec la modélisation des fonctions, ce modèle abstrait des opérations F_OPE_1 est en fait un raffinement du comportement des fonctions du point de vue de la couche opération.

▶ Caractéristiques statiques

On introduit :
- le concept de requêtes (ensemble abstrait $REQUEST$)
- et une fonction caractéristique désignant la requête associée à une fonction de la liste des fonctions concernées par l'étude ($ct\_req \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow REQUEST$).

▶ Caractéristiques dynamiques

La machine F_OPE_1 raffinant le modèle abstrait de fonctions $FUNCT\_1\_1$ spécifie les interactions entre les couches opération et fonction. Cette machine complète les événements abstraits et introduit de nouveaux événements précisant les mises à jour des données véhiculées entre ces couches. Deux nouvelles variables d'état f_rqt et $fopbuffer\_h$ sont introduites. Il s'agit de fonctions assignant un booléen à différents attributs. Elles expriment respectivement l'envoi d'une requête pour l'activation d'une fonction et l'accessibilité des opérations à des observations provenant des fonctions sous-jacentes. Les

échanges de requêtes et d'observations sont inspirés des protocoles de communication mis en place entre les couches équipement et fonction (figure 4.7).

Communication via activation et exécution (fonction)

Les événements issus de la couche fonction sont raffinés au moyen du renforcement de leurs gardes. Ainsi, les événements *f_ execute* et *f_ execute_ r* prennent en compte dans leurs gardes une contrainte supplémentaire relative au protocole de communication proposée entre les couches du système. Cette contrainte indique que le buffer est prêt par rapport à la couche opération pour une mise à jour d'une observation. L'actualisation de l'état du buffer du point de vue de l'opération est réalisée dans l'événement raffiné *op_ enable_ obs* ainsi que dans le nouvel événement *op_ receive_ obs*.

Concernant l'événement *f_ activate*, son raffinement introduit une garde conditionnant l'activation à la présence d'une requête spécifique. Les modifications de la présence des requêtes sont orchestrées par les événements *send_ rqt_ t* et *send_ rqt_ f* qui positionnent respectivement à $TRUE$ et $FALSE$ l'état des requêtes lors de leurs envois.

Algorithme 4.8 f_execute_r et op_enable_obs (F_OPE_1)

event $f\_execute\_r$
any fct, res, v **where**
$fct \mapsto res \in SL\_FCT\_RES$
$f\_flag(fct\_mode(fct)) = f\_executing$
$f\_res(fct \mapsto res) = FALSE$
$fbuffer\_h(fct\_obs(fct)) = FALSE$
$fopbuffer\_h(fct\_obs(fct)) = FALSE$
...
then
$f\_flag(fct\_mode(fct)) := f\_idle$
$f\_res(fct \mapsto res) := TRUE$
$fvalue(fct\_obs(fct)) := v$
$fbuffer\_h(fct\_obs(fct)) := TRUE$
end

event op_enable_obs
any obs **where**
$obs \in F\_OBSERVATION$
$fbuffer\_h(obs) = TRUE$
$fopbuffer\_h(obs) = TRUE$
...
then
$fbuffer\_h(obs) := FALSE$
$fopbuffer\_h(obs) := FALSE$
end

4.5.4 Gestion des modes opérationnels

Ce raffinement (en deux étapes, F_OPE_2 et F_OPE_3) définit un scénario opérationnel et explicite le mécanisme de changement de phases. Ce niveau de l'architecture est chargé de la gestion et de la surveillance en dernier recours de l'ensemble du système. Une faute à ce niveau peut éventuellement être récupérée ultimement par l'opérateur humain du système. La modélisation de l'opérateur étant hors du périmètre de cette thèse, pour pouvoir malgré tout évaluer la robustesse des couches fonction et équipement, nous avons supposé que des fautes n'affectent pas le fonctionnement de du niveau opération.

▶ Caractéristiques statiques

Dans le contexte associé à ce modèle,

- on définit un ensemble abstrait global d'opérations $OPERATION$ comprenant deux opérations singulières distinctes correspondant à une phase de repli OP_bu et une phase d'échec de la mission OP_ca ;
- une séquence d'opérations nominales est également établie ($seq\_op \in 1\,..\,n \rightarrow OPERATION \setminus \{OP\_bu, OP\_ca\}$) ;
- les modes opérationnels sont décrits dans un ensemble énuméré $MODE\_OP = \{m\_ok, m\_backup, m\_aborted, m\_cancelled\}$;
- on définit également un ensemble abstrait de attributs O_MODE à associer à chaque opération ($ope\_mode \in OPERATION \rightarrowtail\!\!\!\!\!\!\rightarrow O\_MODE$) ;
- de plus, des groupes de fonctions sont rassemblés dans des sous-ensembles conformément aux opérations qui les manipulent ($L\_FUNCT\_OPE \subseteq \mathbb{P}_1(L\_FUNCT)$ et $l\_fct\_op \in OPERATION \rightarrowtail\!\!\!\!\!\!\rightarrow L\_FUNCT\_OPE$).

Propriétés Ainsi, une propriété spécifique consiste à indiquer que toutes les fonctions du système doivent être reliées à une opération (voir tableau 4.10).

$$\texttt{axm5} : \forall f \cdot (f \in L\_FUNCT \Rightarrow (\exists ssf \cdot (ssf \in L\_FUNCT\_OPE \wedge f \in ssf)))$$

Tableau 4.10 – Propriété statique de dépendance des fonctions par rapport aux opérations.

▶ Caractéristiques dynamiques

Les caractéristiques dynamiques s'articulent autour d'une variable d'état $i \in 1\,..\,n$ représentant la phase opérationnelle en cours. Les événements décrits dans la machine F_OPE_3 explicite le mécanisme de changement de modes opérationnels. Le passage d'un mode d'opération au suivant dépend de la mission prédéfinie et de l'état courant de l'opération. L'évolution de l'état d'une opération est décrite par l'automate suivant (figure 4.11).

Activation

L'événement $select\_bu$ stipule que le passage du mode nominal au mode de repli se fait à condition qu'une fonction associée à la phase en cours et à la phase suivante soit dans un état erroné et non hors service. Le mode relatif à l'opération de repli est forcé à la valeur m_backup.

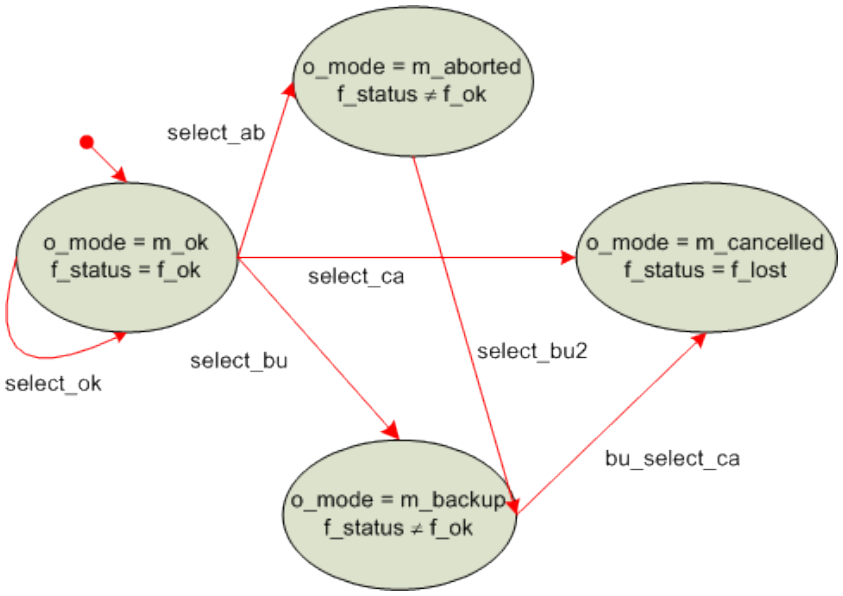

FIG. 4.11 – Automate de modes d'opération.

De même, l'événement *select_ab* passe du mode nominal au mode interrompu si une fonction appartenant uniquement à la phase en cours est défaillante et en service, et que le mode opérationnel suivant est également nominal.

L'événement *select_ca* spécifie que le passage du mode nominal au mode d'annulation de la mission se fait à condition qu'une fonction associée à la phase en cours et à la phase suivante soit dans un état perdu, sans fonction équivalente sélectionnable et hors service. Le mode relatif à l'opération d'annulation est alors forcé à la valeur *m_cancelled*.

L'événement *bu_select_ca* fait passer le système du mode *m_backup* à *m_cancelled* si une fonction en service du mode de repli est défaillante.

L'événement *select_bu2* indique ce qui se passe dans le cas où le mode en cours est nominal et que le mode suivant est interrompu. Le mode en cours devient en repli et on passe dans une phase de repli.

Enfin, l'événement *select_ok* incrémente l'indice de phase i si la phase en cours et la phase suivante sont nominales.

Dans la section suivante, une première vérification indique que la combinaison de ces événements traduit bien le respect de la propriété de sécurité qualitative imposant un minimum de deux fautes indépendantes pour conduire à la perte totale du système.

Algorithme 4.9 select_ab et select_bu (F_OPE_3)

event *select_ab*
any fct **where**
$fct \in l\_fct\_op(seq\_op(i))$
$fct \notin l\_fct\_op(seq\_op(i+1))$
$o\_mode(ope\_mode(seq\_op(i))) = m\_ok$
$o\_mode(ope\_mode(seq\_op(i+1))) = m\_ok$
$f\_status(fct\_surv(fct)) \neq f\_ok$
$f\_flag(fct\_mode(fct)) \neq f\_off$
...
then
$o\_mode(ope\_mode(seq\_op(i))) := m\_aborted$
$i := i+1$
end

event *select_bu*
any fct **where**
$fct \in l\_fct\_op(seq\_op(i)) \cap l\_fct\_op(seq\_op(i+1))$
$o\_mode(ope\_mode(seq\_op(i))) = m\_ok$
$f\_status(fct\_surv(fct)) = f\_erroneous$
$f\_flag(fct\_mode(fct)) \neq f\_off$
...
then
$o\_mode := o\_mode \Leftarrow \{ope\_mode(seq\_op(i)) \mapsto m\_backup, ope\_mode(OP\_bu) \mapsto m\_backup\}$
$i := i+1$
end

Algorithme 4.10 select_ca et select_bu2 (F_OPE_3)

event *select_ca*
any fct **where**
$fct \in l\_fct\_op(seq\_op(i)) \cap l\_fct\_op(seq\_op(i+1))$
$o\_mode(ope\_mode(seq\_op(i))) = m\_ok$
$f\_status(fct\_surv(fct)) = f\_lost$
$f\_flag(fct\_mode(fct)) = f\_off$
then
$o\_mode := o\_mode \Leftarrow \{ope\_mode(seq\_op(i)) \mapsto m\_cancelled, ope\_mode(OP\_ca) \mapsto m\_cancelled\}$
$i := i+1$
end

event *select_bu2*
when
$i < n$
$o\_mode(ope\_mode(seq\_op(i))) = m\_ok$
$o\_mode(ope\_mode(seq\_op(i+1))) = m\_aborted$
...
then
$o\_mode := o\_mode \Leftarrow \{ope\_mode(seq\_op(i)) \mapsto m\_backup, ope\_mode(OP\_bu) \mapsto m\_backup\}$
$i := i+1$
end

4.6 Propriétés de sécurité prouvées

Dans cette section, nous nous intéresserons principalement aux propriétés de sécurité spécifiées dans l'architecture en couches à prouver par la méthode Event-B. La propriété de sécurité globale qui explicite qu' "une panne simple ne conduit pas à un événement redouté" :

peut se traduire en "une panne simple n'a pas de conséquences dramatiques"

peut se traduire en "une faute est complètement maîtrisée"

peut se traduire en "une faute est détectée, identifiée et mitigée"

Voyons maintenant comment cette propriété induite est prise en compte dans le fonctionnement de chacune des couches de notre architecture.

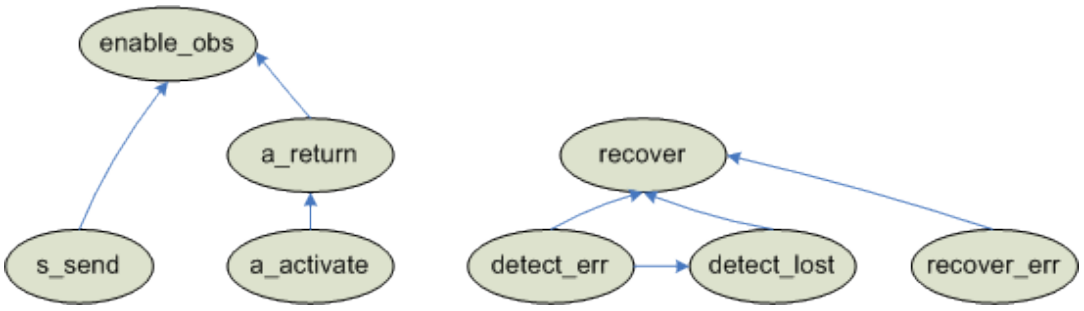

FIG. 4.12 – Synchronisation des événements de la couche équipement.

4.6.1 Synthèse de la couche équipement

Dans la couche équipement, les composants fonctionnels modélisés progressivement rendent compte du comportement des équipements en situation normale où tous les équipements sont dans un état *ok* et également du comportement en situation anormale. Dans ce dernier cas lors d'un premier événement de panne, l'environnement des variables d'état intègre un paramètre de faute *fault* qui déclenche le seul événement de diagnostic activable *detect_x* suivant une classe de fautes caractérisant des pannes permanentes erronées ou perdues (x valant *err* ou *lost*). Le mécanisme d'identification est supposé implicitement établi puisque l'on distingue les pannes erronées et perdues et que l'on précise une relation entre des fautes et des équipements (*e_fault_class*). La synchronisation des événements est représentée dans la figure 4.12. La couche équipement met en œuvre soit des mécanismes de sécurité propres, soit des mécanismes externes (une commande envoyée par une fonction de la couche supérieure est prise en compte dans *recover_err*).

Les trois propriétés invariantes (invariants 1 à 3 de la section 4.3) fixent les états fonctionnels des équipements engagés dans une redondance en fonction d'une variable d'état *nb* indiquant le nombre d'équipements redondants opérationnels. La diversité de ces propriétés permet de couvrir les trois cas de figure distincts possibles :

- cas où l'on a au moins deux équipements redondants opérationnels ;
- cas où l'on a un équipement opérationnel ;
- et cas où il n'y a plus aucun équipement opérationnel.

Ces propriétés assurent la consistance de reconfiguration et imposent des actions impactant plusieurs équipements simultanément pour éviter des situations d'indisponibilité de services (voir l'événement *recover*, section 4.3). Du point de vue de la SdF, il s'agit de prendre compte le critère de disponibilité lors du processus de mise en sécurité du système. L'application de la règle de préservation des invariants à ces propriétés a nécessité des preuves interactives à l'aide du prouveur appelé *Predicate Prover*, notamment du fait des opérateurs universels,

existentiels et d'équivalence logique. A titre d'exemple, la conclusion d'une des preuves est donnée dans le tableau 4.11 :

$$\begin{array}{l}\forall sse \cdot sse \in SS\_EQUIPMENT \wedge \\ \quad (nb \lhd \{eqt\_cat(eq) \mapsto nb(eqt\_cat(eq)) - 1\})(sse) > 1 \\ \qquad \Rightarrow \\ \quad (\exists a, b \cdot a \in sse \wedge b \in sse \wedge a \neq b \wedge \\ \quad (flag \lhd \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto \\ off\})(eqt\_mode(b)) = spare \wedge \\ \quad ((flag \lhd \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto \\ off\})(eqt\_mode(a)) = active \vee \\ \quad (flag \lhd \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto \\ off\})(eqt\_mode(a)) = idle))\end{array}$$

TABLEAU 4.11 – Conclusion d'une preuve.

Ces propriétés relatives à la redondance (invariants 1 à 3 de la section 4.3) servent indirectement à la satisfaction de l'objectif global de sécurité dérivé dans la couche équipement : "une faute simple impactant un équipement ne conduit pas à l'événement redouté de perte totale du système". Cet énoncé suppose qu'une analyse préalable a été effectuée pour déterminer les équipements susceptibles de mener à la perte totale et qu'une redondance adéquate a été mise en place. Ainsi, la post condition qui nous intéresse correspond au théorème *Post* sous les hypothèses que les axiomes et les invariants sont satisfaits [50] :

$$P(c) \wedge I(c, v) \Rightarrow Post \tag{4.1}$$

Avec ici,

$$\begin{array}{l}Post \equiv \\ \quad \forall f1, eq, eqr \cdot ((f1 \mapsto eq \in e\_fault\_class \wedge \\ e\_status(eqt\_surv(eq)) \neq ok \wedge \\ \quad flag(eqt\_mode(eq)) = off \wedge eqr \in eqt\_cat(eq) \wedge \\ flag(eqt\_mode(eqr)) = active \wedge \\ \quad e\_status(eqt\_surv(eqr)) = ok) \Rightarrow f1 \mapsto eqr \notin \\ e\_fault\_class)\end{array}$$

TABLEAU 4.12 – Ecriture du théorème *Post* (équipement).

Cette post condition (tableau 4.12) exprime le maintien de service réalisé par l'équipement redondant après l'apparition d'une faute. Elle stipule que pour toute faute *f1*, pour tout équipement *eq* et pour tout équipement redondant *eqr*, lorsque :

- *f1* est associée à *eq* (*e_fault_class* étant une fonction partielle entre l'ensemble des fautes *DEFAULT* et l'ensemble des équipements *EQUIPMENT*),
- et le statut de *eq* est erroné ou lost,
- et *eq* est hors service,
- et *eqr* est un équipement redondant à *eq* (*eqt_cat* étant une fonction totale de *EQUIPMENT* vers *SS_EQUIPMENT*, ensemble de sous-ensembles d'équipements ; *eqt_cat* stipulant que chaque équipement appartient à une catégorie d'équipements redondants *SS_EQUIPMENT* contenant soit l'équipement seul soit plusieurs équipements redondants),
- et *eqr* est actif avec un statut ok,

alors *f1* n'est pas une faute associée à *eqr*. Ce théorème est vérifié automatiquement par les outils de preuve de la plateforme Rodin.

4.6.2 Synthèse de la couche fonction

La couche fonction revêt une particularité liée à sa position centrale. Ses activités de sécurité concernent la surveillance, le diagnostic et la reconfiguration de ses propres constituants, mais également des équipements contrôlés. Les mécanismes de détection et d'identification sont très similaires à ceux mis en œuvre dans la couche équipement. Cependant, une attention plus importante doit être portée sur ces mécanismes à ce niveau car les conséquences sont plus importantes. Plus précisément, les fonctionnalités de diagnostic de la couche équipement sont plus faciles à réaliser car les activités des équipements sont plus simples et mieux maîtrisées. Concernant la couche fonction, le degré de complexité augmente. Les fonctions sont tributaires des informations éventuellement erronées provenant des équipements tout en assurant une bonne qualité des contrôles réalisés. Dans notre modélisation, cette complexité se ressent dans la spécification des comportements en situation normale et en situation anormale par des interactions renforcées entre les couches et au sein de la couche fonction. Par exemple, dans la couche fonction, une distinction apparaît entre des constituants qui dépendent de ressources des équipements et ceux qui n'en dépendent pas. C'est le cas symbolisé par les événements *f_acquire_r* et *f_execute_r*. De même, les événements *frecover_f* et *frecover_e* établissent des interactions entre couches à travers les observations *obs* et les commandes *cmd_estatus*. En bref, un schéma de synchronisation résume le comportement au sein de la couche fonction (figure 4.13).

De la même manière que pour les équipements, trois propriétés invariantes établissent une équivalence entre le nombre de fonctions dites similaires et l'état de ces fonctions similaires (voir section 4.4). C'est un autre type de redondance au niveau des fonctions qui est moins une copie au sens physique du terme mais plutôt un traitement différencié rendant en partie les mêmes services. Dans notre modélisation, les fonctions similaires sont en permanence activées (ce qui se

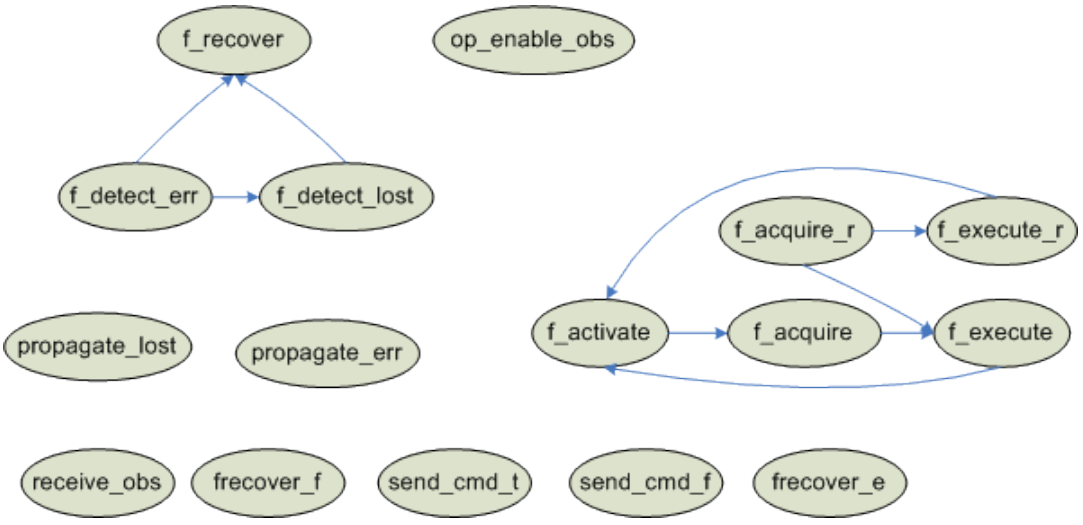

FIG. 4.13 – Synchronisation des événements de la couche fonction.

rapproche d'une redondance chaude) : c'est le cas, par exemple, des fonctions de pilotage manuel, de pilotage automatique "grossier" et de pilotage automatique "fin". Les propriétés invariantes des fonctions assurent également la consistance de reconfiguration et la disponibilité de services. Du point de vue de la SdF, les effets de ces propriétés servent à la couche opération qui gère les configurations globales en fonction de la mission fixée (la gestion de la mission est présentée dans la section 4.5). L'application de la règle de préservation des invariants à ces propriétés a nécessité des preuves interactives plus compliquées que celles des équipements en complétant les hypothèses et en faisant de nombreuses preuves par cas (voir figure 4.14).

