Analyse de sécurité de systèmes autonomes: Formalisation et évaluation en Event-B

Jean-Charles Chaudemar*, Eric Bensana**, Christel Seguin**

*ISAE-DMIA - 10 av. Edouard Belin, Toulouse, France
jean-charles.chaudemar@isae.fr
**ONERA-DCSD - 2 av. Edouard Belin, Toulouse, France
eric.bensana, christel.seguin@onera.fr

Résumé. Cet article présente une partie de l'étude d'architectures de sécurité de systèmes autonomes s'appuyant sur l'utilisation de la méthode formelle Event-B. Le formalisme Event-B supporte bien la conception rigoureuse de ces systèmes qui combinent diverses activités que l'on peut structurer en couches. Sa technique de raffinement permet une modélisation progressive en vérifiant la correction et la pertinence des modèles par décharge de preuves.
L'application de la méthode Event-B dans le cadre de la spécification d'architectures en couches garantit l'émergence de propriétés globales attendues, telles que les propriétés de sécurité, lorsque l'on s'assure du respect de propriétés au niveau des relations entre les couches.
Cet article se situe au début de cette nouvelle étude. Il présente les principes de la modélisation Event-B d'un système de contrôle de drone simplifié. Il caractérise le concept d'architecture en couches utilisée pour cette modélisation. Il décrit ensuite une première modélisation d'une couche avant de conclure sur l'intérêt de cette modélisation pour la validation de systèmes autonomes par rapport aux objectifs de sécurité fixés.

1 Introduction

Les systèmes de contrôle sûrs de fonctionnement d'engins autonomes sont des systèmes complexes composés d'éléments issus de différents domaines métiers tels que la mécanique, les facteurs humains, ou le logiciel. Une organisation hiérarchique en couches de ces systèmes de contrôle permet alors de mieux les appréhender selon Troubitsyna et Laibinis (2004). Mais, cette organisation en couches impose une rigueur particulière dans la conception des interfaces entre les couches pour assurer la pertinence du système et le respect des exigences de sécurité. Event-B est une méthode formelle qui a été développée pour spécifier correctement, et modéliser itérativement des systèmes complexes par un mécanisme de raffinement comme indiqué par Abrial (2010). Le raffinement est une technique incrémentale visant à transformer un modèle abstrait en un modèle concret i.e. un modèle contenant plus de détails dans sa spécification ou étant plus près d'une implémentation.
Cet article vise à proposer une approche, par la méthode Event-B, de spécification formelle de l'architecture en couches du système de contrôle d'un drone dans le cadre d'une analyse de

sécurité. L'expression de propriétés propres aux relations entre les couches permet de spécifier correctement le comportement nominal et anormal du système de contrôle du drone.
Cet article est organisé de la façon suivante : dans la section suivante, nous présenterons le