La post condition de sécurité de la couche fonction ne devrait concerner que des défaillances touchant des fonctions. Les défaillances provenant des équipements telles l'envoi de valeurs erronées ou l'absence de données, soit elles sont mitigées par une détection ad hoc à l'aide de la fonction *f_eval* dans *frecover_f*, soit elles sont symboliquement transformées en pannes propagées sur des fonctions *UP_FAILURE* (voir section 4.4). En effet, nous avons considéré que certaines valeurs erronées d'équipements en panne peuvent néanmoins être propagées vers des fonctions qui deviennent à leur tour erronées. Egalement, la perte d'un équipement et l'absence de données envoyées peuvent conduire à la perte de plusieurs fonctions. Par conséquent, l'établissement de la composante de l'objectif de sécurité global par une post condition propre à la couche fonction mène nécessairement à la définition de configurations de fonctions (tableau 4.13). La post condition de la couche fonction (tableau 4.13) indique que pour toute faute *f*, pour toute fonction susceptible d'être impactée par une erreur propagée *fp*, pour toute fonction de la liste des fonctions du système (défini par

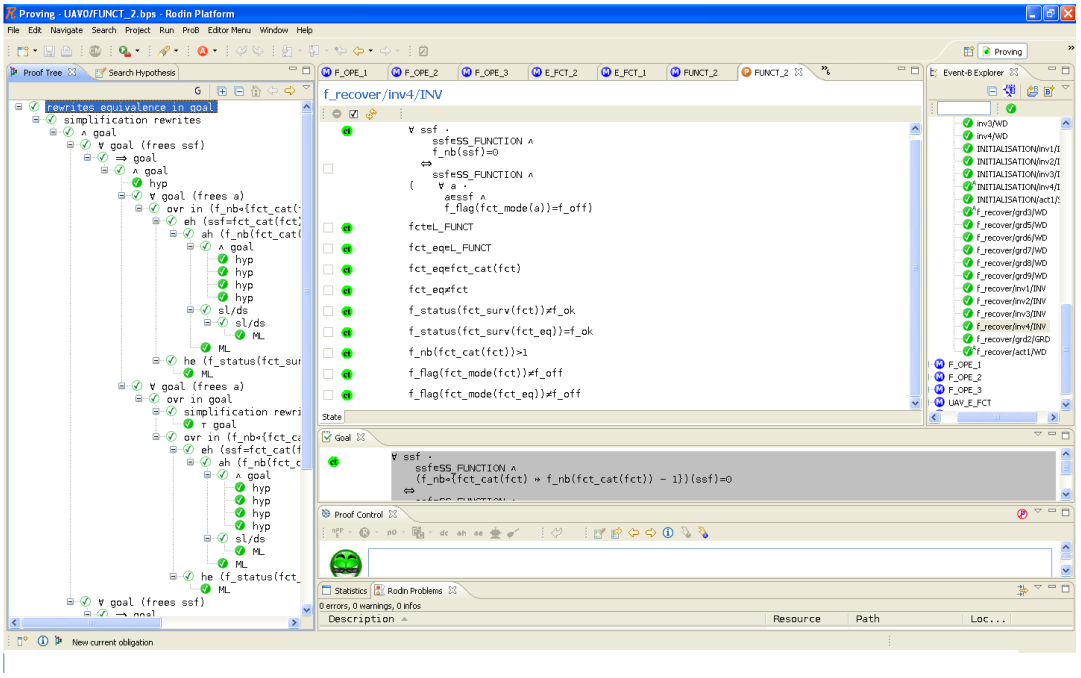

FIG. 4.14 – Environnement de preuves sous Rodin.

L_FUNCT) *fct* et pour toute configuration de fonctions de cette liste *fct_set*, lorsque :

- *f* est associée à *fct* (*f_fault_class* étant une fonction partielle entre l'ensemble des fautes *DEFAULT* et l'ensemble des fonctions spécifiées *L_FUNCT*) ou la fonction *fp* appartient à une configuration de fonctions du système (définie par *f_struct*, relation entre l'ensemble des fonctions susceptibles d'être impactées par une erreur propagée, *UP_FAILURE* et l'ensemble des sous-ensembles de *L_FUNCT*)),
- et le statut de *fct* est erroné ou lost, ou la configuration *fct_set* contient des fonctions au statut erroné ou lost,

alors il existe une configuration *fct_set2* ayant toutes ses fonctions au statut ok.

Le théorème qui découle de cette post condition n'est pas vérifié, comme les configurations de fonctions ne sont pas explicitement définis. Pour le satisfaire, il est nécessaire d'établir des configurations de fonctions pour lesquelles une panne simple n'entraîne pas une perte de l'ensemble des fonctions du système. Par conséquent, cela impose de bien décrire les modes opérationnels.

4.6.3 Synthèse de la couche opération

En rapport avec la mission souhaitée, la couche opération vise à accroître l'autonomie du système et à gérer les différents modes opérationnels en fonction-

Avec ici,
$Post \equiv$
$\forall f, fp, fct, fct\_set \cdot ((f \in DEFAULT \wedge fp \in UP\_FAILURE \wedge fct \in L\_FUNCT \wedge$
$fct\_set \in \mathbb{P}_1(L\_FUNCT) \wedge (f \mapsto fct \in f\_fault\_class \vee fp \mapsto fct\_set \in f\_struct) \wedge$
$(f\_status(fct\_surv(fct)) \neq f\_ok \vee (fct\_set \times \{f\_ok\} \not\subseteq fct\_surv; f\_status))) \Rightarrow$
$(\exists fct\_set2 \cdot (fct\_set2 \in \mathbb{P}_1(L\_FUNCT) \wedge (fct\_set2 \times \{f\_ok\} \subset fct\_surv; f\_status))))$

TABLEAU 4.13 – Ecriture du théorème *Post* (fonction).

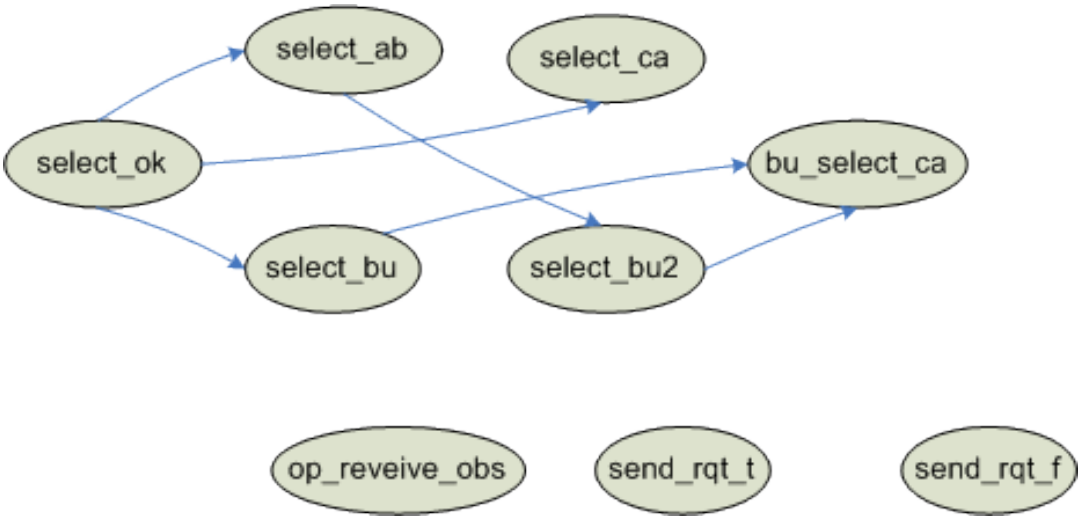

FIG. 4.15 – Synchronisation des événements de la couche opération.

nement normal et anormal (voir section 4.5). On admet que tous les mécanismes de sélection de cette couche sont très sûrs. Par conséquent, on ne considère aucune faute dans cette couche. En réalité, des risques particuliers peuvent affecter ces mécanismes en engageant fortement la sécurité du système. La synchronisation des mécanismes dans cette couche est donnée dans la figure 4.15.

En ce qui concerne l'objectif global de sécurité, il prend tout son sens à partir des observations portées par les opérations. Ainsi, la perte totale du système ou l'événement redouté correspond à un mode opérationnel *m_ cancelled* qu'il convient d'éviter suite à une défaillance simple. La stratégie mise en place permet alors de mieux répondre à cet objectif en proposant des modes dégradés et des interruptions de modes en cas de défaillance simple constatée (tableau 4.14). La post condition de la couche opération (tableau 4.14) ajoute à la post condition de la couche fonction la contrainte d'avoir un mode opéra-

tion soit *ok* (m_ok), soit *avorté* ($m\_aborted$), soit *dégradé* (m_backup), alors le mode opération ne peut être *annulé* ($m\_cancelled$).

$$
\begin{array}{l}
Post \equiv \\
\quad \forall f, fp, fct, fct\_set, op \cdot ((f \in DEFAULT \wedge fp \in \\
UP\_FAILURE \wedge fct \in L\_FUNCT \wedge \\
\quad fct\_set \in \mathbb{P}_1(L\_FUNCT) \wedge (f \mapsto fct \in \\
f\_fault\_class \vee fp \mapsto fct\_set \in f\_struct) \wedge \\
\quad (f\_status(fct\_surv(fct)) \neq f\_ok \vee \\
(fct\_set \times \{f\_ok\} \not\subseteq fct\_surv; f\_status)) \wedge \\
op \in OPERATION \wedge (o\_mode(ope\_mode(op)) = \\
m\_ok \vee o\_mode(ope\_mode(op)) = m\_aborted \vee \\
o\_mode(ope\_mode(op)) = m\_backup)) \\
\Rightarrow \\
\neg(o\_mode(ope\_mode(op)) = m\_cancelled))
\end{array}
$$

TABLEAU 4.14 – Ecriture du théorème *Post* (opération).

Evidemment, cette validation de notre architecture n'est pas la panacée bien que l'intégralité des obligations de preuves soit satisfaite. On se doit de plus de passer cette modélisation générique au crible d'une instanciation des composants pour une représentation plus concrète des modèles.

Chapitre 5

Une interprétation des principes de modélisation et les preuves

Résumé
La modélisation en Event-B a fourni une représentation du comportement d'un système de contrôle. Une interprétation de cette modélisation est réalisée pour évaluer l'applicabilité de la démarche. Le cas d'étude choisi est l'avionique d'un drone autonome. Les preuves à réaliser pour montrer la cohérence des modèles génériques et des raffinements sont effectuées et présentées à l'aide des outils de preuves contenus dans la plateforme de modélisation Event-B appelée Rodin.

« *L'observation recueille les faits ; la réflexion les combine ; l'expérience vérifie le résultat de la combinaison.* »
Denis Diderot

Dans ce chapitre, on se propose de valider expérimentalement l'adéquation des modèles réalisés par rapport aux objectifs applicatifs visés. Pour cela, on interprète notre modélisation générique sur un domaine rassemblant les constituants d'un système de contrôle d'un drone particulier. Puis, on montre comment les modèles génériques peuvent être partiellement raffinés pour traiter le drone considéré.

On présente d'abord le cas d'étude d'un projet de drone autonome (section 5.1). On détaille les différents éléments du système de contrôle de ce drone. Puis, on effectue des interprétations de notre modélisation générique par rapport à ce cas d'étude. Une première interprétation concerne la couche équipement de notre modélisation (section 5.2). De même, on réalise une interprétation de la couche fonction (section 5.3). Puis, on poursuit par une interprétation de la couche opération (section 5.4). Ensuite, on évoque l'expérience d'un raffinement minimal de notre modélisation (section 5.5). Enfin, on expose les résultats des preuves réalisées sur la plateforme Rodin pour décharger les obligations de preuves relatives aux modèles génériques et aux raffinements (section 5.6).

5.1 Cas d'étude d'un drone

Dans le cadre de ses recherches sur le traitement d'information embarqué à bord des systèmes (projet ReSSAC - Recherche et Sauvetage par Système Autonome Coopérant), l'ONERA/DCSD a fait l'acquisition de deux hélicoptères radiocommandés, fabriqués par la compagnie japonaise YAMAHA. Ces appareils ont été dans un premier temps utilisés en radio commande en tant que plates-formes volantes pour des équipements embarqués passifs tels que caméra vidéo, calculateur d'enregistrement et de transmission de données numériques, liaisons numériques, capteurs divers (gyromètres, accéléromètres, GPS, laser, ...). Puis, une nouvelle avionique développée par l'ONERA a été intégrée pour accroître les capacités d'autonomie du drone (figure 5.2). Les vols d'essais se sont effectués sous les autorisations de vol accordées préalablement par la DGAC en respectant certaines exigences de sécurité.

Dans le cadre de missions, les configurations de vols envisagent des modes manuels ou automatiques (figure 5.3). Les phases opérationnelles d'un mode automatique distinguent des phases d'avancement rapide (*Route* ou *Retour base*) et des phases stationnaires ou basses vitesses ; le décollage et l'atterrissage peuvent éventuellement être réalisés en automatique. Ces phases opérationnelles sont détaillées dans la figure 5.4.

En ce qui concerne les fonctions de contrôle implantées dans l'avionique ONERA, elles sont regroupées dans deux modules :

FIG. 5.1 – Drone hélicoptère Rmax.

- le module de "Stratégie" contient les fonctions nécessaires pour gérer le vol, communiquer avec la station sol, et calculer les consignes de navigation ; un extrait du diagramme fonctionnel de navigation fait apparaître des blocs de gestion du vol en rapport avec des phases opérationnelles définissant des objectifs de guidage en termes de consignes de position et de vitesse du drone (figure 5.5) ;
- tandis que le module "Commande" contient la fonction d'estimation de l'état du drone, la fonction de calcul des commandes à envoyer aux actionneurs via l'avionique Yamaha, et la fonction de surveillance. Un extrait du diagramme fonctionnel du pilote embarqué met en évidence des fonctions de pilotage manuel ("Pilote model"), de pilotage en angles de roulis, tangage, lacet ("Pilote RTL") et de pilotage en attitude suivant les angles et variations d'Euler ("Pilote attitude") (figure 5.6).

Concernant les équipements, on s'intéresse plus particulièrement aux capteurs et aux actionneurs de contrôle. Les capteurs concernent essentiellement les capteurs d'attitude tels que les gyromètres, les accéléromètres et les magnétomètres présents dans l'avionique Yamaha, ainsi que des capteurs spécifiques ONERA comme le radar, le laser, le GPS. De même, les actionneurs sont principalement des servomoteurs servant à :

- la commande d'un plateau cyclique par trois vérins électriques YAMAHA ;
- la commande de l'anti-couple et des gaz.

Par ailleurs, les anomalies du système ONERA ReSSAC sont signalées par les alarmes lumineuses sur l'écran de contrôle du pilotage ou détectées par les

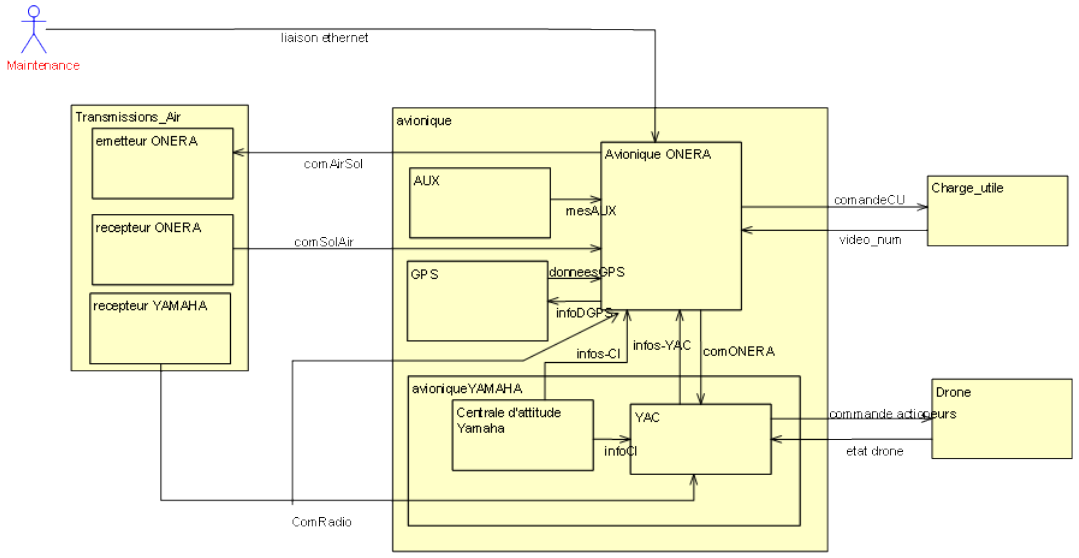

FIG. 5.2 – Diagramme fonctionnel du système ReSSAC.

opérateurs dans le cas de dysfonctionnements algorithmiques entraînant une trajectoire erronée de l'appareil.

Les alarmes opérateur signalant des anomalies du système YAMAHA RMAX sont signalées par les alarmes lumineuses sur l'écran de contrôle du pilotage. Les anomalies de liaison entre le système YAMAHA RMAX et ONERA ReSSAC sont signalées par les alarmes lumineuses sur l'écran de contrôle du pilotage.

De plus, dans la configuration de vol automatique, toute anomalie grave déclenche une alarme sonore pour une reprise en pilotage manuelle immédiate. Dans cette configuration de vol, l'opérateur de navigation surveille la trajectoire suivie par l'appareil. Pour tout écart jugé anormal, il prévient le pilote de sécurité pour une reprise manuelle. Dans le même temps, ce pilote qui observe le mouvement réel de l'hélicoptère peut décider d'interrompre le vol automatique pour toute manœuvre non prévue de l'appareil.

Dans tous les cas d'anomalies, une reprise de contrôle de l'appareil est effectuée en mode manuel par l'opérateur (dit pilote de sécurité) à l'aide de l'émetteur Yamaha.

Les alarmes concernent les anomalies spécifiques suivantes :

- Panne calculateur : une panne générale du calculateur ONERA provoque l'arrêt de la transmission des ordres de commande vers le calculateur YAMAHA. L'avionique YAMAHA reprend alors le contrôle de l'appareil dans les conditions d'un vol radiocommandé.
- Panne GPS : la liaison avec le récepteur GPS n'est plus valide, la position de l'hélicoptère se fait alors par intégration des mesures accéléro-

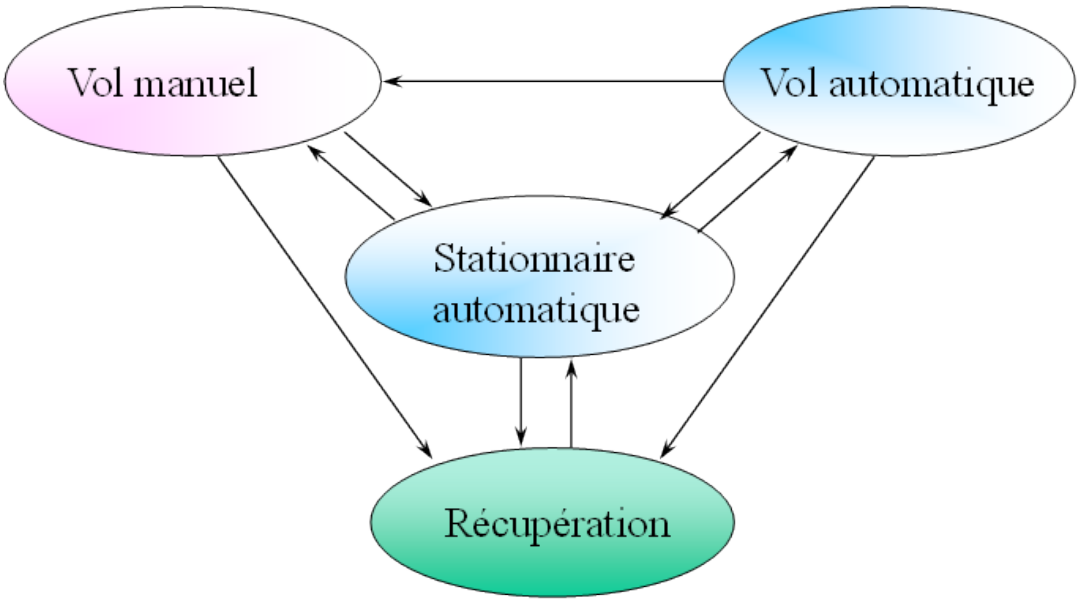

FIG. 5.3 – Configuration de vols.

métriques (navigation à l'estime). La validité de ces informations n'est garantie que sur un court horizon. Le vol automatique doit être interrompu. Ceci est réalisé en avertissant l'opérateur de navigation par une alarme sonore en plus des informations visuelles. Celui-ci informe alors le pilote de sécurité qu'il doit reprendre le contrôle de l'appareil.

– Panne centrale d'attitude : les informations de la centrale d'attitude nécessaires à la stabilisation de l'engin ne sont plus disponibles. Dans l'attente d'une action opérateur, les mesures d'attitude fournies par les inclinomètres du magnétomètre sont utilisées pour entre autre alimenter l'horizon artificiel du poste de pilotage.
– Panne magnétomètre : l'absence du compas magnétique est compensée par l'intégration du gyromètre de lacet. La qualité de la mesure gyrométrique assure une bonne qualité de la mesure de cap pendant plusieurs minutes et ne remet pas en cause la navigation de l'hélicoptère.
– Panne capteurs ONERA : cette panne entraîne la perte des informations des capteurs spécifiques ONERA, comme par exemple le radar, la température du calculateur. Aussi, excepté certaines phases de vol automatique (décollage, atterrissage), cette panne reste sans influence sur le comportement de l'appareil.

Ainsi, on constate que dans le cas de défaillances de certains capteurs, la reconfiguration est assurée non par des capteurs redondants identiques mais par des capteurs équivalents permettant d'avoir des informations rapprochées. De plus, l'exigence de sécurité qui consiste à dire qu'une faute simple ne conduit pas à la perte totale du système, est en grande partie satisfaite par l'omniprésence

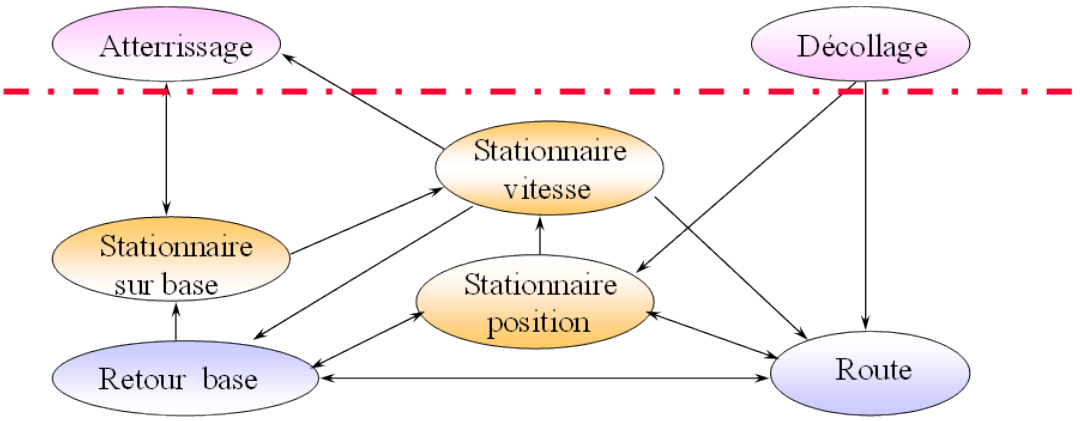

FIG. 5.4 – Vol automatique.

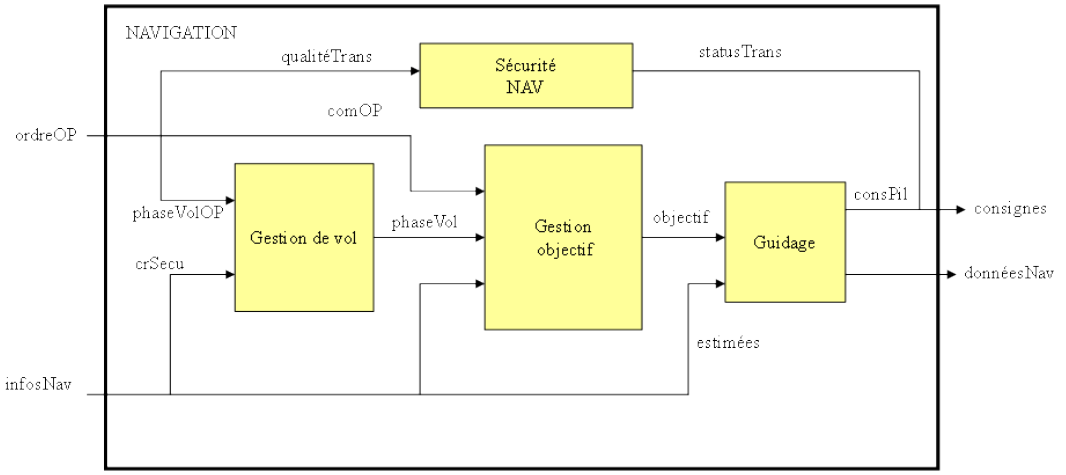

FIG. 5.5 – Diagramme fonctionnel de navigation.

de l'opérateur humain dans les stratégies de vol. Par conséquent, les règles de redondances ou d'équivalences sont moins strictes.

Mais, voyons dans les sections suivantes l'interprétation de ce système avionique dans notre architecture en couches.

5.2 Interprétation de la couche équipement

5.2.1 Interprétation des contextes

On peut associer différents éléments dans les contextes de la couche équipement à des composants physiques dans notre cas d'étude. Ainsi, le tableau

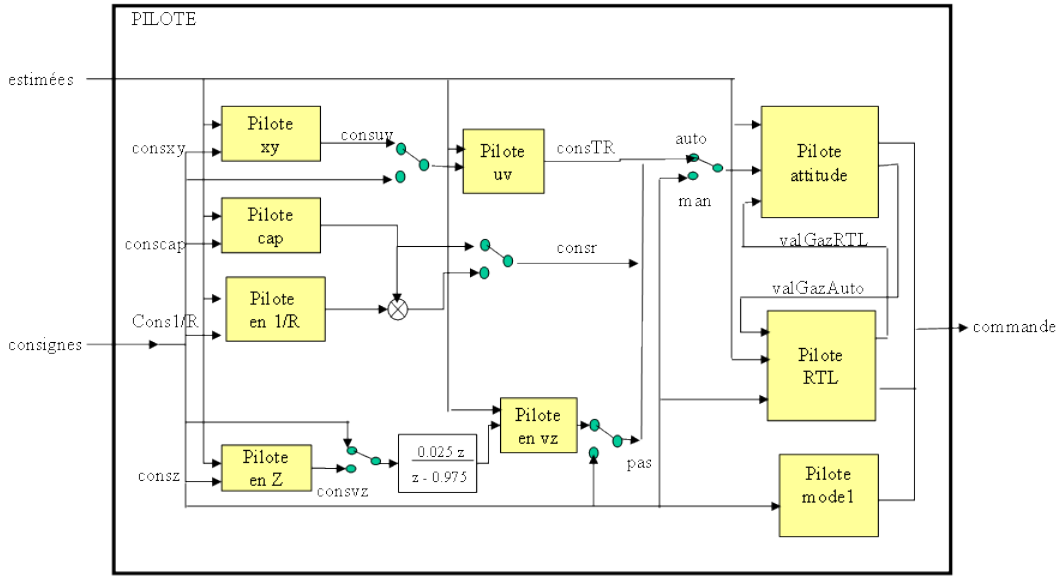

FIG. 5.6 – Diagramme fonctionnel de commande.

suivant (tableau 5.1) fait la correspondance entre la modélisation générique et le système ReSSAC.

Modèles génériques	**Système ReSSAC**
SENSOR	gyromètres, accéléromètres, magnétomètres, radar, laser, GPS, vitesse moteur
ACTUATOR	servomoteurs
DEFAULT	pannes GPS, centrale d'attitude (accéléromètres, gyromètres, magnétomètres), vitesse moteur, radar, laser

TABLEAU 5.1 – Interprétation des contextes équipement

On rappelle que les caractéristiques et les attributs relatifs aux équipements et associés aux ensembles E_MODE, E_SURVEILLANCE et E_OBSERVATION sont respectivement identifiés par id_nom-équipement_mode, id_nom-équipement_statut et id_nom-équipement_obs. Ces identifiants sont contenus et repérés dans la figure 5.2 par :

– *etat drone* en ce qui concerne les informations relatives aux capteurs YAMAHA et aux actionneurs du drone ; ces informations sont ensuite transmises via *Infos-YAC* vers l'avionique ONERA ;

– *mesAUX* pour représenter les données des capteurs additifs ONERA (radar, centrale d'attitude, par exemple) ;
– *donneesGPS* symbolisant les données du GPS.

Enfin, les valeurs propres aux équipements contenues dans VALUE sont directement des valeurs analogiques et numériques manipulées par les différents équipements.

5.2.2 Interprétation des événements

Dans le cas d'étude du ReSSAC, les fonctionnalités d'envoi de mesures ou de réalisation d'une commande sont implicitement représentées lorsqu'on mentionne les équipements concernés.

Ainsi, les activités nominales d'activation (a_activate), d'exécution (a_return, s_send), de communication (enables_obs) et les activités relatives au comportement anormal (detect_err, detect_lost, recover, recover_err) ont été implémentées dans les blocs *AUX*, *GPS* et *Drone* de la figure 5.2.

Les événements mentionnés ci-dessus sont alors gérés par les calculateurs ONERA et YAMAHA.

5.2.3 Interprétation des propriétés

De même, les propriétés de redondance sont implicitement exprimées par les équivalences entre équipements.

La centrale d'attitude ou inertielle est redondée : une centrale ONERA (plus précise) et une centrale YAMAHA.

L'altitude est mesurée par un capteur de pression (YAMAHA) et un laser (ONERA).

Concernant les actionneurs, les servomoteurs ont également été redondés.

5.3 Interprétation de la couche fonction

5.3.1 Interprétation des contextes

On peut associer différents éléments dans les contextes de la couche fonction à des algorithmes dans notre cas d'étude. Ainsi, le tableau suivant (tableau 5.2) fait la correspondance entre la modélisation générique et le système ReSSAC.

On rappelle que les caractéristiques et les attributs relatifs aux fonctions et associés aux ensembles F_MODE, F_SURVEILLANCE et F_OBSERVATION sont respectivement identifiés par id_nom-fonction_mode, id_nom-fonction_statut et id_nom-fonction_obs. Toutes les fonctions du sys-

Modèles génériques	Système ReSSAC
SL_FCT_RES	acquisition, estimation
fct_cmd	pilotes model, RTL, attitude
L_FUNCT	navigation, guidage, sécurité, communication, ...
DEFAULT	pannes liaisons, calculateur ONERA

TABLEAU 5.2 – Interprétation des contextes fonction

tème de contrôle du drone possèdent ces identifiants. Certains de ces identifiants sont contenus et repérés dans les figures 5.5 et 5.6 par :

- *statusTrans* en ce qui concernent les informations de sécurité des fonctions de navigation ;
- *estimées* pour représenter les grandeurs estimées par la fonction d'estimation ;
- *consignes* contenant des modes associés aux fonctions de commande.

Enfin, les valeurs propres aux fonctions contenues dans VALUE sont directement des valeurs numériques manipulées par les différentes fonctions.

5.3.2 Interprétation des événements

D'abord, dans les modèles génériques, les différents événements de communication entre la couche équipement et la couche fonction (receive_obs, enable_obs) traduisent les fonctions de communications dans le cas d'étude et sont contenus dans les blocs *Pilote attitude*, *Pilote RTL*, *Pilote model* (figure 5.6).

De même, les événements d'acquisition entre la couche équipement et la couche fonction (f_acquire, f_acquire_r) traduisent les fonctions d'acquisition contenues par exemple dans les fonctions *estimation*, *Pilote en Z*.

Enfin, les événements d'exécution (f_execute, send_cmd_t) correspondent aux traitements réalisés dans chaque bloc fonctionnel dans le cas d'étude.

5.3.3 Interprétation des propriétés

De même, les propriétés de redondance sont implicitement exprimées par les équivalences entre fonctions (par exemple, les différentes fonctions de pilotage).

5.4 Interprétation de la couche opération

5.4.1 Interprétation des contextes

On peut associer différents éléments dans les contextes de la couche opération à des composants physiques dans notre cas d'étude.

Ainsi, les différents modes d'opérations dans notre modélisation correspondent aux modes d'opération (MODE_OP) présentés dans la figure 5.4 pour le projet ReSSAC, à savoir les modes : vol manuel, vol automatique stationnaire, vol automatique route, vol automatique retour base, décollage, atterrissage.

On rappelle que les attributs relatifs aux opérations et associés à l'ensemble O_MODE est identifié par id_nom-opération_mode. Ces attributs sont représentés en partie dans les blocs *Gestion de vol* et *Gestion objectif* de la figure 5.5.

5.4.2 Interprétation des événements

Dans le cas d'étude du ReSSAC, l'automate de modes présentés dans la figure 5.4 est directement implanté dans le module de stratégie du système avionique ONERA. Ainsi, nos événements de sélection de modes se trouvent interprétés par cet automate.

On constate également que parmi les modes d'opérations, il n'y a pas explicitement de mode d'interruption. De même, le mode annulé qui consisterait à réaliser un atterrissage d'urgence en catastrophe n'est pas implémenté. Le mode dégradé est lui représenté par le mode de retour à la base.

5.5 Raffinement

Une première concrétisation de cette modélisation est réalisée dans le cas du système de contrôle du drone RMAX (projet ReSSAC). Dans cette instanciation, la séquence de modes opérationnels nominaux concerne un décollage *TO_LD*, puis un vol de croisière automatique appelé *CR_auto1* pour lequel l'opérateur fournit des requêtes de haut niveau (position, vitesse), suivi d'un vol de croisière autonome appelé *CR_auto2* pour lequel l'opérateur n'intervient pas (le drone planifie ou suit une séquence d'actions), et enfin un atterrissage appelé *TO_LD*. Ces modes sont associés à une configuration de fonctions de contrôle comprenant des fonctions de pilotage manuelles *MP1,2*, un pilotage automatique grossier *AP1* et un pilotage autonome *AP2* lié à une fonction de navigation *NAV*. Des fonctions spécifiques de pilotage sont identifiées pour le mode de repli dégradé (*P_bu*) et pour le mode d'échec de la mission (*P_ca*). La fonction d'estimation de l'attitude du drone *EST* est liée de façon transverse à toutes les fonctions de pilotage et de navigation. Elle est également connectée aux capteurs (altimètre *ALT*, centrale inertielle *IRS* et *GPS*). L'actionneur

de commande du pas des rotors du drone *SERVO* est lui connecté aux fonctions de pilotage à l'exception de P_ca qui laisse chuter le drone. On note que des redondances fonctionnelles et matérielles sont envisagées pour respecter les exigences de sécurité. Dans la figure 5.7, on décrit l'instanciation des modèles développés : les flèches épaisses représentent les relations inter couches, tandis que les simples flèches indiquent les relations à l'intérieur des couches.

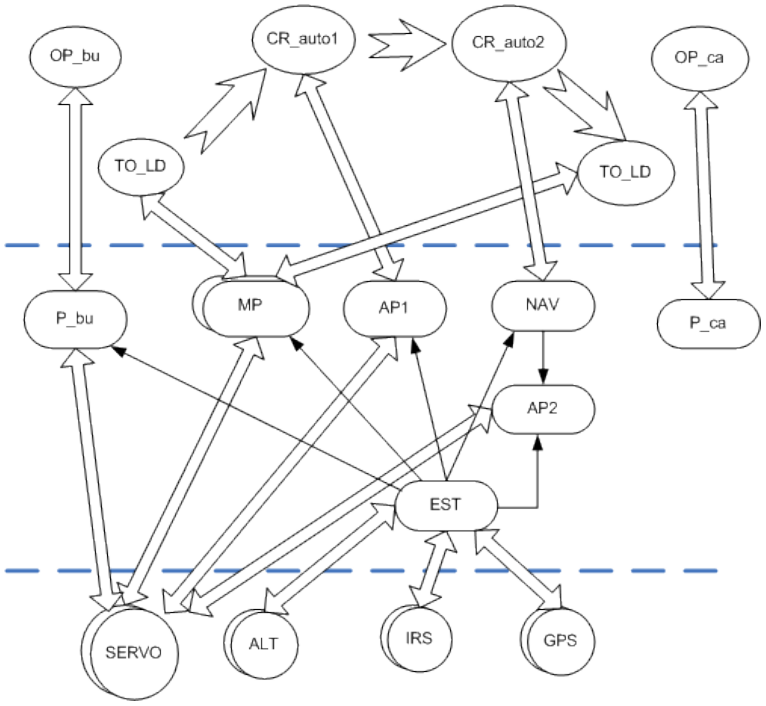

FIG. 5.7 – Application de la modélisation au cas d'étude.

En effet, la concrétisation consiste à instancier l'ensemble des paramètres génériques et à s'assurer que les règles d'obligations de preuves restent satisfaites. Pour cela, on définit en particulier un contexte étendu concret pour lequel :

- l'ensemble énuméré des opérations est décrit par :
  $OPERATION = \{TO\_LD, CR\_auto1, CR\_auto2, OP\_bu, OP\_ca\}$;
- l'ensemble abstrait des fonctions est remplacé par un ensemble énuméré de fonctions :
  $FUNCTION = \{P\_bu, P\_ca, MP1, MP2, AP1, AP2, NAV, EST\} = L\_FUNCT$;
  de même, on définit la sous-liste des fonctions traitant des commandes envoyées aux actionneurs :
  $SL\_FUNCT = \{MP1, MP2, AP1, AP2, P\_bu\}$;
- l'ensemble des équipements est :
  $SENSOR = \{IRS1, IRS2, GPS1, GPS2, ALT1, ALT2\}$ et $ACTUATOR = \{SERVO1, SERVO2\}$.

L'application des modèles génériques proposés dans ce chapitre à notre cas d'étude a permis de vérifier la cohérence et la pertinence de notre modélisation.

5.6 Synthèse de preuves

D'autre part, la vérification des différentes obligations de preuves est réalisée avec la plateforme Rodin[1] qui est un support de la méthode Event-B [59]. Cette plateforme intègre des prouveurs permettant de valider des règles d'obligation de preuves intrinsèques à la modélisation. Ce sont par exemple, des règles de préservation des invariants ou des règles de faisabilité d'une action d'initialisation.

Le tableau 5.3 résume le nombre de preuves déchargées i.e. validées, automatiquement et interactivement avec Rodin. Pour les références 1, 3, 6, 7, 11 et 15, il a suffi de changer l'option de preuve en sélectionnant le prouveur adapté pour terminer les preuves. La nature des preuves réalisées concerne principalement les règles de préservation des invariants de typage et quelques règles de correction des raffinements.

Dans les références 5 et 12 pour les invariants 1 à 3 de la section 4.3 et 5 à 8 de la section 4.4, les décharges de preuves ont été plus compliquées. Dans ces cas, la nature des preuves réalisées concerne principalement les règles de préservation de ces invariants de sécurité. Ces invariants à prouver nécessitent de raisonner sur le cardinal d'ensembles d'objets redondants (équipements, fonctions).

Ainsi, les hypothèses sur l'existence du nombre minimal de redondances fonctionnelles requises ont été formalisées en introduisant un ensemble minimal de 4 fonctions L_FUNCT_min, inclus dans l'ensemble des fonctions L_FUNCT (voir contexte UAV_CF2 dans le modèle intitulé UAV_CF2n en annexe A.3) ; de plus, ces fonctions sont déclarées regroupées en paires de redondance disjointes. Ces hypothèses nous permettent de conduire des preuves pour des patrons d'architecture (avec duplication). D'autres hypothèses permettraient de raisonner sur d'autres patrons.

Par ailleurs, quelques statistiques sur les modèles réalisés montrent les efforts consacrés à cette modélisation et le volume de travail accompli, la qualité du travail étant estimée par sa validation (voir tableaux 5.4).

Pour conclure, on constate que la totalité des preuves a été déchargée, ce qui conforte notre confiance dans la pertinence et la correction de notre modélisation[4] en Event-B.

[1] Voir `http ://www.event-b.org`.

[2] Pour les preuves relatives aux invariants sur la redondance, le temps peut excéder 30 min, à 100%.

[3] Z, AltaRica, réseaux de Petri, UML, eFFBD.

[4] Plus de détails sur les modèles sont donnés en annexe A.3.

Réf.	**Raffinement**	**Automatique**	**Interactive**	**Total**
1	EQUIP_0	4	1	5
2	EQUIP_1	12	0	12
3	EQUIP_1_1	23	1	24
4	EQUIP_2	9	0	9
5	EQUIP_3	3	12	15
6	EQUIP_4	5	1	6
7	E_FCT_1	23	3	26
8	E_FCT_2	14	1	15
9	FUNCT_0	13	0	13
10	FUNCT_1	6	0	6
11	FUNCT_1_1	27	1	28
12	FUNCT_T2	3	17	20
13	F_OPE_1	15	0	15
14	F_OPE_2	19	0	19
15	F_OPE_3	50	2	52
	TOTAL	226	39	265

TABLEAU 5.3 – Preuves dans la modélisation

	Modèles Event-B		
	à 50% (équipement + fonction en partie)	**à 90%** (équipement + fonction)	**à 100%** (3 couches)
Temps de preuve[2]	$< 10s$	$< 1min$	$< 5min$

	Modèles précurseurs	**Modèles présentés**
Nombre de versions	10 (Event-B) 10 (autres[3])	12
Temps de conception	20%	50%

TABLEAU 5.4 – Statistiques sur la modélisation

Chapitre 6

Des travaux similaires

Résumé
Ce chapitre présente quelques formalismes susceptibles d'enrichir notre modélisation par l'apport de nouveaux points de vue. Puis, il propose un état de l'art succinct des modélisations ayant trait aux architectures en couches.

« *Gare aux simples points de vue. Souvent, un point de vue équivaut à ne point avoir de vue ou à ne voir qu'un seul et unique point.* »
Yves Breton - Bâtir sa communauté : un cadre et des pistes

De quoi s'agit-il dans cette thèse de recherche ? Quels sont les travaux existants similaires ? Rappelons que cette recherche a pour objectif, entre autres, de formaliser des architectures de sécurité de systèmes autonomes. Cette formalisation passe par la définition des architectures de sécurité et par le choix d'un formalisme de modélisation. Dans notre approche, nous avons apporté une modélisation générique de la problématique pour tous les systèmes de contrôle capables d'autonomie. Ainsi, notre modélisation est applicable aussi bien à des systèmes de contrôle d'attitude et d'orbites de satellites qu'à des systèmes de contrôle moteur ou d'assistance à la conduite dans l'automobile. Mais, en quoi nos travaux se distinguent-ils des autres études du domaine ?

Les choix relatifs à cette thèse concernent tout particulièrement le formalisme et la modélisation d'un système critique donné. Choisir Event-B s'est alors avéré adapté pour **modéliser** :

- formellement,
- génériquement,
- une architecture en couches
- tolérante aux fautes

et pour **valider** la proposition spécifiée.

Cependant, quels sont les autres formalismes possibles (section 6.1) ? Quels sont les autres travaux de modélisation (avec preuves) existant et ayant des objectifs similaires (section 6.2) ?

C'est ce que nous verrons dans les sections suivantes.

6.1 Formalismes alternatifs

6.1.1 AltaRica

▶ Principe du formalisme

Dans le domaine SdF, AltaRica est un formalisme spécialisé dans l'analyse de la propagation de défaillances. AltaRica est un langage formel développé au Laboratoire Bordelais de Recherche en Informatique (LaBRI) ayant pour but une modélisation compositionnelle et hiérarchique du comportement fonctionnel et dysfonctionnel de systèmes complexes [60; 61]. Chaque composant du système est modélisé par un nœud (*node*) (voir figure 6.1), qui est un automate de modes constitué de trois parties [62].

La première partie sert à déclarer différents paramètres : état (*state*), flux de données (*flow*) et événement (*event*). Les états sont des variables internes symbolisant le mode de fonctionnement du nœud (nominal ou dysfonctionnel). Les flux décrivent les données d'entrée et de sortie du composant. Les événe-

ments représentent des phénomènes déclenchant des transitions depuis un état courant vers un état sollicité.

La seconde partie du nœud décrit l'initialisation des états et les transitions. Une transition est un t-uplet $g \xrightarrow{evt} e$, avec g étant la garde de la transition, evt le nom d'un événement déclencheur et e la conséquence de la transition. La garde s'exprime par une formule booléenne sur les variables d'état et de flux. Elle définit ainsi une précondition dans laquelle la transition est tirable si un événement concerné apparaît.

La troisième partie du nœud concerne un ensemble d'assertions qui établissent des relations invariantes entre les variables du composant et elles déterminent la valeur des variables de sortie en fonction des variables courantes d'état et d'entrée.

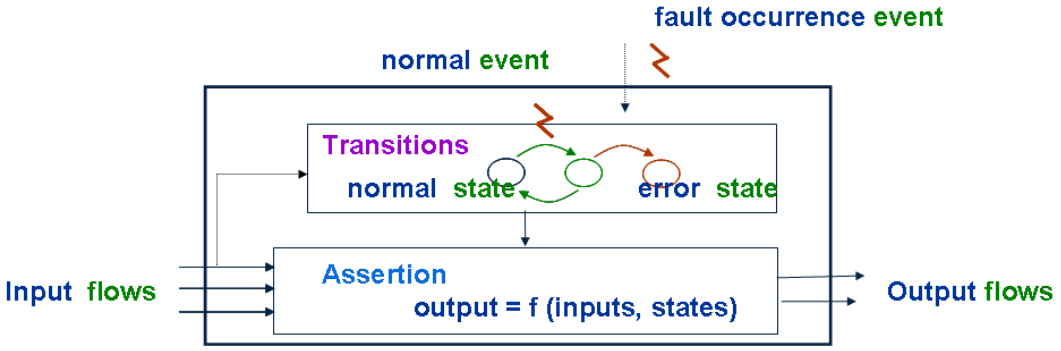

FIG. 6.1 – Composant interfacé en AltaRica.

▶ Avantages et inconvénients de l'expressivité du formalisme

Les points forts et les points faibles du langage AltaRica résident dans :
+ la compositionnalité du langage ;
− l'absence de généricité des modèles ; la modélisation d'une architecture en couches tolérante aux fautes est nécessairement instanciée en AltaRica.

▶ Avantages et inconvénients des outils de calcul disponibles

Du point de vue des calculs de SdF, les points forts et les points faibles du langage AltaRica résident dans :
+ l'évaluation automatique pour toutes les gammes de propriétés de la SdF : évaluation qualitative et évaluation quantitative.

► **Exemple d'application**

Une illustration d'une modélisation détaillée en AltaRica est donnée en annexe B en prenant comme support notre cas d'étude. AltaRica apporte à notre modélisation un arsenal d'outils d'analyse de sécurité qualifiés, ainsi qu'une possibilité de simulation de la propagation des défaillances.

6.1.2 SCR

► **Principe du formalisme**

SCR (pour Software Reduction Cost) est une méthode formelle de spécification utilisant une notation spécifique des exigences à l'aide de tables manipulées par des outils d'analyse [63]. SCR est basé sur les concepts de Parnas [64] qui décrit le comportement attendu d'un système à partir de relations mathématiques entre quatre types de variables :

- les variables contrôlées sont des entités de l'environnement contrôlées par le système ; ce sont par exemple le niveau sonore ou la vitesse du vent qui sont modifiés par l'action des rotors d'un hélicoptère ;
- les variables supervisées sont des grandeurs physiques de l'environnement observées ou mesurées par le système ;
- les variables d'entrée et de sortie sont les pendants des variables précédentes ; elles sont directement produites et consommées par le système et sont associées à des composants du système.

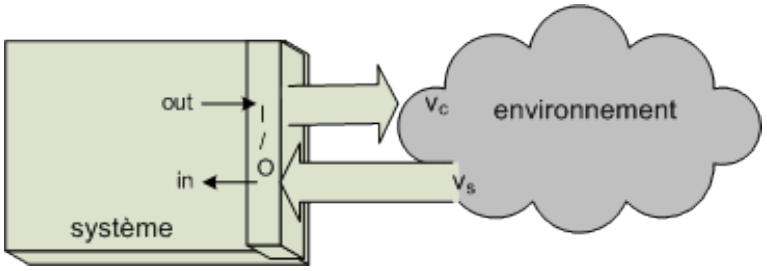

FIG. 6.2 – Variables de Parnas.

Par la suite, seulement deux types de variables, contrôlées et supervisées, sont considérés et on établit des relations entre ces variables dont deux relations importantes pour la spécification du système :

- NAT traduit des contraintes naturelles imposées par des lois physiques et environnementales ;
- tandis que REQ définit des exigences spécifiées sur le comportement du système par des relations entre les variables contrôlées et supervisées.

Les spécifications tabulaires de SCR décrit les relations NAT et REQ du modèle de Parnas en termes de variables contrôlées et supervisées, de classes

de modes et des termes (désignant les variables internes du système). C'est principalement la construction des classes de modes correspondant à des modes d'opérations du système qu'il convient de comparer avec notre architecture en couches dans le cadre de la tolérance aux fautes.

mode n°	**mode en cours**	**événement**	**mode modifié**
1	*m_init*	@S(initialisation)	*m_ok*
2	*m_ok*	@F(failure)	*m_backup*
3	*m_ok*	@S(failure1) AND @S(failure2)	*m_cancelled*

TABLEAU 6.1 – Table de transition de modes en SCR

▶ Avantages et inconvénients de l'expressivité du formalisme

De même que AltaRica, les points forts et les points faibles de SCR résident dans :

+ la compositionnalité des tables ;
− l'absence de généricité des modèles ; la modélisation d'une architecture en couches tolérante aux fautes est nécessairement instanciée en utilisant une sémantique d'une machine à états finis associée aux tables SCR.

▶ Avantages et inconvénients des outils de preuve disponibles

Comme Event-B, SCR dispose d'outils de validation de propriétés invariantes décrivant les relations NAT et REQ. Cependant, un avantage de SCR concerne la prise en compte des aspects temporels dans les relations grâce à l'établissement d'une librairie d'opérateurs tel que l'opérateur DUR. Par exemple, le prédicat $DUR(c) = k$ est satisfait à l'instant i, si à l'instant i la condition c est vraie et est vraie depuis exactement k unités de temps [65].

▶ Exemple d'application

Dans l'article [65], les auteurs proposent une modélisation SCR en conservant une approche standard qui formalise la relation entre le comportement global tolérant aux fautes et le comportement nominal souhaité que l'on cherche à maintenir et à préserver pour des systèmes critiques sûrs de fonctionnement. Cette relation correspond à la notion de raffinement partiel qui n'établit une correspondance qu'entre la composante nominale du système global tolérant aux fautes et la représentation idéale (sans faute) de ce même système.

D'après [65], SCR permet de spécifier le comportement en dysfonctionnement en identifiant les fautes associées aux différents équipements puis en

décrivant le comportement global incluant les mécanismes utiles pour la tolérance aux fautes. Une illustration est donnée dans cet article par la modélisation du dispositif avionique de gestion de l'altitude.

La formalisation de la relation entre le comportement global tolérant aux fautes et le comportement idéal souhaité correspondant au raffinement partiel conforte notre modélisation qui distingue à travers des modes et des états de surveillance un comportement nominal et un comportement dysfonctionnel du système.

6.1.3 PVS

▶ Principe du formalisme

D'après [66], PVS (pour Prototype Verification System) qui est un langage de spécification intégré avec un assistant de preuve de théorèmes développé par le laboratoire SRI International aux Etats-Unis, a aidé des équipes de la NASA à spécifier des plateformes de calculateurs très fiables. Le langage de spécification et les outils d'analyse de PVS ont permis de corriger de nombreuses erreurs de conception de systèmes critiques.

PVS est basé sur la logique d'ordre supérieur (HOL : High Order Logic) et sur l'utilisation accrue de types (par exemple, les nombres entiers, réels, des structures *enregistrements*) dont la cohérence est assurée par un efficace vérificateur de types (*typechecking*) [67]. Une spécification PVS est définie comme un ensemble de *théories* caractérisées par un nom symbolisant un nouveau type, des constantes, des propriétés et des opérations faisant évoluer des variables. Une illustration décrivant une théorie *Specify* ayant comme paramètre t de type *non vide* ($TYPE+$), et des constantes booléennes a,b,c ainsi qu'un théorème *Property* portant sur ces constantes, est donnée dans l'algorithme 6.1 :

Algorithme 6.1 Exemple de théorie PVS

```
Specify [t : TYPE+] : THEORY
BEGIN
    a,b,c : bool
    Property : THEOREM (a AND b IMPLIES c)
END Specify
```

▶ Avantages et inconvénients de l'expressivité du formalisme

L'expression de théories PVS permet de mettre en évidence des similitudes avec la méthode B :

- les ensembles *SETS* deviennent des types *TYPES* ;

- on utilise des axiomes et des théorèmes pour définir une opération (semblable à une *THEORY* en PVS).

Toutefois, l'expressivité de PVS est supérieure celle de Event-B. Par conséquent, la généricité de modèles est également réalisable avec PVS.

► **Avantages et inconvénients des outils de preuve disponibles**

Par ailleurs, PVS dispose de prouveurs très performants ce qui en fait un outil adapté à notre modélisation.

Un outil particulier PBS développé par César Muñoz [68] transforme une pseudo machine abstraite B en théorie PVS (voir figure 6.3). On bénéficie ainsi d'une méthodologie B enrichie d'un langage de spécification fortement typé et d'un prouveur de théorème puissant (PVS) [69; 70].

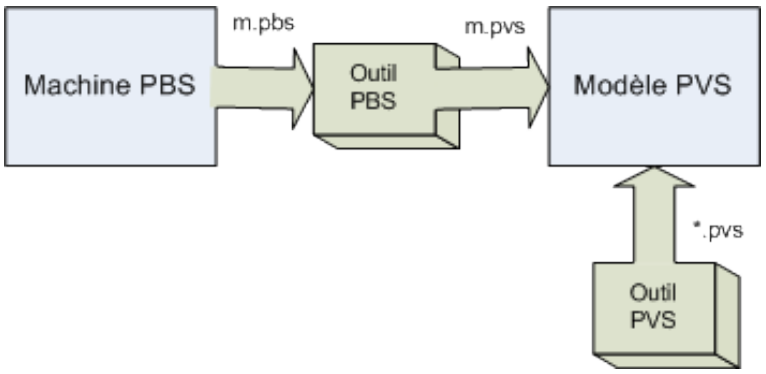

FIG. 6.3 – Transformation avec PBS.

Dans notre étude, PVS ou PBS pourrait être utilisé en phase plus avancée pour décrire plus précisément les algorithmes de détection d'erreurs, par exemple.

6.2 Autres modélisations formelles d'architectures

6.2.1 Modélisations d'une architecture générique en couches

Une modélisation d'une architecture en couches d'un système de contrôle est également proposée en B classique par [36]. Cette modélisation suggère une couche supérieure supplémentaire de communication avec un opérateur humain. Vu du niveau opérateur (plus haut niveau), le système de contrôle/commande se comporte de la façon suivante :

- initialement “dormant”, il devient actif quand il reçoit une requête ;
- en mode actif, il exécute de façon autonome des opérations jusqu'à complète exécution de la requête ;
- il retourne un accusé pour chaque exécution correctement achevée, et il redevient inactif.

Une amélioration de cette approche est apportée en introduisant des modes opérationnels permettant de configurer les couches de haut niveau de l'architecture étudiée [42]. Dans [42], les modes opérationnels sont caractérisés par des propriétés fonctionnelles à respecter, appelées *garanties*, dans des conditions de fonctionnement, appelées *hypothèses*. Les mécanismes de tolérance aux fautes apportent alors un procédé permettant de distinguer des comportements nominaux, caractérisés par des modes nominaux, et des comportements défaillants, caractérisés par des modes anormaux ou erronés. Après un partitionnement des modes, on définit des modes de reconfiguration associés à des modes nominaux. Des mécanismes de détection d'erreur sont activés en permanence et interviennent dans les hypothèses des modes de reconfiguration. Par le raffinement, [42] met en évidence le procédé permettant de passer d'un comportement normal à un comportement défaillant en mettant en jeu les garanties et les hypothèses précédemment définies.

Ces approches constituent une stratégie similaire à la nôtre. L'approche de [36] comprend également un algorithme de FDIR rendant le système tolérant aux fautes. Seulement, dans ces approches, les aspects de sécurité globale du système sont omis étant donné que la redondance d'équipements n'est pas prise en compte. De plus, les propriétés vérifiées concernent uniquement la correction de la synchronisation des différentes tâches. Les propriétés intrinsèques aux couches et inter couches ne sont pas spécifiées. Ces modélisations représentent donc un comportement sûr d'un système de contrôle, mais ne permettent pas de juger de la correction du système ainsi réalisé par rapport à des propriétés de tolérance aux fautes.

6.2.2 Modélisation d'une architecture instanciée dans le spatial

Dans le domaine spatial, l'architecture en couches est également de rigueur dans les systèmes de contrôle d'attitude et d'orbites (également appelés SCAO). La figure 6.4 présente une architecture de satellite tolérante aux fautes, car elle favorise l'intégration de stratégies de FDIR [71]. En fait, la FDIR fait l'objet de nombreuses études dans le spatial, notamment pour les vols en formation [72; 73]. Dans [72], une stratégie considère des mécanismes FDIR distribués sur plusieurs satellites de la formation. Cette stratégie a été modélisée avec les réseaux de Petri combinés au langage Z.

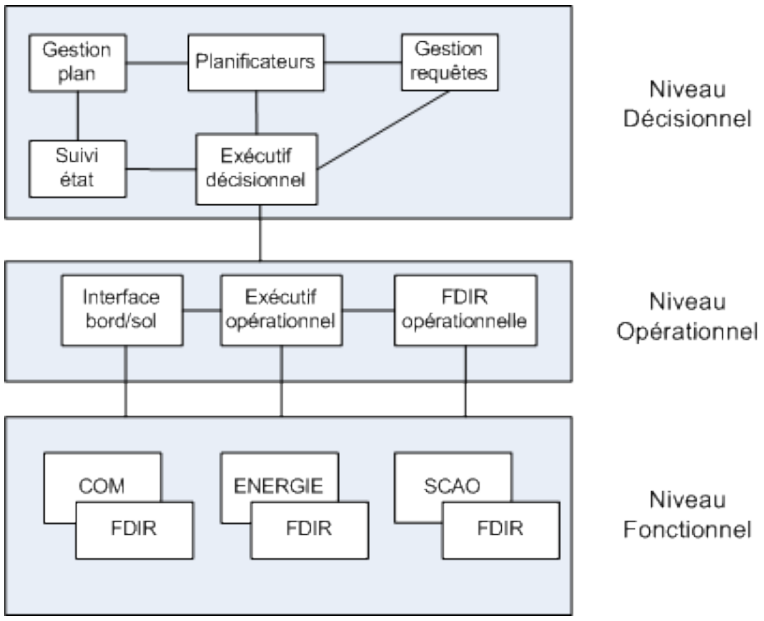

FIG. 6.4 – Architecture en couches du système de contrôle-commande d'un satellite.

La modélisation d'une architecture en couches à l'aide des réseaux de Petri et de Z a permis d'exprimer formellement des propriétés de sécurité. Cependant, en raison de l'absence d'outils de preuves combinant les deux formalismes réseaux de Petri et Z, la validation de ces propriétés n'a pas été réalisée [74].

L'étude de vols en formation a également permis d'identifier des modes opérationnels dans le traitement de la sécurité lorsque les satellites intègrent des capacités d'autonomie : modes nominaux, modes fail-safe, mode fail-operationnal (voir figure 6.5).

6.2.3 Modélisation d'une architecture instanciée en robotique mobile

Notre architecture en couches est également inspirée des architectures en couches largement utilisées en robotique mobile [33]. En effet, la structuration en couches apporte de nombreuses facilités pour intégrer de nouvelles fonctionnalités dans le système, notamment les fonctionnalités relatives à l'autonomie (voir figure 6.6). Par l'association du planificateur et du superviseur, on met alors en place un mécanisme efficace de délibération et de réaction. De plus, le LAAS a développé des outils facilitant la conception de modules dans chaque couche de l'architecture :

- *IxTeT* est un planificateur temporel associé à un outil de supervision, *PRS*,
- *Kheops* est un contrôleur d'exécution,

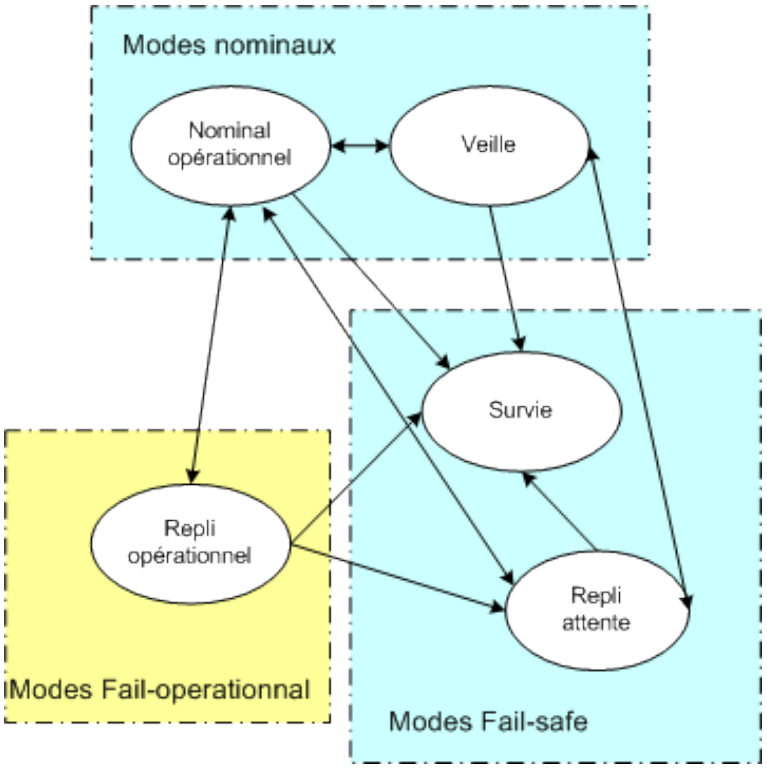

FIG. 6.5 – Modes d'opérations d'un satellite.

– $G^{en}oM$ permet le développement de nouvelles fonctions.

Par ailleurs, le comportement des fonctions de contrôle est rendu plus sûr en associant à $G^{en}oM$ une chaîne d'outils développés par $Verimag$ [75] : *Behaviour Interaction Priorities* (ou BIP) et *D-Finder*. Ces outils permettent de spécifier et de valider des propriétés (dont des invariants) intrinsèques à un composant fonctionnel ainsi que d'autres propriétés entre des composants, tout comme dans notre modélisation Event-B.

Cependant, un point faible de cette spécification fonctionnelle reste le risque d'explosion combinatoire dans la modélisation, qui est néanmoins limité par une construction compositionnelle autour de composants déterminés. De plus, les propriétés à valider n'exploitent pas la complexité de la logique des prédicats du premier ordre comme en Event-B, par contre elles intègrent bien des aspects temporels [75].

Enfin, la généricité de la modélisation est plus assurée avec Event-B qu'avec BIP.

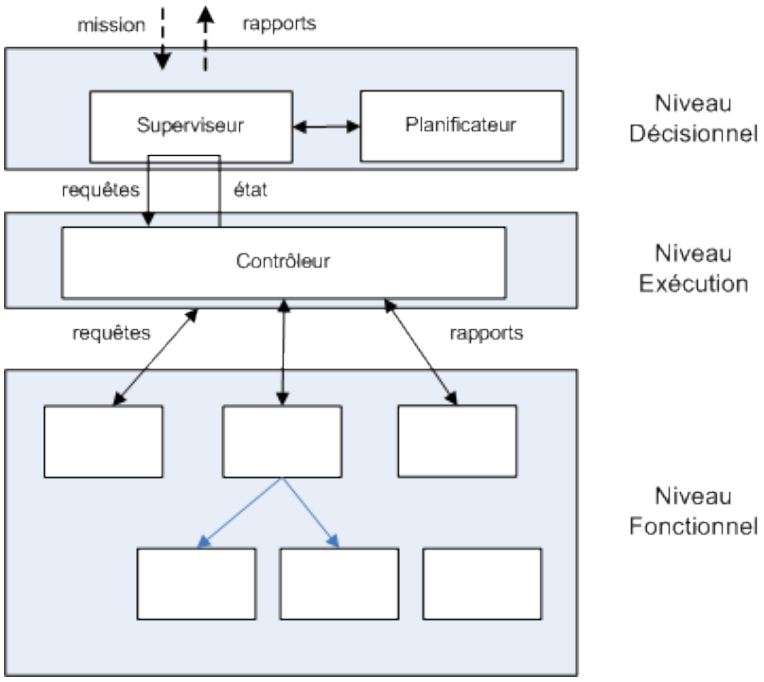

FIG. 6.6 – Architecture en couches du LAAS.

En bref L'architecture en couches fonctionnelles est une représentation abstraite déjà répandue et adaptée à l'étude de sécurité des systèmes de contrôle autonomes. Une formalisation de cette architecture permet surtout de consolider les propriétés structurantes au niveau des couches. Les formalismes utilisés pour la modélisation de systèmes à événements discrets révèlent à leur manière différentes spécificités de l'architecture en couches.

Conclusion

Dans la première partie de cette thèse, nous avons décrit le contexte des systèmes de contrôle autonomes critiques, ainsi que les concepts et les méthodes de sûreté de fonctionnement (SdF) vis-à-vis de la conception de ces systèmes. Nous avons ainsi constaté que la complexité de ces systèmes et le niveau de sécurité à atteindre rendent nécessaires une vérification et une validation (V & V) rigoureuse des propriétés de sécurité de ces systèmes. L'efficacité de cette V & V formalisée repose en partie sur la prise en compte au niveau de la conception des systèmes, des mécanismes de SdF associés à des architectures de sécurité éprouvées.

Par la suite, nous avons alors présenté les principales propositions de la littérature dans le cadre des techniques de tolérance aux fautes. Les techniques présentées distinguent des mécanismes de sécurité à prépondérance logicielle, telles que des mécanismes de diagnostic de systèmes continus ou discrets par rapport à d'autres mécanismes orientés vers le matériel comme la mise en œuvre de redondances de composants et des reconfigurations matérielles. L'ensemble de ces techniques forment une méthode de tolérance aux fautes appelée la FDIR (Fault Detection Isolation and Reconfiguration). Nous nous sommes alors inspirés de ces techniques ainsi que des architectures développées pour des engins mobiles autonomes pour proposer une architecture en couches fonctionnelles. Nous nous sommes efforcés de montrer que cette architecture permet de de modéliser des mécanismes de sécurité génériques et de prouver les propriétés de sécurité génériques requises pour un système de contrôle autonome critique.

Pour cela, nous nous sommes intéressés à sa spécification formelle en Event-B. Event-B offre un cadre rationnel de modélisation et de développement adapté aux systèmes complexes. Event-B a été retenu car un des objectifs de la thèse était de mettre en évidence les caractéristiques génériques d'une architecture qui permettent de tenir des exigences de sécurité. Event-B permet en effet d'exprimer des propriétés génériques comme l'existence de redondances sans pour autant expliciter la liste des composants particuliers de la redondance. Des modèles formels alternatifs comme AltaRica ne permettent de raisonner que sur des architectures complètement instanciées.

La formalisation Event-B du système étudié que nous avons réalisée a mis en œuvre le principe de raffinement dans la méthode appliquée à l'architecture en couches adoptée. Nous avons spécifié le mode de fonctionnement sûr du système en détaillant les mécanismes de sécurité associés à la propagation de défaillances à l'intérieur des couches et entre les couches. Puis, nous avons évalué qualitativement les garanties de sécurité offertes par chaque couche par rapport à un événement redouté : la perte du système complet. Nous rappelons dans ce qui suit les principaux résultats de notre démarche de spécification et de validation de l'architecture étudiée.

1. **La spécification d'une architecture générique en couches fonctionnelles.** La généricité de notre architecture permet d'étendre son application à de nombreux systèmes de contrôle. Par ailleurs, nous nous sommes basés sur une architecture en couches développée pour des systèmes autonomes. Nous avons proposé une formalisation de cette architecture en Event-B. Lors de la formalisation, nous avons mis l'accent sur le principe d'interaction entre les couches et sur les propriétés caractéristiques des couches. Nous avons également montré comment se propagent les défaillances dans le système. Nous avons alors appliqué des modes opérationnels afin de prendre en compte cette propagation et den mitiger les effets.

2. **Une méthode d'évaluation qualitative basée sur un raffinement ascendant ("bottom-up").** Pour valider l'exigence de sécurité exprimant qu'*une faute simple ne peut conduire à la perte totale du système*, nous nous sommes d'abord intéressés à la structuration et au fonctionnement de la couche de plus bas niveau, à savoir la couche équipement. Puis, l'interaction entre les couches nous a conduits à dériver cette exigence sur les autres couches supérieures. La validation de cette exigence a été rendue possible par l'application de la logique de Hoare en définissant des théorèmes dans chaque couche.

Dans cette thèse, le travail effectué a donc permis de montré l'intérêt d'une formalisation d'une architecture en couches. Cependant, nous nous sommes concentrés sur l'étude d'une exigence de sécurité qualitative. D'autres voies exploratoires peuvent alors être envisagées, comme :

1. **l'étude des exigences de sécurité quantitatives à l'aide d'une méthode formelle telle que Event-B.** Dans ce cas, l'apport d'une formalisation ne serait pas dans le mode de calcul mais pourrait être utile pour valider un comportement instancié à l'aide de propriétés associées à des événements combinés à des probabilités, comme dans les graphes de Markov.

2. **l'intégration des facteurs humains dans la modélisation**. Cet aspect se traduirait par l'étude du comportement de l'opérateur. En effet, l'opérateur interviendrait au niveau supérieur à la couche opération. Les conditions d'activation des événements de sélection des modes opération seraient renforcées par des propriétés caractéristiques d'un opérateur.

3. **l'application de notre modélisation à d'autres études de cas, dans le domaine spatial ou automobile.** Nous adapterions le modèle en fonction du degré d'autonomie recherché.

4. **la prolongation de la modélisation jusqu'à des algorithmes implémentables.** Notre modélisation contient des spécifications relatives à la fois au matériel et au logiciel. Par conséquent, nous pourrions poursuivre la modélisation en Event-B pour faire apparaître des algorithmes qui seraient d'abord implémentés sur un simulateur de drone.

Annexe A

Modélisation globale en Event-B

A.1 Synthèse des raffinements

Cette annexe représente une synthèse des différents modèles réalisés dans notre étude. Le premier modèle de la couche équipement *EQUIP_0* donne une vue simpliste du comportement nominal d'un capteur et d'un actionneur. Un premier raffinement de ce modèle introduit l'aspect communication et précise l'état de fonctionnement des équipements. Le raffinement suivant *EQUIP_1_1* complète la description du comportement nominal et de l'aspect de communication, et intègre un comportement en présence de fautes. Dans cette couche, les machines *EQUIP_2* et *EQUIP_3* précisent le traitement des défaillances. Par ailleurs, un autre raffinement de *EQUIP_1_1* constitue une interface avec la couche fonction (*E_FCT_1*) et indique les événements de contrôle de cette couche par rapport à la couche équipement. Le raffinement suivant *E_FCT_2* complète ce contrôle en vérifiant la validité de données provenant des équipements. Une conséquence de ce contrôle (*EQUIP_4*) est la reconfiguration d'un équipement liée à une défaillance non détectée au niveau équipement. De façon similaire, le comportement des fonctions est décrit de façon nominal puis intègre l'éventualité de défaillances au fur et à mesure des raffinements en y associant le principe de communication dans la machine *FUNCT_1_1*. Enfin, la couche opération débute par une machine interface contenant des activités des fonctions, raffinée à deux reprises pour spécifier la gestion de la mission et la gestion de la configuration des modes opérationnels.

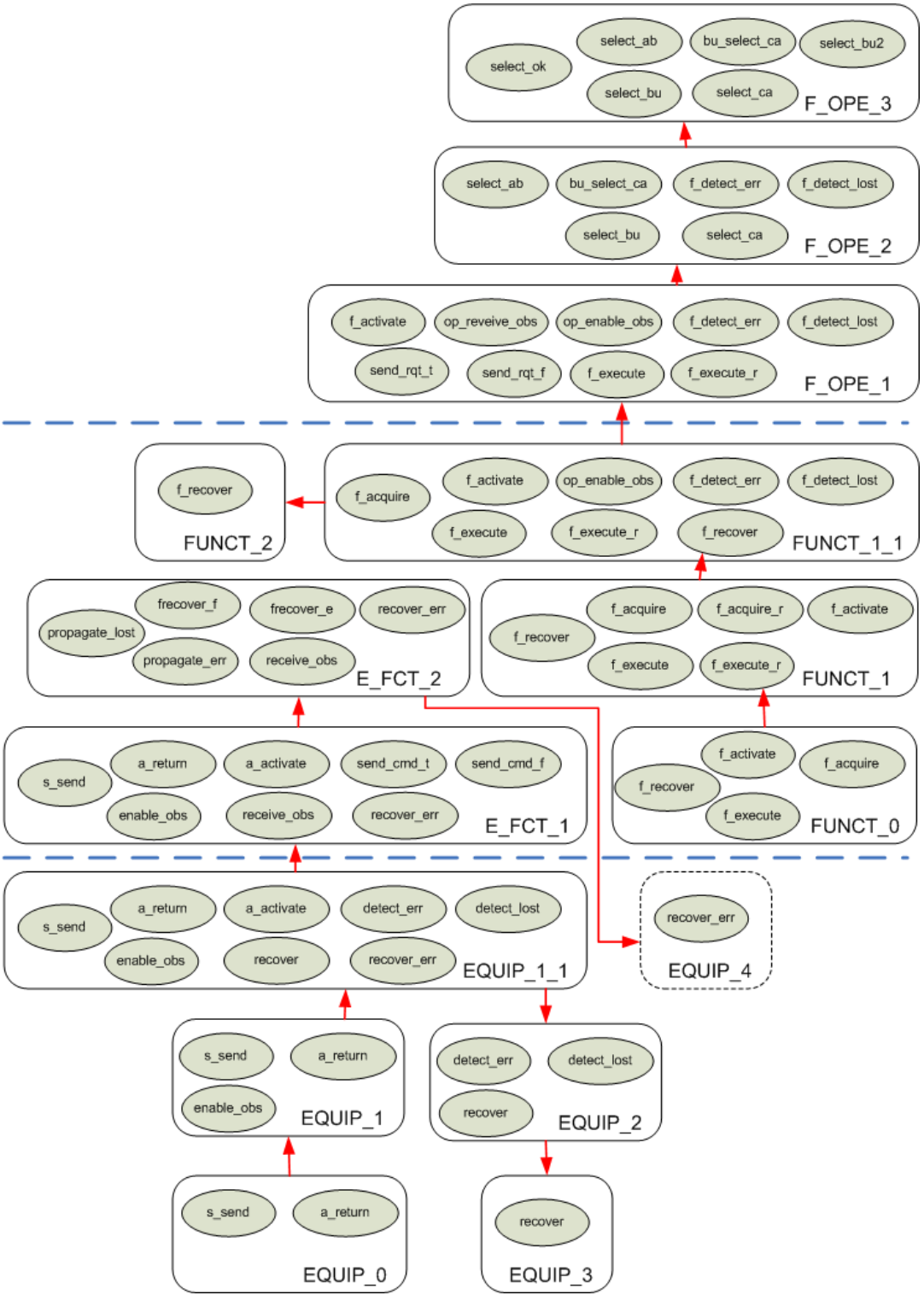

FIG. A.1 – Synthèse des raffinements de la modélisation.

A.2 Synchronisation globale des événements

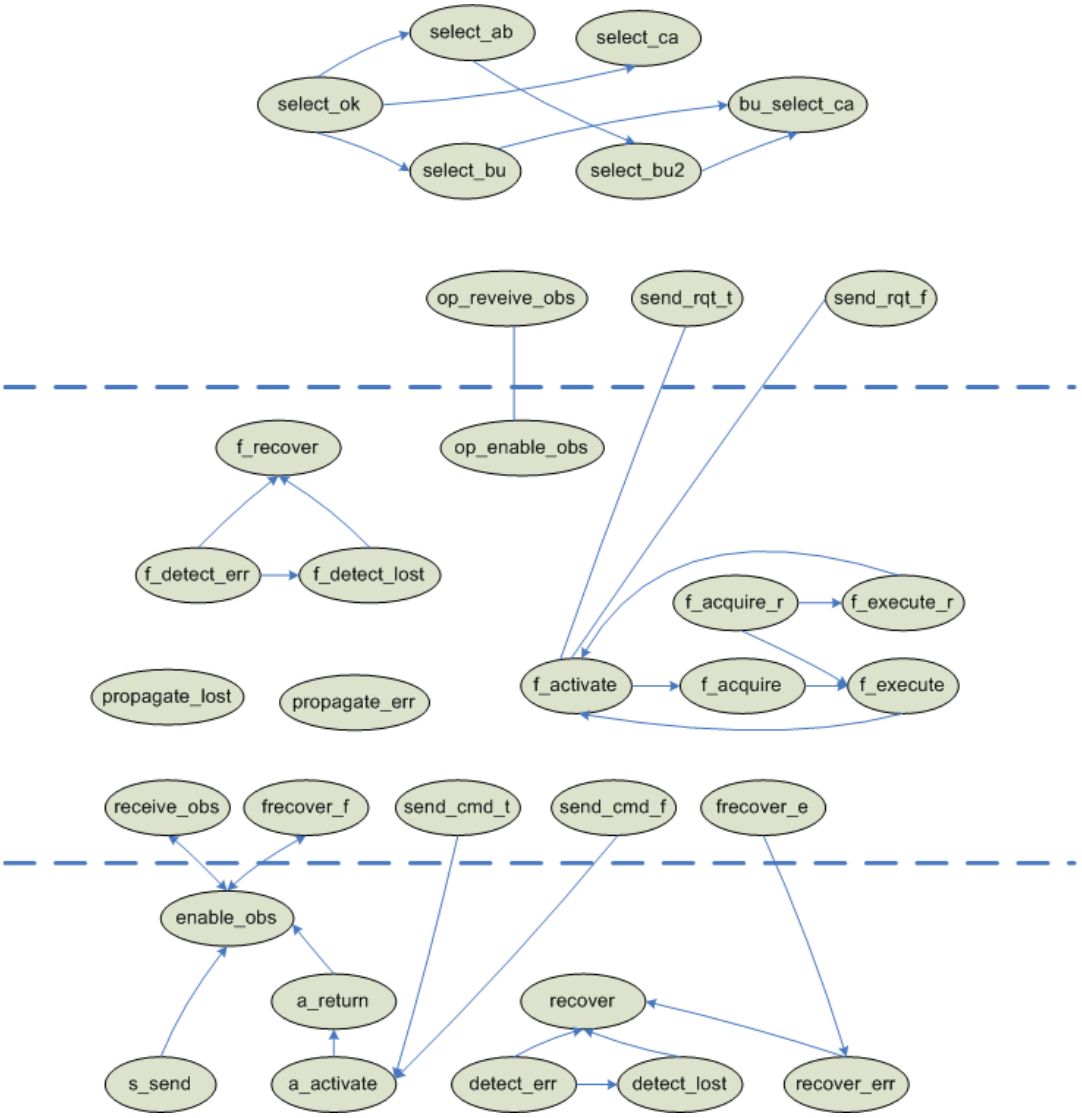

FIG. A.2 – Synchronisation des événements de la modélisation.

A.3 Ensemble des modèles développés

An Event-B Specification of UAV0_C0
Generated Date: 21 Oct 2011 @ 00 :43 :30 PM

CONTEXT UAV0_C0

SETS

EQUIPMENT

VALUE

RESOURCE

CONSTANTS

ACTUATOR

SENSOR

E_OBSERVATION

eqt_obs

vinit

AXIOMS

axm1 : $SENSOR \subseteq EQUIPMENT$

axm2 : $ACTUATOR \subseteq EQUIPMENT$

axm3 : $SENSOR \cap ACTUATOR = \varnothing$

axm4 : $EQUIPMENT = SENSOR \cup ACTUATOR$

axm5 : $E\_OBSERVATION \in \mathbb{P}_1(RESOURCE)$

axm6 : $eqt\_obs \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow E\_OBSERVATION$

axm7 : $vinit \in VALUE$

axm8 : $E\_OBSERVATION \neq \varnothing$

END

An Event-B Specification of UAV0_C1
Generated Date: 21 Oct 2011 @ 00 :44 :06 PM

CONTEXT UAV0_C1

EXTENDS UAV0_C0

SETS

E_FLAG

E_MODE

CONSTANTS

active

off

idle

spare

eqt_mode

AXIOMS

axm1 : $partition(E\_FLAG, \{active\}, \{off\}, \{idle\}, \{spare\})$

axm2 : $eqt\_mode \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow E\_MODE$

axm3 : $E\_MODE \rightarrow E\_FLAG \neq \varnothing$

END

An Event-B Specification of UAV0_C2
Generated Date: 21 Oct 2011 @ 00 :44 :19 PM

CONTEXT UAV0_C2

EXTENDS UAV0_C1

SETS

E_SURVEILLANCE

E_STATUS

DEFAULT

CONSTANTS

eqt_surv

ok

erroneous

lost

e_fault_class

SS_EQUIPMENT

eqt_cat

AXIOMS

axm1 : $eqt\_surv \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow E\_SURVEILLANCE$

axm2 : $partition(E\_STATUS, \{ok\}, \{erroneous\}, \{lost\})$

axm3 : $e\_fault\_class \in DEFAULT \nrightarrow EQUIPMENT$

axm4 : $SS\_EQUIPMENT \subseteq \mathbb{P}_1(EQUIPMENT)$

axm5 : $eqt\_cat \in EQUIPMENT \rightarrow SS\_EQUIPMENT$

END

An Event-B Specification of UAV_C2_1
Generated Date: 21 Oct 2011 @ 00 :44 :33 PM

CONTEXT UAV_C2_1
EXTENDS UAV0_C2
SETS
COMMAND
FUNCTION
CONSTANTS
act_cmd
L_FUNCT
SL_FUNCT
fct_cmd
SL_FCT_RES
AXIOMS
axm1 : $act\_cmd \in ACTUATOR \rightarrowtail\!\!\!\!\!\rightarrow COMMAND$
axm2 : $L\_FUNCT \in \mathbb{P}_1(FUNCTION)$
axm4 : $SL\_FUNCT \subseteq L\_FUNCT$
axm3 : $fct\_cmd \in SL\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow COMMAND$
axm5 : $COMMAND \rightarrow BOOL \neq \varnothing$
axm6 : $SL\_FCT\_RES \in L\_FUNCT \leftrightarrow RESOURCE$
axm7 : $SL\_FCT\_RES \rightarrow BOOL \neq \varnothing$
END

An Event-B Specification of UAV_C2_2
Generated Date: 21 Oct 2011 @ 00 :44 :43 PM

CONTEXT UAV_C2_2
EXTENDS UAV_C2_1
SETS
F_ESURVEILLANCE
CONSTANTS
FAILED
NOMINAL
f_eval
eqt_fsurv
AXIOMS
axm1 : $partition(VALUE, FAILED, NOMINAL)$
axm2 : $f\_eval \in VALUE \rightarrow BOOL$
axm3 : $\forall v \cdot (v \in NOMINAL \Rightarrow f\_eval(v) = TRUE)$

axm4 : $\forall v \cdot (v \in FAILED \Rightarrow f\_eval(v) = FALSE)$

axm5 : $eqt\_fsurv \in EQUIPMENT \rightarrowtail F\_ESURVEILLANCE$

END

An Event-B Specification of UAV_CT3
Generated Date: 23 Oct 2011 @ 10 :46 :57 PM

CONTEXT UAV_CT3

EXTENDS UAV0_C2

CONSTANTS

sen1

sen2

sen3

sen4

act1

act2

act3

AXIOMS

axm3 : $partition(SENSOR, \{sen1\}, \{sen2\}, \{sen3\}, \{sen4\})$

axm4 : $partition(ACTUATOR, \{act1\}, \{act2\}, \{act3\})$

axm5 : $SS\_EQUIPMENT = \{\{sen1, sen2\}, \{sen3, sen4\}, \{act1, act2\}, \{act3\}\}$

axm1 : $\forall sse \cdot (sse \in SS\_EQUIPMENT \Rightarrow ((sse \cap SENSOR \neq \varnothing \wedge sse \cap ACTUATOR = \varnothing) \vee (sse \cap SENSOR = \varnothing \wedge sse \cap ACTUATOR \neq \varnothing)))$

axm2 : $\forall sse1, sse2 \cdot (sse1 \in SS\_EQUIPMENT \wedge sse2 \in SS\_EQUIPMENT \Rightarrow (sse1 \neq sse2 \Leftrightarrow sse1 \cap sse2 = \varnothing))$

axm6 : $\forall x \cdot (x \in dom(eqt\_cat) \Rightarrow x \in eqt\_cat(x))$

axm7 : $\forall sse \cdot (sse \in SS\_EQUIPMENT \Rightarrow (card(sse) \leq 2 \wedge finite(sse)))$

END

An Event-B Specification of UAV_CF0
Generated Date: 21 Oct 2011 @ 00 :45 :31 PM

CONTEXT UAV_CF0

SETS

FUNCTION

F_FLAG

F_MODE

CONSTANTS

f_off
f_active
f_idle
f_executing
L_FUNCT
fct_mode

AXIOMS

axm1 : $partition(F\_FLAG, \{f\_active\}, \{f\_idle\}, \{f\_executing\}, \{f\_off\})$
axm2 : $L\_FUNCT \in \mathbb{P}_1(FUNCTION)$
axm3 : $F\_MODE \rightarrow F\_FLAG \neq \varnothing$
axm4 : $fct\_mode \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow F\_MODE$

END

An Event-B Specification of UAV_CF1
Generated Date: 21 Oct 2011 @ 00 :45 :39 PM

CONTEXT UAV_CF1
EXTENDS UAV_CF0
SETS
RESOURCE
CONSTANTS
SL_FCT_RES
AXIOMS
axm1 : $SL\_FCT\_RES \in L\_FUNCT \leftrightarrow RESOURCE$
axm2 : $SL\_FCT\_RES \rightarrow BOOL \neq \varnothing$
END

An Event-B Specification of UAV_CF1_1
Generated Date: 21 Oct 2011 @ 00 :45 :46 PM

CONTEXT UAV_CF1_1
EXTENDS UAV_CF1
SETS
F_SURVEILLANCE
DEFAULT
F_STATUS
VALUE
F_OBSERVATION

CONSTANTS

f_ok

f_erroneous

f_lost

fct_surv

f_fault_class

vinit

fct_obs

SS_FUNCTION

AXIOMS

axm1 : $partition(F\_STATUS, \{f\_ok\}, \{f\_erroneous\}, \{f\_lost\})$

axm2 : $fct\_surv \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow F\_SURVEILLANCE$

axm3 : $f\_fault\_class \in DEFAULT \nrightarrow L\_FUNCT$

axm4 : $vinit \in VALUE$

axm5 : $fct\_obs \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow F\_OBSERVATION$

axm6 : $SS\_FUNCTION \subseteq \mathbb{P}_1(L\_FUNCT)$

axm7 : $\forall ssf1, ssf2 \cdot (ssf1 \in SS\_FUNCTION \land ssf2 \in SS\_FUNCTION \land (ssf1 \neq ssf2 \Leftrightarrow ssf1 \cap ssf2 = \varnothing))$

END

An Event-B Specification of UAV_CF1_2
Generated Date: 21 Oct 2011 @ 00 :45 :53 PM

CONTEXT UAV_CF1_2

EXTENDS UAV_CF1_1

SETS

REQUEST

CONSTANTS

fct_req

AXIOMS

axm1 : $fct\_req \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow REQUEST$

axm2 : $REQUEST \rightarrow BOOL \neq \varnothing$

END

An Event-B Specification of UAV_CF1_3
Generated Date: 21 Oct 2011 @ 00 :46 :22 PM

CONTEXT UAV_CF1_3

EXTENDS UAV_CF1_2

SETS

MODE_OP

OPERATION

O_MODE

CONSTANTS

m_ok

m_backup

m_aborted

m_cancelled

ope_mode

L_FUNCT_OPE

l_fct_op

OP_bu

OP_ca

AXIOMS

axm1 : $partition(MODE\_OP, \{m\_ok\}, \{m\_backup\}, \{m\_aborted\}, \{m\_cancelled\})$

axm2 : $ope\_mode \in OPERATION \rightarrowtail\!\!\!\!\!\!\rightarrow O\_MODE$

axm3 : $L\_FUNCT\_OPE \subseteq \mathbb{P}_1(L\_FUNCT)$

axm4 : $l\_fct\_op \in OPERATION \rightarrowtail\!\!\!\!\!\!\rightarrow L\_FUNCT\_OPE$

axm5 : $\forall f \cdot (f \in L\_FUNCT \Rightarrow (\exists ssf \cdot (ssf \in L\_FUNCT\_OPE \wedge f \in ssf)))$

axm6 : $OP\_bu \in OPERATION$

axm7 : $OP\_ca \in OPERATION$

axm8 : $OP\_bu \neq OP\_ca$

END

An Event-B Specification of UAV_CF1_4
Generated Date: 21 Oct 2011 @ 00 :46 :29 PM

CONTEXT UAV_CF1_4

EXTENDS UAV_CF1_3

CONSTANTS

n

seq_op

fct_cat

AXIOMS

axm1 : $n \in \mathbb{N}$

axm3 : $n > 1$

axm2 : $seq\_op \in 1 \mathinner{..} n \rightarrow OPERATION \setminus \{OP\_bu, OP\_ca\}$

axm4 : $F\_MODE \rightarrow F\_FLAG \setminus \{f\_off\} \neq \varnothing$

axm5 : $fct\_cat \in L\_FUNCT \rightarrow SS\_FUNCTION$

END

An Event-B Specification of UAV_CF2
Generated Date: 23 Oct 2011 @ 10 :46 :28 PM

CONTEXT UAV_CF2

EXTENDS UAV_CF1_1

SETS

REQUEST

CONSTANTS

fct_req

fct_cat

f1

f2

f3

f4

AXIOMS

axm5 : $partition(L\_FUNCT, \{f1\}, \{f2\}, \{f3\}, \{f4\})$

axm1 : $fct\_req \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow REQUEST$

axm2 : $SS\_FUNCTION = \{\{f1, f2\}, \{f3, f4\}\}$

axm3 : $fct\_cat \in L\_FUNCT \rightarrow SS\_FUNCTION$

axm4 : $SS\_FUNCTION \rightarrow \mathbb{N} \neq \varnothing$

axm6 : $\forall f \cdot (f \in dom(fct\_cat) \Rightarrow f \in fct\_cat(f))$

axm7 : $\forall ssf \cdot (ssf \in SS\_FUNCTION \Rightarrow (card(ssf) \leq 2 \wedge finite(ssf)))$

END

An Event-B Specification of UAV_CF2n
Generated Date: 1 Dec 2011 @ 00 :30 :32 AM

CONTEXT UAV_CF2

EXTENDS UAV_CF1_1

SETS

REQUEST

CONSTANTS

fct_req

fct_cat

f1

f2

f3

f4

L_FUNCT_min

AXIOMS

axm10 : $finite(SS\_FUNCTION)$

axm9 : $\{f1, f2, f3, f4\} \subseteq L\_FUNCT$

axm8 : $L\_FUNCT\_min \subseteq L\_FUNCT$

axm5 : $partition(L\_FUNCT\_min, \{f1\}, \{f2\}, \{f3\}, \{f4\})$

axm1 : $fct\_req \in L\_FUNCT \rightarrowtail\!\!\!\!\!\rightarrow REQUEST$

axm2 : $\{\{f1, f2\}, \{f3, f4\}\} \subseteq SS\_FUNCTION$

axm3 : $fct\_cat \in L\_FUNCT \rightarrow SS\_FUNCTION$

axm4 : $SS\_FUNCTION \rightarrow \mathbb{N} \neq \varnothing$

axm6 : $\forall f \cdot (f \in dom(fct\_cat) \Rightarrow f \in fct\_cat(f))$

axm7 : $\forall ssf \cdot (ssf \in SS\_FUNCTION \Rightarrow (card(ssf) \leq 2 \wedge finite(ssf)))$

END

An Event-B Specification of EQUIP_0
Generated Date: 21 Oct 2011 @ 00 :47 :36 PM

MACHINE EQUIP_0
SEES UAV0_C0
VARIABLES
value
INVARIANTS
inv1 : $value \in E\_OBSERVATION \rightarrow VALUE$
EVENTS
Initialisation
begin
act1 : $value := E\_OBSERVATION \times \{vinit\}$
end
Event $s\_send \mathrel{\widehat{=}}$
any
sen
v
where
grd1 : $sen \in SENSOR$
grd2 : $v \in VALUE$
then
act1 : $value(eqt\_obs(sen)) := v$
end
Event $a\_return \mathrel{\widehat{=}}$
any
act
v
where
grd1 : $act \in ACTUATOR$
grd2 : $v \in VALUE$
then
act1 : $value(eqt\_obs(act)) := v$
end
END

An Event-B Specification of EQUIP_1
Generated Date: 21 Oct 2011 @ 00 :47 :44 PM

MACHINE EQUIP_1
REFINES EQUIP_0
SEES UAV0_C1
VARIABLES
`value`
`buffer_obs`
INVARIANTS
`inv2` : $value \in E\_OBSERVATION \rightarrow VALUE$
`inv3` : $buffer\_obs \in E\_OBSERVATION \rightarrow BOOL$
EVENTS
Initialisation
begin
`act2` : $value := E\_OBSERVATION \times \{vinit\}$
`act3` : $buffer\_obs := E\_OBSERVATION \times \{FALSE\}$
end
Event $s\_send \mathrel{\widehat{=}}$
refines s_send
any
sen
v
where
`grd1` : $sen \in SENSOR$
`grd2` : $v \in VALUE$
`grd3` : $buffer\_obs(eqt\_obs(sen)) = FALSE$
then
`act1` : $value(eqt\_obs(sen)) := v$
`act2` : $buffer\_obs(eqt\_obs(sen)) := TRUE$
end
Event $a\_return \mathrel{\widehat{=}}$
refines a_return
any
act
v

where

grd1 : $act \in ACTUATOR$

grd2 : $v \in VALUE$

grd3 : $buffer\_obs(eqt\_obs(act)) = FALSE$

then

act1 : $value(eqt\_obs(act)) := v$

act3 : $buffer\_obs(eqt\_obs(act)) := TRUE$

end

Event $enable\_obs \widehat{=}$

any

obs

where

grd1 : $obs \in E\_OBSERVATION$

grd2 : $buffer\_obs(obs) = TRUE$

then

act1 : $buffer\_obs(obs) := FALSE$

end

END

An Event-B Specification of EQUIP_1_1
Generated Date: 21 Oct 2011 @ 00 :47 :51 PM

MACHINE EQUIP_1_1
REFINES EQUIP_1
SEES UAV0_C2
VARIABLES

`flag`
`e_status`
`buffer_obs`
`value`

INVARIANTS

inv1 : $e\_status \in E\_SURVEILLANCE \rightarrow E\_STATUS$
inv2 : $flag \in E\_MODE \rightarrow E\_FLAG$

EVENTS
Initialisation
begin

act1 : $flag :\in E\_MODE \rightarrow E\_FLAG$
act2 : $e\_status := E\_SURVEILLANCE \times \{ok\}$
act3 : $buffer\_obs := E\_OBSERVATION \times \{FALSE\}$
act4 : $value := E\_OBSERVATION \times \{vinit\}$

end
Event $s\_send \mathrel{\widehat{=}}$
refines s_send
any

sen
v

where

grd1 : $sen \in SENSOR$
grd2 : $v \in VALUE$
grd3 : $flag(eqt\_mode(sen)) = active$
grd4 : $e\_status(eqt\_surv(sen)) \neq lost$
grd5 : $buffer\_obs(eqt\_obs(sen)) = FALSE$

then

act1 : $value(eqt\_obs(sen)) := v$
act2 : $buffer\_obs(eqt\_obs(sen)) := TRUE$

end

Event $a\_return \mathrel{\widehat{=}}$

refines a_return

any

act

v

where

grd1 : $act \in ACTUATOR$

grd2 : $v \in VALUE$

grd3 : $flag(eqt\_mode(act)) = active$

grd4 : $e\_status(eqt\_surv(act)) \neq lost$

grd5 : $buffer\_obs(eqt\_obs(act)) = FALSE$

then

act1 : $value(eqt\_obs(act)) := v$

act2 : $flag(eqt\_mode(act)) := idle$

act3 : $buffer\_obs(eqt\_obs(act)) := TRUE$

end

Event $a\_activate \mathrel{\widehat{=}}$

any

act

where

grd1 : $act \in ACTUATOR$

grd2 : $flag(eqt\_mode(act)) = idle$

grd3 : $e\_status(eqt\_surv(act)) \neq lost$

then

act1 : $flag(eqt\_mode(act)) := active$

end

Event $enable\_obs \mathrel{\widehat{=}}$

refines $enable\_obs$

any

obs

where

grd1 : $obs \in E\_OBSERVATION$

grd2 : $buffer\_obs(obs) = TRUE$

then

act1 : $buffer\_obs(obs) := FALSE$

end

Event $detect\_err \mathrel{\widehat{=}}$

any

$fault$

eqt

where

grd1 : $fault \in DEFAULT$

grd2 : $eqt \in EQUIPMENT$

then

act1 : $e\_status(eqt\_surv(eqt)) := erroneous$

end

Event $detect\_lost \mathrel{\widehat{=}}$

any

$fault$

eqt

where

grd1 : $fault \in DEFAULT$

grd2 : $eqt \in EQUIPMENT$

then

act1 : $e\_status(eqt\_surv(eqt)) := lost$

end

Event $recover \mathrel{\widehat{=}}$

any

eq

eq_red

where

grd1 : $eq \in EQUIPMENT$

grd2 : $eq\_red \in EQUIPMENT$

grd3 : $flag(eqt\_mode(eq)) = active$

grd4 : $flag(eqt\_mode(eq\_red)) = spare$

then

act1 : $flag := flag \lhd \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto off\}$

end

Event $recover\_err \mathrel{\widehat{=}}$

any

eqt

where

grd1 : $eqt \in EQUIPMENT$

grd2 : $e\_status(eqt\_surv(eqt)) = ok$

then

act1 : $e\_status(eqt\_surv(eqt)) := erroneous$

end

END

An Event-B Specification of EQUIP_2
Generated Date: 21 Oct 2011 @ 00 :47 :58 PM

MACHINE EQUIP_2
REFINES EQUIP_1_1
SEES UAV0_C2
VARIABLES
`e_status`
`flag`
INVARIANTS
`inv1` : $e\_status \in E\_SURVEILLANCE \rightarrow E\_STATUS$
EVENTS
Initialisation
begin
`act1` : $e\_status := E\_SURVEILLANCE \times \{ok\}$
`act2` : $flag :\in E\_MODE \rightarrow E\_FLAG$
end
Event $detect\_err \mathrel{\widehat{=}}$
refines $detect\_err$
any
$fault$
eqt
where
`grd1` : $fault \mapsto eqt \in e\_fault\_class$
`grd2` : $e\_status(eqt\_surv(eqt)) = ok$
`grd3` : $flag(eqt\_mode(eqt)) = active$
then
`act1` : $e\_status(eqt\_surv(eqt)) := erroneous$
end
Event $detect\_lost \mathrel{\widehat{=}}$
refines $detect\_lost$
any
$fault$
eqt
where
`grd1` : $fault \mapsto eqt \in e\_fault\_class$

grd2 : $e\_status(eqt\_surv(eqt)) \neq lost$

grd3 : $flag(eqt\_mode(eqt)) = active$

then

act1 : $e\_status(eqt\_surv(eqt)) := lost$

end

Event *recover* $\widehat{=}$

refines *recover*

any

eq

eq_red

where

grd1 : $eq \in EQUIPMENT$

grd2 : $eq\_red \in EQUIPMENT$

grd3 : $eq\_red \in eqt\_cat(eq)$

grd4 : $e\_status(eqt\_surv(eq)) = lost$

grd5 : $flag(eqt\_mode(eq)) = active$

grd6 : $flag(eqt\_mode(eq\_red)) = spare$

then

act1 : $flag := flag \lhd \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto off\}$

end

END

An Event-B Specification of EQUIP_T3
Generated Date: 21 Oct 2011 @ 00 :48 :24 PM

MACHINE EQUIP_T3

REFINES EQUIP_2

SEES UAV_CT3

VARIABLES

`flag`

`e_status`

`nb`

INVARIANTS

inv1 : $nb \in SS\_EQUIPMENT \rightarrow 0 \mathinner{..} 2$

inv2 : $\forall sse \cdot ((sse \in SS\_EQUIPMENT \wedge nb(sse) > 1) \Rightarrow (\exists a, b \cdot (a \in sse \wedge b \in sse \wedge a \neq b \wedge flag(eqt\_mode(b)) = spare \wedge (flag(eqt\_mode(a)) = active \vee flag(eqt\_mode(a)) = idle))))$

inv3 : $\forall sse, b \cdot ((sse \in SS\_EQUIPMENT \wedge b \in sse \wedge nb(sse) = 1) \Rightarrow (\exists a \cdot (a \in sse \wedge flag(eqt\_mode(a)) \neq off \wedge (a \neq b \Rightarrow flag(eqt\_mode(b)) = off))))$

inv4 : $\forall sse, a \cdot ((sse \in SS\_EQUIPMENT \wedge a \in sse \wedge nb(sse) = 0) \Rightarrow flag(eqt\_mode(a)) = off)$

EVENTS

Initialisation

begin

act1 : $e\_status := E\_SURVEILLANCE \times \{ok\}$

act2 : $flag := \{eqt\_mode(sen1) \mapsto active, eqt\_mode(sen2) \mapsto spare, eqt\_mode(sen3) \mapsto spare, eqt\_mode(sen4) \mapsto active, eqt\_mode(act1) \mapsto idle, eqt\_mode(act2) \mapsto spare, eqt\_mode(act3) \mapsto idle\}$

act3 : $nb := \{\{sen1, sen2\} \mapsto 2, \{sen3, sen4\} \mapsto 2, \{act1, act2\} \mapsto 2, \{act3\} \mapsto 1\}$

end

Event *recover* $\widehat{=}$

refines *recover*

any

eq

eq_red

where

grd1 : $eq \in EQUIPMENT$

grd2 : $eq\_red \in EQUIPMENT$

grd3 : $eq\_red \in eqt\_cat(eq)$

grd4 : $e\_status(eqt\_surv(eq)) = lost$

grd5 : $flag(eqt\_mode(eq)) = active$

grd6 : $flag(eqt\_mode(eq\_red)) = spare$

grd7 : $nb(eqt\_cat(eq)) > 1$

then

act1 : $flag := flag \Leftarrow \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto off\}$

act2 : $nb(eqt\_cat(eq)) := nb(eqt\_cat(eq)) - 1$

end

END

An Event-B Specification of EQUIP_4
Generated Date: 21 Oct 2011 @ 00 :48 :16 PM

MACHINE EQUIP_4
REFINES E_FCT_2
SEES UAV_C2_2
VARIABLES
`flag`
`f_res`
`e_cmd`
`e_status`
`cmd_estatus`
`value`
INVARIANTS
`inv1` : $\forall eq \cdot ((eq \in EQUIPMENT \land flag(eqt\_mode(eq)) = active \land e\_status(eqt\_surv(eq)) = erroneous) \Rightarrow (f\_eval(value(eqt\_obs(eq))) = FALSE))$
EVENTS
Initialisation
begin
`act1` : $flag :\in E\_MODE \rightarrow E\_FLAG$
`act2` : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$
`act3` : $e\_cmd :\in COMMAND \rightarrow BOOL$
`act4` : $e\_status := E\_SURVEILLANCE \times \{ok\}$
`act5` : $cmd\_estatus := F\_ESURVEILLANCE \times \{ok\}$
`act6` : $value := E\_OBSERVATION \times \{vinit\}$
end
Event $recover\_err \mathrel{\widehat{=}}$
refines $recover\_err$
any
eqt
where
`grd1` : $eqt \in EQUIPMENT$
`grd2` : $e\_status(eqt\_surv(eqt)) = ok$
`grd3` : $cmd\_estatus(eqt\_fsurv(eqt)) = erroneous$
`grd4` : $f\_eval(value(eqt\_obs(eqt))) = FALSE$
then
`act1` : $e\_status(eqt\_surv(eqt)) := erroneous$
end
END

An Event-B Specification of E_FCT_1
Generated Date: 21 Oct 2011 @ 00 :48 :50 PM

MACHINE E_FCT_1

REFINES EQUIP_1_1

SEES UAV_C2_1

VARIABLES

`flag`

`e_cmd`

`e_status`

`value`

`buffer_obs`

`f_res`

`fbuffer_obs`

INVARIANTS

`inv1` : $e\_cmd \in COMMAND \rightarrow BOOL$

`inv2` : $f\_res \in SL\_FCT\_RES \rightarrow BOOL$

`inv3` : $fbuffer\_obs \in E\_OBSERVATION \rightarrow BOOL$

`inv4` : $\forall obs \cdot (obs \in E\_OBSERVATION \Rightarrow (buffer\_obs(obs) = TRUE \lor fbuffer\_obs(obs) = FALSE))$

EVENTS

Initialisation

begin

`act1` : $flag :\in E\_MODE \rightarrow E\_FLAG$

`act2` : $e\_cmd :\in COMMAND \rightarrow BOOL$

`act3` : $e\_status := E\_SURVEILLANCE \times \{ok\}$

`act4` : $value := E\_OBSERVATION \times \{vinit\}$

`act5` : $buffer\_obs := E\_OBSERVATION \times \{FALSE\}$

`act6` : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$

`act7` : $fbuffer\_obs := E\_OBSERVATION \times \{FALSE\}$

end

Event $s\_send \mathrel{\widehat{=}}$

refines s_send

any

sen

v

where

grd1 : $sen \in SENSOR$

grd2 : $v \in VALUE$

grd3 : $flag(eqt\_mode(sen)) = active$

grd4 : $e\_status(eqt\_surv(sen)) \neq lost$

grd5 : $buffer\_obs(eqt\_obs(sen)) = FALSE$

grd6 : $fbuffer\_obs(eqt\_obs(sen)) = FALSE$

then

act1 : $value(eqt\_obs(sen)) := v$

act2 : $buffer\_obs(eqt\_obs(sen)) := TRUE$

end

Event $a\_activate \mathrel{\widehat{=}}$

refines $a\_activate$

any

act

where

grd1 : $act \in ACTUATOR$

grd2 : $flag(eqt\_mode(act)) = idle$

grd3 : $e\_status(eqt\_surv(act)) \neq lost$

grd4 : $e\_cmd(act\_cmd(act)) = TRUE$

then

act1 : $flag(eqt\_mode(act)) := active$

end

Event $a\_return \mathrel{\widehat{=}}$

refines a_return

any

act

v

where

grd1 : $act \in ACTUATOR$

grd2 : $v \in VALUE$

grd3 : $flag(eqt\_mode(act)) = active$

grd4 : $e\_status(eqt\_surv(act)) \neq lost$

grd5 : $e\_cmd(act\_cmd(act)) = FALSE$

grd6 : $buffer\_obs(eqt\_obs(act)) = FALSE$

grd7 : $fbuffer\_obs(eqt\_obs(act)) = FALSE$

then

act1 : $flag(eqt\_mode(act)) := idle$

act2 : $value(eqt\_obs(act)) := v$

act3 : $buffer\_obs(eqt\_obs(act)) := TRUE$

end

Event $send\_cmd\_t \mathrel{\widehat{=}}$

any

fct

where

grd1 : $fct \in SL\_FUNCT$

then

act1 : $e\_cmd(fct\_cmd(fct)) := TRUE$

end

Event $send\_cmd\_f \mathrel{\widehat{=}}$

any

fct

where

grd1 : $fct \in SL\_FUNCT$

then

act1 : $e\_cmd(fct\_cmd(fct)) := FALSE$

end

Event $receive\_obs \mathrel{\widehat{=}}$

any

obs

SL_fct_o

where

grd1 : $obs \in E\_OBSERVATION$

grd2 : $SL\_fct\_o \subseteq L\_FUNCT$

grd3 : $SL\_fct\_o \times \{obs\} \subseteq SL\_FCT\_RES$

grd4 : $buffer\_obs(obs) = TRUE$

grd5 : $fbuffer\_obs(obs) = FALSE$

then

act1 : $f\_res :\in \{f\_res \Leftarrow ((SL\_fct\_o \times \{obs\}) \times \{TRUE\}), f\_res \Leftarrow ((SL\_fct\_o \times \{obs\}) \times \{FALSE\})\}$

act2 : $fbuffer\_obs(obs) := TRUE$

end

Event $enable\_obs \widehat{=}$

refines $enable\_obs$

any

obs

where

grd1 : $obs \in E\_OBSERVATION$

grd2 : $buffer\_obs(obs) = TRUE$

grd3 : $fbuffer\_obs(obs) = TRUE$

then

act1 : $buffer\_obs(obs) := FALSE$

act2 : $fbuffer\_obs(obs) := FALSE$

end

Event $recover\_err \widehat{=}$

refines $recover\_err$

any

eqt

where

grd1 : $eqt \in EQUIPMENT$

grd2 : $e\_status(eqt\_surv(eqt)) = ok$

then

act1 : $e\_status(eqt\_surv(eqt)) := erroneous$

end

END

An Event-B Specification of E_FCT_2
Generated Date: 21 Oct 2011 @ 00 :48 :58 PM

MACHINE E_FCT_2
REFINES E_FCT_1
SEES UAV_C2_2
VARIABLES

`f_res`
`buffer_obs`
`fbuffer_obs`
`e_cmd`
`flag`
`value`
`e_status`
`cmd_estatus`

INVARIANTS

`inv1` : $cmd\_estatus \in F\_ESURVEILLANCE \rightarrow E\_STATUS$

EVENTS
Initialisation
begin

`act1` : $buffer\_obs := E\_OBSERVATION \times \{FALSE\}$
`act2` : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$
`act3` : $fbuffer\_obs := E\_OBSERVATION \times \{FALSE\}$
`act4` : $e\_cmd :\in COMMAND \rightarrow BOOL$
`act5` : $flag :\in E\_MODE \rightarrow E\_FLAG$
`act6` : $value := E\_OBSERVATION \times \{vinit\}$
`act7` : $e\_status := E\_SURVEILLANCE \times \{ok\}$
`act8` : $cmd\_estatus := F\_ESURVEILLANCE \times \{ok\}$

end

Event $receive\_obs \mathrel{\widehat{=}}$
refines $receive\_obs$
any

obs
SL_fct_o

where

`grd1` : $obs \in E\_OBSERVATION$

grd2 : $SL\_fct\_o \subseteq L\_FUNCT$
grd3 : $SL\_fct\_o \times \{obs\} \subseteq SL\_FCT\_RES$
grd4 : $buffer\_obs(obs) = TRUE$
grd5 : $fbuffer\_obs(obs) = FALSE$
grd6 : $f\_eval(value(obs)) = TRUE$

then

act1 : $f\_res := f\_res \Leftarrow ((SL\_fct\_o \times \{obs\}) \times \{TRUE\})$
act2 : $fbuffer\_obs(obs) := TRUE$

end

Event *frecover_f* $\widehat{=}$

refines *receive_obs*

any

obs
SL_fct_o

where

grd1 : $obs \in E\_OBSERVATION$
grd2 : $SL\_fct\_o \subseteq L\_FUNCT$
grd3 : $SL\_fct\_o \times \{obs\} \subseteq SL\_FCT\_RES$
grd4 : $buffer\_obs(obs) = TRUE$
grd5 : $fbuffer\_obs(obs) = FALSE$
grd6 : $f\_eval(value(obs)) = FALSE$

then

act1 : $f\_res := f\_res \Leftarrow ((SL\_fct\_o \times \{obs\}) \times \{FALSE\})$
act2 : $fbuffer\_obs(obs) := TRUE$

end

Event *frecover_e* $\widehat{=}$

any

eqt

where

grd1 : $eqt \in EQUIPMENT$
grd2 : $flag(eqt\_mode(eqt)) = active$
grd3 : $e\_status(eqt\_surv(eqt)) = ok$
grd4 : $f\_eval(value(eqt\_obs(eqt))) = FALSE$
grd5 : $cmd\_estatus(eqt\_fsurv(eqt)) = ok$

then

act1 : $cmd\_estatus(eqt\_fsurv(eqt)) := erroneous$

end

Event $recover\_err \mathrel{\widehat{=}}$

refines $recover\_err$

any

eqt

where

grd1 : $eqt \in EQUIPMENT$

grd2 : $e\_status(eqt\_surv(eqt)) = ok$

grd3 : $cmd\_estatus(eqt\_fsurv(eqt)) = erroneous$

then

act1 : $e\_status(eqt\_surv(eqt)) := erroneous$

end

END

An Event-B Specification of FUNCT_0
Generated Date: 21 Oct 2011 @ 00 :49 :05 PM

MACHINE FUNCT_0
SEES UAV_CF0
VARIABLES
f_flag
INVARIANTS
inv1 : $f\_flag \in F\_MODE \rightarrow F\_FLAG$
EVENTS
Initialisation
begin
act1 : $f\_flag :\in F\_MODE \rightarrow F\_FLAG$
end
Event $f\_activate \mathrel{\widehat{=}}$
any
fct
where
grd1 : $fct \in L\_FUNCT$
grd2 : $f\_flag(fct\_mode(fct)) = f\_idle$
then
act1 : $f\_flag(fct\_mode(fct)) := f\_active$
end
Event $f\_acquire \mathrel{\widehat{=}}$
any
fct
where
grd1 : $fct \in L\_FUNCT$
grd2 : $f\_flag(fct\_mode(fct)) = f\_active$
then
act1 : $f\_flag(fct\_mode(fct)) := f\_executing$
end
Event $f\_execute \mathrel{\widehat{=}}$
any
fct
where

grd1 : $fct \in L\_FUNCT$

grd2 : $f\_flag(fct\_mode(fct)) = f\_executing$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_idle$

end

Event $f\_recover \mathrel{\widehat{=}}$

any

fct

where

grd1 : $fct \in L\_FUNCT$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_off$

end

END

An Event-B Specification of FUNCT_1
Generated Date: 21 Oct 2011 @ 00 :49 :13 PM

MACHINE FUNCT_1
REFINES FUNCT_0
SEES UAV_CF1
VARIABLES
`f_flag`
`f_res`
INVARIANTS
inv1 : $f\_res \in SL\_FCT\_RES \rightarrow BOOL$
EVENTS
Initialisation
begin
act1 : $f\_flag :\in F\_MODE \rightarrow F\_FLAG$
act2 : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$
end
Event $f\_activate \mathrel{\widehat{=}}$
refines $f\_activate$
any
fct
where
grd1 : $fct \in L\_FUNCT$
grd2 : $f\_flag(fct\_mode(fct)) = f\_idle$
then
act1 : $f\_flag(fct\_mode(fct)) := f\_active$
end
Event $f\_acquire \mathrel{\widehat{=}}$
refines $f\_acquire$
any
fct
where
grd1 : $fct \in L\_FUNCT$
grd2 : $f\_flag(fct\_mode(fct)) = f\_active$
then
act1 : $f\_flag(fct\_mode(fct)) := f\_executing$

end

Event $f\_execute \widehat{=}$

refines $f\_execute$

any

fct

where

grd1 : $fct \in L\_FUNCT$

grd2 : $f\_flag(fct\_mode(fct)) = f\_executing$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_idle$

end

Event $f\_acquire\_r \widehat{=}$

refines $f\_acquire$

any

fct

res

where

grd1 : $fct \in L\_FUNCT$

grd2 : $fct \mapsto res \in SL\_FCT\_RES$

grd3 : $f\_flag(fct\_mode(fct)) = f\_active$

grd4 : $f\_res(fct \mapsto res) = TRUE$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_executing$

act2 : $f\_res(fct \mapsto res) := FALSE$

end

Event $f\_execute\_r \widehat{=}$

refines $f\_execute$

any

fct

res

where

grd1 : $fct \in L\_FUNCT$

grd2 : $fct \mapsto res \in SL\_FCT\_RES$

grd3 : $f\_flag(fct\_mode(fct)) = f\_executing$

grd4 : $f\_res(fct \mapsto res) = FALSE$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_idle$

act2 : $f\_res(fct \mapsto res) := TRUE$

end

Event $f\_recover \mathrel{\widehat{=}}$

refines $f\_recover$

any

fct

where

grd1 : $fct \in L\_FUNCT$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_off$

end

END

An Event-B Specification of FUNCT_1_1
Generated Date: 21 Oct 2011 @ 00 :49 :19 PM

MACHINE FUNCT_1_1
REFINES FUNCT_1
SEES UAV_CF1_1
VARIABLES

f_flag
f_res
f_status
fvalue
fbuffer_h

INVARIANTS

inv1 : $f\_status \in F\_SURVEILLANCE \rightarrow F\_STATUS$
inv2 : $fvalue \in F\_OBSERVATION \rightarrow VALUE$
inv3 : $fbuffer\_h \in F\_OBSERVATION \rightarrow BOOL$

EVENTS
Initialisation

begin

act1 : $f\_flag :\in F\_MODE \rightarrow F\_FLAG$
act2 : $f\_status := F\_SURVEILLANCE \times \{f\_ok\}$
act3 : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$
act4 : $fvalue := F\_OBSERVATION \times \{vinit\}$
act5 : $fbuffer\_h := F\_OBSERVATION \times \{FALSE\}$

end

Event $f\_activate \mathrel{\widehat{=}}$
refines $f\_activate$

any

fct

where

grd1 : $fct \in L\_FUNCT$
grd2 : $f\_flag(fct\_mode(fct)) = f\_idle$
grd3 : $f\_status(fct\_surv(fct)) \neq f\_lost$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_active$

end

Event $f\_acquire \mathrel{\widehat{=}}$
refines $f\_acquire$
any
fct
where
grd1 : $fct \in L\_FUNCT$
grd2 : $f\_flag(fct\_mode(fct)) = f\_active$
then
act1 : $f\_flag(fct\_mode(fct)) := f\_executing$
end
Event $f\_execute \mathrel{\widehat{=}}$
refines $f\_execute$
any
fct
v
where
grd1 : $fct \in L\_FUNCT$
grd3 : $v \in VALUE$
grd2 : $f\_flag(fct\_mode(fct)) = f\_executing$
grd4 : $fbuffer\_h(fct\_obs(fct)) = FALSE$
then
act1 : $f\_flag(fct\_mode(fct)) := f\_idle$
act2 : $fvalue(fct\_obs(fct)) := v$
act3 : $fbuffer\_h(fct\_obs(fct)) := TRUE$
end
Event $f\_execute\_r \mathrel{\widehat{=}}$
refines $f\_execute\_r$
any
fct
res
v
where
grd1 : $fct \in L\_FUNCT$
grd2 : $fct \mapsto res \in SL\_FCT\_RES$
grd3 : $v \in VALUE$

grd4 : $f\_flag(fct\_mode(fct)) = f\_executing$
grd5 : $f\_res(fct \mapsto res) = FALSE$
grd6 : $fbuffer\_h(fct\_obs(fct)) = FALSE$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_idle$
act2 : $f\_res(fct \mapsto res) := TRUE$
act3 : $fvalue(fct\_obs(fct)) := v$
act4 : $fbuffer\_h(fct\_obs(fct)) := TRUE$

end

Event $f\_detect\_err \mathrel{\widehat{=}}$

any

$fault$
fct

where

grd4 : $fct \in L\_FUNCT$
grd1 : $fault \mapsto fct \in f\_fault\_class$
grd2 : $f\_flag(fct\_mode(fct)) = f\_executing$
grd3 : $f\_status(fct\_surv(fct)) = f\_ok$

then

act1 : $f\_status(fct\_surv(fct)) := f\_erroneous$

end

Event $f\_detect\_lost \mathrel{\widehat{=}}$

any

$fault$
fct

where

grd4 : $fct \in L\_FUNCT$
grd1 : $fault \mapsto fct \in f\_fault\_class$
grd2 : $f\_flag(fct\_mode(fct)) = f\_executing \lor f\_flag(fct\_mode(fct)) = f\_active$
grd3 : $f\_status(fct\_surv(fct)) \neq f\_lost$

then

act1 : $f\_status(fct\_surv(fct)) := f\_lost$

end

Event $op\_enable\_obs \mathrel{\widehat{=}}$

any

obs

where

grd1 : $obs \in F\_OBSERVATION$

grd2 : $fbuffer\_h(obs) = TRUE$

then

act1 : $fbuffer\_h(obs) := FALSE$

end

Event $f\_recover \mathrel{\widehat{=}}$

refines $f\_recover$

any

fct

where

grd1 : $fct \in L\_FUNCT$

grd2 : $f\_status(fct\_surv(fct)) = f\_erroneous$

grd3 : $f\_flag(fct\_mode(fct)) \neq f\_off$

then

act1 : $f\_status(fct\_surv(fct)) := f\_lost$

act2 : $f\_flag(fct\_mode(fct)) := f\_off$

end

END

An Event-B Specification of FUNCT_T2
Generated Date: 21 Oct 2011 @ 00 :49 :33 PM

MACHINE FUNCT_T2

REFINES FUNCT_1_1

SEES UAV_CF2

VARIABLES

`f_flag`

`f_res`

`f_nb`

`f_status`

INVARIANTS

`inv1` : $f\_nb \in SS\_FUNCTION \rightarrow 0 \mathinner{..} 2$

`inv2` : $\forall ssf \cdot ((ssf \in SS\_FUNCTION \land f\_nb(ssf) > 1) \Rightarrow (\exists a, b \cdot (a \in ssf \land b \in ssf \land a \neq b \land f\_flag(fct\_mode(a)) \neq f\_off \land f\_flag(fct\_mode(b)) \neq f\_off)))$

`inv3` : $\forall ssf, b \cdot ((ssf \in SS\_FUNCTION \land b \in ssf \land f\_nb(ssf) = 1) \Rightarrow (\exists a \cdot (a \in ssf \land f\_flag(fct\_mode(a)) \neq f\_off \land (a \neq b \Rightarrow f\_flag(fct\_mode(b)) = f\_off))))$

`inv4` : $\forall ssf, a \cdot ((ssf \in SS\_FUNCTION \land a \in ssf \land f\_nb(ssf) = 0) \Rightarrow f\_flag(fct\_mode(a)) = f\_off)$

EVENTS

Initialisation

begin

`act1` : $f\_flag := F\_MODE \times \{f\_idle\}$

`act2` : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$

`act3` : $f\_nb := SS\_FUNCTION \times \{2\}$

`act4` : $f\_status := F\_SURVEILLANCE \times \{f\_ok\}$

end

Event $f\_recover \mathrel{\widehat{=}}$

refines $f\_recover$

any

fct

fct_eq

where

`grd1` : $fct \in L\_FUNCT$

`grd2` : $fct\_eq \in L\_FUNCT$

grd3 : $fct\_eq \in fct\_cat(fct)$

grd4 : $fct\_eq \neq fct$

grd5 : $f\_status(fct\_surv(fct)) \neq f\_ok$

grd6 : $f\_status(fct\_surv(fct\_eq)) = f\_ok$

grd7 : $f\_nb(fct\_cat(fct)) > 1$

grd8 : $f\_flag(fct\_mode(fct)) \neq f\_off$

grd9 : $f\_flag(fct\_mode(fct\_eq)) \neq f\_off$

then

act1 : $f\_nb(fct\_cat(fct)) := f\_nb(fct\_cat(fct)) - 1$

act2 : $f\_status(fct\_surv(fct)) := f\_lost$

act3 : $f\_flag(fct\_mode(fct)) := f\_off$

end

END

An Event-B Specification of F_OPE_1
Generated Date: 21 Oct 2011 @ 00 :50 :03 PM

MACHINE F_OPE_1
REFINES FUNCT_1_1
SEES UAV_CF1_2
VARIABLES

`f_flag`
`f_res`
`f_rqt`
`f_status`
`fbuffer_h`
`fvalue`
`fopbuffer_h`

INVARIANTS

`inv1` : $f\_rqt \in REQUEST \rightarrow BOOL$
`inv2` : $fopbuffer\_h \in F\_OBSERVATION \rightarrow BOOL$

EVENTS
Initialisation
begin

`act1` : $f\_flag :\in F\_MODE \rightarrow F\_FLAG$
`act2` : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$
`act3` : $f\_rqt :\in REQUEST \rightarrow BOOL$
`act4` : $f\_status := F\_SURVEILLANCE \times \{f\_ok\}$
`act5` : $fbuffer\_h := F\_OBSERVATION \times \{FALSE\}$
`act6` : $fvalue := F\_OBSERVATION \times \{vinit\}$
`act7` : $fopbuffer\_h := F\_OBSERVATION \times \{FALSE\}$

end
Event $f\_activate \mathrel{\widehat{=}}$
refines $f\_activate$
any

fct

where

`grd1` : $fct \in L\_FUNCT$
`grd2` : $f\_flag(fct\_mode(fct)) = f\_idle$
`grd3` : $f\_status(fct\_surv(fct)) \neq f\_lost$

grd4 : $f\_rqt(fct\_req(fct)) = TRUE$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_active$

end

Event $send\_rqt\_t \mathrel{\widehat{=}}$

any

fct

where

grd1 : $fct \in L\_FUNCT$

then

act1 : $f\_rqt(fct\_req(fct)) := TRUE$

end

Event $send\_rqt\_f \mathrel{\widehat{=}}$

any

fct

where

grd1 : $fct \in L\_FUNCT$

then

act1 : $f\_rqt(fct\_req(fct)) := FALSE$

end

Event $f\_execute \mathrel{\widehat{=}}$

refines $f\_execute$

any

fct

v

where

grd1 : $fct \in L\_FUNCT$

grd2 : $v \in VALUE$

grd3 : $f\_flag(fct\_mode(fct)) = f\_executing$

grd4 : $fbuffer\_h(fct\_obs(fct)) = FALSE$

grd5 : $fopbuffer\_h(fct\_obs(fct)) = FALSE$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_idle$

act2 : $fvalue(fct\_obs(fct)) := v$

act3 : $fbuffer\_h(fct\_obs(fct)) := TRUE$

end

Event $f\_execute\_r \widehat{=}$

refines $f\_execute\_r$

any

fct

res

v

where

grd1 : $fct \in L\_FUNCT$

grd2 : $fct \mapsto res \in SL\_FCT\_RES$

grd3 : $v \in VALUE$

grd4 : $f\_flag(fct\_mode(fct)) = f\_executing$

grd5 : $f\_res(fct \mapsto res) = FALSE$

grd6 : $fbuffer\_h(fct\_obs(fct)) = FALSE$

grd7 : $fopbuffer\_h(fct\_obs(fct)) = FALSE$

then

act1 : $f\_flag(fct\_mode(fct)) := f\_idle$

act2 : $f\_res(fct \mapsto res) := TRUE$

act3 : $fvalue(fct\_obs(fct)) := v$

act4 : $fbuffer\_h(fct\_obs(fct)) := TRUE$

end

Event $op\_enable\_obs \widehat{=}$

refines op_enable_obs

any

obs

where

grd1 : $obs \in F\_OBSERVATION$

grd2 : $fbuffer\_h(obs) = TRUE$

grd3 : $fopbuffer\_h(obs) = TRUE$

then

act1 : $fbuffer\_h(obs) := FALSE$

act2 : $fopbuffer\_h(obs) := FALSE$

end

Event $op\_receive\_obs \widehat{=}$

any

obs

where

grd1 : $obs \in F\_OBSERVATION$

grd2 : $fbuffer\_h(obs) = TRUE$

grd3 : $fopbuffer\_h(obs) = FALSE$

then

act1 : $fopbuffer\_h(obs) := TRUE$

end

Event $f\_detect\_err \mathrel{\widehat{=}}$

refines f_detect_err

any

fault

fct

where

grd1 : $fault \mapsto fct \in f\_fault\_class$

grd2 : $fct \in L\_FUNCT$

grd3 : $f\_flag(fct\_mode(fct)) = f\_executing$

grd4 : $f\_status(fct\_surv(fct)) = f\_ok$

then

act1 : $f\_status(fct\_surv(fct)) := f\_erroneous$

end

Event $f\_detect\_lost \mathrel{\widehat{=}}$

refines f_detect_lost

any

fault

fct

where

grd1 : $fault \mapsto fct \in f\_fault\_class$

grd2 : $fct \in L\_FUNCT$

grd3 : $f\_flag(fct\_mode(fct)) = f\_executing \lor f\_flag(fct\_mode(fct)) = f\_active$

grd4 : $f\_status(fct\_surv(fct)) \neq f\_lost$

then

act1 : $f\_status(fct\_surv(fct)) := f\_lost$

end

END

An Event-B Specification of F_OPE_2
Generated Date: 21 Oct 2011 @ 00 :50 :11 PM

MACHINE F_OPE_2

REFINES F_OPE_1

SEES UAV_CF1_3

VARIABLES

`f_flag`

`f_res`

`f_rqt`

`o_mode`

`f_status`

INVARIANTS

`inv1` : $o\_mode \in O\_MODE \rightarrow MODE\_OP$

EVENTS

Initialisation

begin

`act1` : $f\_flag :\in F\_MODE \rightarrow F\_FLAG$

`act2` : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$

`act3` : $f\_rqt :\in REQUEST \rightarrow BOOL$

`act4` : $o\_mode := O\_MODE \times \{m\_ok\}$

`act5` : $f\_status := F\_SURVEILLANCE \times \{f\_ok\}$

end

Event $select\_bu \mathrel{\widehat{=}}$

any

op

where

`grd1` : $op \in OPERATION \setminus \{OP\_bu, OP\_ca\}$

`grd4` : $o\_mode(ope\_mode(op)) = m\_ok$

then

`act1` : $o\_mode := o\_mode \Leftarrow \{ope\_mode(op) \mapsto m\_backup, ope\_mode(OP\_bu) \mapsto m\_backup\}$

end

Event $select\_ab \mathrel{\widehat{=}}$

any

op

fct

where

grd1 : $op \in OPERATION \setminus \{OP\_bu, OP\_ca\}$

grd2 : $fct \in L\_FUNCT$

grd3 : $fct \in l\_fct\_op(op)$

grd4 : $o\_mode(ope\_mode(op)) = m\_ok$

grd5 : $f\_status(fct\_surv(fct)) \neq f\_ok$

then

act1 : $o\_mode(ope\_mode(op)) := m\_aborted$

end

Event *select_ca* $\widehat{=}$

any

op

fct

where

grd1 : $op \in OPERATION \setminus \{OP\_bu, OP\_ca\}$

grd2 : $fct \in L\_FUNCT$

grd3 : $fct \in l\_fct\_op(op)$

grd4 : $o\_mode(ope\_mode(op)) = m\_ok$

grd5 : $f\_status(fct\_surv(fct)) = f\_lost$

then

act1 : $o\_mode := o\_mode \lhd \{ope\_mode(op) \mapsto m\_cancelled, ope\_mode(OP\_ca) \mapsto m\_cancelled\}$

end

Event *bu_select_ca* $\widehat{=}$

any

fct

where

grd1 : $fct \in L\_FUNCT$

grd2 : $fct \in l\_fct\_op(OP\_bu)$

grd3 : $o\_mode(ope\_mode(OP\_bu)) = m\_backup$

grd4 : $f\_status(fct\_surv(fct)) \neq f\_ok$

then

act1 : $o\_mode(ope\_mode(OP\_bu)) := m\_cancelled$

end

Event $f\_detect\_err \mathrel{\widehat{=}}$
refines f_detect_err
any
$fault$
fct
where
grd2 : $fct \in L\_FUNCT$
grd1 : $fault \mapsto fct \in f\_fault\_class$
grd3 : $f\_flag(fct\_mode(fct)) = f\_executing$
grd4 : $f\_status(fct\_surv(fct)) = f\_ok$
then
act1 : $f\_status(fct\_surv(fct)) := f\_erroneous$
end
Event $f\_detect\_lost \mathrel{\widehat{=}}$
refines f_detect_lost
any
$fault$
fct
where
grd2 : $fct \in L\_FUNCT$
grd1 : $fault \mapsto fct \in f\_fault\_class$
grd3 : $f\_flag(fct\_mode(fct)) = f\_executing \lor f\_flag(fct\_mode(fct)) = f\_active$
grd4 : $f\_status(fct\_surv(fct)) \neq f\_lost$
then
act1 : $f\_status(fct\_surv(fct)) := f\_lost$
end
END

An Event-B Specification of F_OPE_3
Generated Date: 21 Oct 2011 @ 00 :50 :18 PM

MACHINE F_OPE_3
REFINES F_OPE_2
SEES UAV_CF1_4
VARIABLES

`f_flag`
`f_res`
`f_rqt`
`o_mode`
`f_status`
`i`
`f_nb`

INVARIANTS

`inv1` : $i \in 1 \mathinner{..} n$
`inv2` : $f\_nb \in SS\_FUNCTION \rightarrow 0 \mathinner{..} 2$

EVENTS
Initialisation

begin

`act1` : $f\_flag :\in F\_MODE \rightarrow F\_FLAG \setminus \{f\_off\}$
`act2` : $f\_res :\in SL\_FCT\_RES \rightarrow BOOL$
`act3` : $f\_rqt :\in REQUEST \rightarrow BOOL$
`act4` : $o\_mode := O\_MODE \times \{m\_ok\}$
`act5` : $f\_status := F\_SURVEILLANCE \times \{f\_ok\}$
`act6` : $i := 1$
`act7` : $f\_nb := SS\_FUNCTION \times \{2\}$

end

Event $select\_bu \mathrel{\widehat{=}}$
refines $select\_bu$

any

fct

where

`grd1` : $i < n$
`grd2` : $fct \in L\_FUNCT$
`grd3` : $fct \in l\_fct\_op(seq\_op(i)) \cap l\_fct\_op(seq\_op(i+1))$

grd4 : $o\_mode(ope\_mode(seq\_op(i))) = m\_ok$
grd5 : $f\_status(fct\_surv(fct)) = f\_erroneous$
grd6 : $f\_flag(fct\_mode(fct)) \neq f\_off$

with

op : $\mathtt{op} = \mathtt{seq\_op(i)}$

then

act1 : $o\_mode := o\_mode \lhd \{ope\_mode(seq\_op(i)) \mapsto m\_backup, ope\_mode(OP\_bu) \mapsto m\_backup\}$
act2 : $i := i + 1$

end

Event *select_ab* $\widehat{=}$
refines *select_ab*

any

fct

where

grd1 : $i < n$
grd2 : $fct \in L\_FUNCT$
grd3 : $fct \in l\_fct\_op(seq\_op(i))$
grd7 : $fct \notin l\_fct\_op(seq\_op(i+1))$
grd4 : $o\_mode(ope\_mode(seq\_op(i))) = m\_ok$
grd6 : $o\_mode(ope\_mode(seq\_op(i+1))) = m\_ok$
grd5 : $f\_status(fct\_surv(fct)) \neq f\_ok$
grd8 : $f\_flag(fct\_mode(fct)) \neq f\_off$

with

op : $\mathtt{op} = \mathtt{seq\_op(i)}$

then

act1 : $o\_mode(ope\_mode(seq\_op(i))) := m\_aborted$
act2 : $i := i + 1$

end

Event *select_ca* $\widehat{=}$
refines *select_ca*

any

fct

where

grd1 : $i < n$

grd2 : $fct \in L\_FUNCT$

grd3 : $fct \in l\_fct\_op(seq\_op(i)) \cap l\_fct\_op(seq\_op(i+1))$

grd4 : $o\_mode(ope\_mode(seq\_op(i))) = m\_ok$

grd5 : $f\_status(fct\_surv(fct)) = f\_lost$

grd6 : $f\_flag(fct\_mode(fct)) = f\_off$

grd7 : $f\_nb(fct\_cat(fct)) = 0$

with

op : $\mathtt{op} = \mathtt{seq\_op(i)}$

then

act1 : $o\_mode := o\_mode \Leftarrow \{ope\_mode(seq\_op(i)) \mapsto m\_cancelled, ope\_mode(OP\_ca) \mapsto m\_cancelled\}$

act2 : $i := i + 1$

end

Event $bu\_select\_ca \mathrel{\widehat{=}}$

refines bu_select_ca

any

fct

where

grd5 : $i < n$

grd1 : $fct \in L\_FUNCT$

grd2 : $fct \in l\_fct\_op(OP\_bu)$

grd3 : $o\_mode(ope\_mode(OP\_bu)) = m\_backup$

grd4 : $f\_status(fct\_surv(fct)) \neq f\_ok$

grd6 : $f\_flag(fct\_mode(fct)) \neq f\_off$

then

act1 : $o\_mode(ope\_mode(OP\_bu)) := m\_cancelled$

act2 : $i := i + 1$

end

Event $select\_ok \mathrel{\widehat{=}}$

when

grd1 : $i < n$

grd2 : $o\_mode(ope\_mode(seq\_op(i))) = m\_ok$

grd3 : $o\_mode(ope\_mode(seq\_op(i+1))) = m\_ok$

then

act1 : $i := i + 1$

end

Event *select_bu2* $\widehat{=}$

refines *select_bu*

when

grd1 : $i < n$

grd2 : $o\_mode(ope\_mode(seq\_op(i))) = m\_ok$

grd3 : $o\_mode(ope\_mode(seq\_op(i+1))) = m\_aborted$

with

op : $\mathtt{op} = \mathtt{seq\_op(i)}$

then

act1 : $o\_mode := o\_mode \lhd \{ope\_mode(seq\_op(i)) \mapsto m\_backup, ope\_mode(OP\_bu) \mapsto m\_backup\}$

act2 : $i := i + 1$

end

END

Annexe B

Modélisation du cas d'étude en AltaRica

On a entrepris une modélisation parallèle du système de contrôle du RMAX en AltaRica. Cette modélisation est illustrée par l'exemple suivant représentant la phase opérationnelle d'un drone. Le composant générique *Phase_Source* permet l'initialisation des phases définies pour une mission donnée du drone. Il contient une variable d'état et trois flux de sortie qui évoluent comme suit :

- l'état booléen *on* est initialisé à *false*. Il passe à *true* dès que la mission a démarré à l'aide d'un événement déterministe *startMission*. Cette valeur est propagée à la première phase de la mission par le flux de sortie *go* ;
- l'entier naturel *timeZero* indique le temps écoulé avant l'entrée dans une phase ; dans cet exemple, l'assertion attribue une valeur nulle à *timeZero* ;
- la variable *sce* de type énuméré *OR_OperationalImpact* stipule dans quelles conditions la phase suivante est activée (sans interruption ou à la suite d'une annulation).

Algorithme B.1 Composant générique de phase opérationnelle en AltaRica

```
domain OR_OperationalImpact = {nointerrupt, cancel};

node Phase_Source
// declaration of node parameters
 flow
   go : bool : out;
   timeZero : int : out;
   sce : OR_OperationalImpact : out;
 state
   on : bool;
 event
   startMission;
// automata initialization and transitions
 init
   on := false;
 trans
   on = false -startMission-> on := true;
// assertions
 assert
   go = on,
   timeZero = 0,
   sce = nointerrupt;
edon
```

Une première modélisation de l'architecture de contrôle du drone ONERA a été réalisée en AltaRica (figure B.1). Cette modélisation facilite l'expression et l'évaluation de l'impact d'une faute sur le comportement et sur la mission du drone. On y retrouve l'architecture en couches instanciée avec :

- des composants relatifs à la gestion de la mission et à la gestion des opérations de contrôle modélisant le gestionnaire de la charge utile (*mis_managt*), le module de plannification (*dec_planning*), le module de supervision (*superv*), le module de contrôleur d'exécution (*exe_control*) ;
- des composants fonctionnels de contrôle tels que la navigation (*nav*), le guidage (*guidance*), les fonctions de pilotage manuel et automatique (*man*, *ap1* et *ap2*), l'estimateur d'états continus (*st_estimation*) et la fonction de relais des acquisitions de mesures et d'envoi de commandes (*acq_command*) ;
- les capteurs et actionneurs sont modélisés sous l'image l'hélicoptère ;
- des composants connexes de communication (*com*) et de contrôle de la charge utile (*command_com*) sont également modélisés.

Cette architecture s'inspire également des architectures fonctionnelles en couches décrites par Alami [33] adaptées au niveaux d'autonomie formulés par Fargeon [3] en robotique mobile.

L'exploitation de cette modélisation en simulation a permis de valider le cheminement d'une défaillance à travers divers composants du système, l'événement redouté observé étant l'annulation d'une phase opérationnelle. L'interface graphique de l'atelier Cecilia OCAS développé par Dassault Aviation s'est avé-

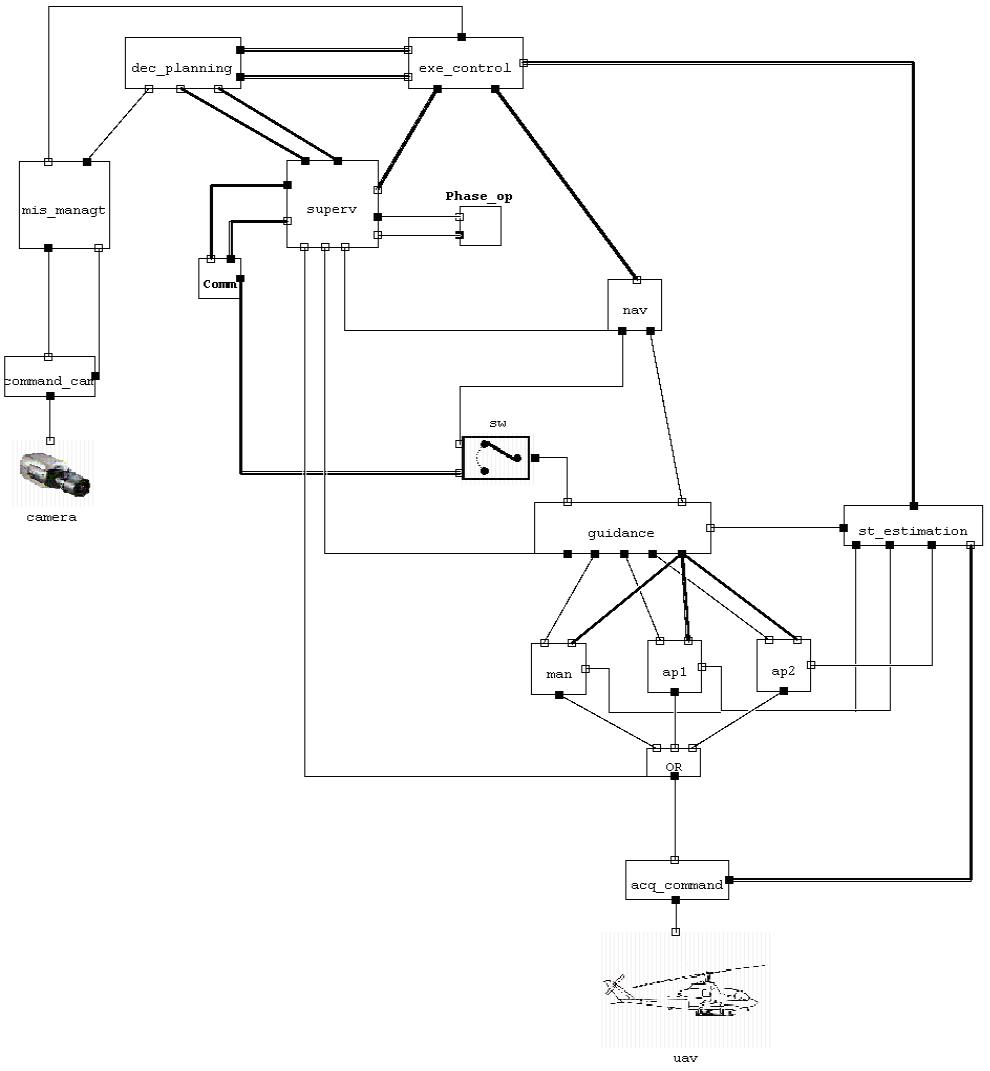

FIG. B.1 – Modélisation AltaRica du système de contrôle du drone RMAX.

rée très pratique lors de la vérification en simulation. L'activation de chaque défaillance de chaque composant valide les connections et les dépendances dysfonctionnelles existantes ainsi que les configurations fonctionnelles déterminant des modes opérationnels. Ainsi, une défaillance peut conduire à l'événement redouté sous l'hypothése de sélection d'un mode particulier, alors que la sélection d'un autre mode constitue une mitigation de la défaillance. De même, l'analyse des séquences de défaillances à l'aide de l'outil de génération de coupes minimales indique l'ensemble des combinaisons de pannes et de sélection de modes correspondant à l'apparition de l'événement redouté. Le résultat d'une des générations de séquences est enregistré dans le fichier *test.seq* dont voici un extrait :

Séquence non minimale + ordre 3 + combinaison

products(Basic('Phase_ op.MissionStatus.cancel')) =
'Phase_op.Cruise1.abort', 'acq_command.Error', 'acq_command.Loss'
'Phase_op.Cruise1.abort', 'acq_command.Error', 'dec_planning.Loss_c'
'Phase_op.Cruise1.abort', 'acq_command.Error', 'st_estimation.Loss_p'
'Phase_op.Cruise1.abort', 'acq_command.Error', 'superv.Loss_d'
'Phase_op.Cruise1.abort', 'acq_command.Loss'
'Phase_op.Cruise1.abort', 'ap1.Error', 'dec_planning.Loss_c'
'st_estimation.Error_p', 'superv.Loss_d'
'st_estimation.Loss'
'superv.Error_o', 'superv.Loss_d'
'superv.Loss_d'
'man.Error', 'superv.Loss_d'
'man.Loss'
'guidance.Error', 'man.Error', 'superv.Loss_d'
'guidance.Error', 'man.Loss'
'dec_planning.Error_m', 'guidance.selAP1', 'superv.Loss_d'
'dec_planning.Error_m', 'man.Error', 'superv.Loss_d'
'dec_planning.Error_m', 'man.Loss'
...

Tableau B.1 – Résultats AltaRica : séquences d'événements menant à l'événement redouté.

Cette liste de séquences indique des combinaisons non minimales d'ordre 3 de pannes et d'événements de sélection de configurations fonctionnelles qui mènent à l'annulation de la mission. On constate que des pannes simples peuvent conduire à l'événement redouté d'où la nécessité d'établir des redondances fonctionnelles. De plus, les pannes sur les équipements tels que les capteurs inertiels, GPS et les servomoteurs, les relais ou les équipements d'alimentation électrique ne sont pas prises en compte dans cette analyse. Des études plus poussées ont

été menées avec des stagiaires pour se rendre compte des types de redondance matérielle à mettre en place.

Les modélisations AltaRica et Event-B ont été réalisées en parallèle autorisant ainsi une fertilisation croisée de notre compréhension du comportement du système de contrôle du drone en présence de fautes. Les principaux atouts des modèles AltaRica résident dans sa simplicité d'utilisation, dans son interface graphique et dans sa panoplie d'outils d'analyse et d'évaluation de la propagation de pannes. Les modèles Event-B ont apporté une méthode progressive de modélisation s'appuyant sur des propriétés logiques clairement spécifiées et validées par décharge de preuves.

Bibliographie

[1] J. de Rosnay : *Le macroscope : Vers une vision globale*. Seuil, 1977.

[2] E. Chanthery : *Planification de mission pour un véhicule autonome*. SUPAERO, Toulouse, FR, 2005.

[3] Catherine Fargeon et Jean-Philippe Quin : *Robotique mobile*. Teknea, 1993.

[4] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell et Carl Landwehr : Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Secur. Comput.*, 1:11–33, January 2004.

[5] Alain Pagès et Michel Gondran : *Fiabilité des systèmes*. Editions Eyrolles, 1980.

[6] Alain Villemeur : *Sûreté de fonctionnement des systèmes industriels : Fiabilité - Facteurs humains - Informatisation*. Editions Eyrolles, 1988.

[7] Jean-Claude Laprie : *Guide de la sûreté de fonctionnement*. Editions Cépaduès, 1995.

[8] Leslie Lamport, Robert Shostak et Marshall Pease : The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[9] Gérard Le Lann : The Ariane 5 Flight 501 Failure - A Case Study in System Engineering for Computing Systems. *Rapport de recherche n°3079*, 1996.

[10] C. Lievens : *Sécurité des Systèmes*. Editions Cépaduès, 1976.

[11] SAE : ARP 4754a : Guidelines for Development of Civil Aircraft and Systems, 2010.

[12] SAE International. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, 1996.

[13] Ch. Kehren et C. Seguin : Evaluation qualitative de systèmes physiques pour la sûreté de fonctionnement. *In Formalisation des Activités Concurrentes, FAC'03*, 2003.

[14] Ragavan Manian, Joanne Bechta Dugan, David Coppit et Kevin J. Sullivan : Combining Various Solution Techniques for Dynamic Fault Tree Analysis of Computer Systems. *High-Assurance Systems Engineering, IEEE International Symposium on*, 0:21, 1998.

[15] Marc Bouissou et Jean-Louis Bon : A new formalism that combines advantages of fault-trees and Markov models : Boolean logic Driven Markov Processes. *Reliability Engineering and System Safety*, 82(2):149 – 163, 2003.

[16] Marko Cepin et Borut Mavko : A Dynamic Fault Tree. *Reliability Engineering and System Safety*, 75:83–91, 2002.

[17] N. Halbwachs, P. Caspi, P. Raymond et D. Pilaud : The synchronous dataflow programming language LUSTRE. *In Proceedings of the IEEE*, pages 1305–1320, 1991.

[18] James Lyle Peterson : *Petri Net Theory and the Modeling of Systems.* Prentice Hall, 1981.

[19] J.-P. Meinadier : *Ingénierie et intégration des systèmes.* Hermès, 1998.

[20] C. Kehren : *Motifs formels d'architectures de systèmes pour la sûreté de fonctionnement.* SUPAERO, Toulouse, FR, 2005.

[21] Projet SIRASAS : Stratégies Innovantes et Robustes pour l'Autonomie des Systèmes Aéronautiques et Spatiaux. sous l'égide de la Fondation de Recherche pour l'Aéronautique et l'Espace, 2007 - 2010.

[22] Anish Arora et Sandeep S. Kulkarni : Detectors and Correctors : A Theory of Fault-Tolerance Components. *In International Conference on Distributed Computing Systems*, pages 436–443, 1998.

[23] A. Arora et S.S. Kulkarni : Component based design of multitolerant systems. *Software Engineering, IEEE Transactions on*, 24(1):63–78, 1998.

[24] E. Bensana et P. Ribot : *Projet Hélimaintenance R et D Lot 4.* ONERA, FR, 2011. RTS 5/14019 DCSD.

[25] Philippe Charbonnaud : supervision, surveillance et diagnostic. *In Notes de cours, M2R SAID*, 2004.

[26] Igor Nikiforov : Estimation et Détection pour les Systèmes Embarqués : Approche statistique. *In Notes de cours, Ecole Doctorale Systèmes - ED 309*, 2010.

[27] Rosario TOSCANO : *Commande et diagnostic des systèmes dynamiques*. Ellipses, 2005.

[28] Louise TRAVÉ-MASSUYÈS : Bridging FDI and DX Model Based Diagnosis for Continuous Systems. *In Notes de cours, Ecole Doctorale Systèmes*, 2008.

[29] Louise TRAVÉ-MASSUYÈS : A unified formulation of diagnosability for CS and DES : perspectives for Hybrid Systems. *In Notes de cours, Ecole Doctorale Systèmes*, 2008.

[30] Yannick PENCOLÉ : Diagnosticabilité des systèmes à événements discrets. *In Notes de cours, Ecole Doctorale Systèmes*, 2008.

[31] Alban GRASTIEN et ANBULAGAN : Diagnostic de systèmes à événements discrets à base de cohérence par SAT. *Revue d'Intelligence Artificielle*, vol. 24(no 6):pages 757–786, 2010.

[32] C. E. SHANNON : A Mathematical Theory of Communication. *the Bell System Technical Journal*, 1948.

[33] R. ALAMI, R. CHATILA, S. FLEURY, M. GHALLAB et F. INGRAND : An Architecture for Autonomy. *International Journal of Robotics Research*, 17:315–337, 1998.

[34] Elena TROUBITSYNA : Enhancing Dependability via Parameterized Refinement. *In Pacific Rim International Symposium on Dependable Computing*, pages 120–127. Hong Kong, China, December 1999.

[35] Elena TROUBITSYNA : Integrating Safety Analysis into Formal Specification of Dependable Systems. *In International Annual IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*. Nice, France, April 2003.

[36] E. TROUBITSYNA et L. LAIBINIS : Fault Tolerance in a Layered Architecture : a General Specification Pattern in B. *Turku Centre for Computer Science Technical Report no 609*, pages 1–40, 2004.

[37] Linas LAIBINIS et Elena TROUBITSYNA : Refinement of fault tolerant control systems in B. *In International conference on Computer Safety, Reliability, and Security - SAFECOMP'2004*, volume 3219, pages 254–268. Potsdam, Germany, Springer Verlag, September 2004.

[38] Dubravka ILIC et Elena TROUBITSYNA : Modelling Fault Tolerance of Transient Faults. *In Workshop on Rigorous Engineering of Fault-Tolerant Systems (REFT 2005)*, pages 84–92. Newcastle Upon Tyne, UK, July 2005.

[39] D. Ilic, E. Troubitsyna, L. Laibinis et S. Leppänen : Formal Verification of Consistency in Model-Driven Development of Distributed Communicating Systems and Communication Protocols. *In ISoLA*, pages 425–432, 2006.

[40] Alexei Iliasov, Alexander Romanovsky, Budi Arief, Linas Laibinis et Elena Troubitsyna : Rigorous Design and Implementation of Fault Tolerant Ambient Systems. *In 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC07)*. Santorini, Greece, May 2007.

[41] Elena Troubitsyna : Elicitation and Specification of Safety Requirements. *In The Third International Conference on Systems - ICONS'08*, April 2008.

[42] A. Iliasov, A. Romanovsky et F. L. Dotti : Structuring Specifications with Modes. *In Dependable Computing, 2009. LADC '09. Fourth Latin-American Symposium*, pages 81 –88, 2009.

[43] David Powell : Failure Mode Assumptions and Assumption Coverage. *In 22nd IEEE International Symposium on Fault Tolerant Computing (FTCS 22)*, pages 386 – 395, 1992.

[44] J.F. Allen : Maintaining Knowledge about Temporal Intervals. *In Communications of the ACM*, volume 26, pages 832–843, 1983.

[45] H. Bestougeff et G. Ligozat : *Outils logiques pour le traitement du temps : de la linguistique à l'intelligence artificielle*. Masson, 1989.

[46] Alan M. Turing : Checking a Large Routine. *In Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, 1949.

[47] C. A. R. Hoare : An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, octobre 1969.

[48] E.W. Dijkstra : A Discipline of Programming. *Prentice-Hall Series in Automatic Computation*, 1976.

[49] C.C. Morgan : Programming from Specification. *Prentice-Hall Series*, 1990.

[50] Jacques Julliand : *Cours et exercices corrigés d'algorithmique - Vérifier, tester et concevoir des programmes en les modélisant*. Editions Vuibert, 2010.

[51] E. Feron : From Control Systems to Control Software - Integrating Lyapunov-theoretic proofs within code. *Control Systems Magazine, IEEE*, 30(6):50 –71, 2010.

[52] J.-R. Abrial : *The B-book : Assigning Program to Meanings*. Cambridge University Press, 1996.

[53] Christos G. Cassandras et Stephane Lafortune : *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[54] R.-J. Back et J. von Wright : *Refinement Calculus : A Systematic Introduction*. Springer-Verlag, 1998.

[55] J.-R. Abrial : *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, 2010.

[56] Dominique Cansell et Dominique Méry : Abstraction and refinement of features. *In* Ryan Stephen, Gilmore et Mark, éditeur : *Language Constructs for Designing Features*. Springer.

[57] J.-R. Abrial : Refinement, Decomposition and Instantiation of Discrete Models. *Fundamentae Informatica*, 77:2006, 2006.

[58] M. Butler : Incremental Design of Distributed Systems with Event-B. *In Engineering Methods and Tools for Software Safety and Security - Marktoberdorf Summer School 2008*, pages 131 –160. IOS Press, 2009.

[59] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede et Laurent Voisin : An Open Extensible Tool Environment for Event-B. *In* Zhiming Liu et Jifeng He, éditeurs : *Formal Methods and Software Engineering*, volume LNCS 4260/2006, pages 588–605. Springer Berlin / Heidelberg, November 2006.

[60] A. Arnold, A. Griffault, G. Point et A. Rauzy : The Altarica formalism for describing concurrent systems. *Fundamenta Informaticae n°40*, pages pages 109–124, 2000.

[61] S. Humbert, J.-M. Bosc, C. Castel, P. Darfeuil, Y. Dutuit, E. Focone et C. Seguin : Déclinaison d'exigences de sécurité du système vers le logiciel, assistée par des modèles formels. *In Approches Formelles dans l'Assistance au Développement de Logiciels*, 2007.

[62] A. Rauzy : Mode automata and their compilation into fault trees. *Reliability Engineering and System Safety, 78*, pages pages 1–12, 2002.

[63] Alessandra Russo, Rob Miller, Bashar Nuseibeh et Jeff Kramer : An Abductive Approach for Handling Inconsistencies in SCR Specifications. *In (ICSE2000) International Workshop on Intelligent Software Engineering*. Limerick, Ireland, 2000.

[64] David L. Parnas et Jan. Madey : Functional Documents for Computer Systems. *In Science of Computer Programming*, volume 25, pages 41 –51, McMaster University, Hamilton, ON, Canada, 1995.

[65] Ralph Jeffords, Constance Heitmeyer, Myla Archer et Elizabeth Leonard : A Formal Method for Developing Provably Correct Fault-Tolerant Systems Using Partial Refinement and Composition. *In Lecture Notes in Computer Science. Formal Methods, 2009*, volume 5850/2009, pages 173 –189, 2009.

[66] S. Owre, J. Rushby, N. Shankar et F. von Henke : Formal Verification for Fault-Tolerant Architectures : Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering*, 21:107–125, 1995.

[67] S. Owre, N. Shankar, J. M. Rushby et D. W. J. Stringer-Calvert : *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, 1999.

[68] César Muñoz : PBS : Support for the B-Method in PVS, 1999.

[69] Roland Atoui, Xavier Dumas, Sébastien Jardel et Laurent Vendredi : Conception Formelle en PVS, 2007.

[70] J.M. Zhou, Jian Guo et Fu Song : Integrating the B-method into PVS. *In International Conference on Information Engineering and Computer Science, 2009. ICIECS 2009*, pages 1 – 4, 2009.

[71] S. Lemai, M.-C. Charmeau et X. Olive : Intégrer des planificateurs dans le logiciel de vol d'un satellite autonome. *In JFPDA'06 - Journées Francophones Planification, Décision, Apprentissage pour la conduite de système*. Toulouse, FR, 2006.

[72] Ch. Castel, J.-F. Gabard et C. Tessier : *Etude de la problématique FDIR du vol en formation et impact sur l'architecture de commande/contrôle - Synthèse de l'étude*. ONERA, FR, 2005. RTS 4/1079601F DCSD.

[73] D. Sabatier, B. Dellandrea et D. Chemouil : FDIR Strategy Validation with the B Method. *In The International Space System Engineering Conference - DASIA*, 2008.

[74] J.-Ch. Chaudemar : *FDIR d'une formation de satellites : spécification formelle basée sur les méthodes Z et réseaux de Petri*. ONERA-SUPAERO, FR, 2007.

[75] Saddek Bensalem, Lavindra de Silva, Matthieu Gallien, Félix Ingrand et Rongjie Yan : A Verifiable and Correct by Construction Controller for Robots in Human Environment. *In 7th IARP Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE)*, Toulouse, FR, 2010.

Étude des architectures de sécurité de systèmes autonomes : formalisation et évaluation en Event B

La recherche de la sûreté de fonctionnement des systèmes complexes impose une démarche de conception rigoureuse. Les travaux de cette thèse s'inscrivent dans le cadre la modélisation formelle des systèmes de contrôle autonomes tolérants aux fautes. Le premier objectif a été de proposer une formalisation d'une architecture générique en couches fonctionnelles qui couvre toutes les activités essentielles du système de contrôle et qui intègre des mécanismes de sécurité. Le second objectif a été de fournir une méthode et des outils pour évaluer qualitativement les exigences de sécurité.
Le cadre formel de modélisation et d'évaluation repose sur le formalisme Event-B. La modélisation Event-B proposée tire son originalité d'une prise en compte par raffinements successifs des échanges et des relations entre les couches de l'architecture étudiée. Par ailleurs, les exigences de sécurité sont spécifiées à l'aide d'invariants et de théorèmes. Le respect de ces exigences dépend de propriétés intrinsèques au système décrites sous forme d'axiomes. Les preuves que le principe d'architecture proposé satisfait bien les exigences de sécurité attendue ont été réalisées avec les outils de preuve de la plateforme Rodin. L'ensemble des propriétés fonctionnelles et des propriétés relatives aux mécanismes de tolérance aux fautes, ainsi modélisées en Event-B, renforce la pertinence de la modélisation adoptée pour une analyse de sécurité. Cette approche est par la suite mise en œuvre sur un cas d'étude d'un drone ONERA.

Mots clés : méthode formelle, Event-B, raffinement, architectures tolérantes aux fautes, sécurité

Model based safety of FDIR architectures for autonomous systems: formal specification and assessment with Event-B

The study of complex system safety requires a rigorous design process. The context of this work is the formal modeling of fault tolerant autonomous control systems. The first objective has been to provide a formal specification of a generic layered architecture that covers all the main activities of control system and implement safety mechanisms. The second objective has been to provide tools and a method to qualitatively assess safety requirements.
The formal framework of modeling and assessment relies on Event-B formalism. The proposed Event-B modeling is original because it takes into account exchanges and relations between architecture layers by means of refinement. Safety requirements are first specified with invariants and theorems. The meeting of these requirements depends on intrinsic properties described with axioms. The proofs that the concept of the proposed architecture meets the specified safety requirements were discharged with the proof tools of the Rodin platform. All the functional properties and the properties relating to fault tolerant mechanisms improve the relevance of the adopted Event-B modeling for safety analysis. Then, this approach is implemented on a study case of ONERA UAV.

Keywords: formal method, Event-B, refinement, fault tolerant architectures, safety, dependability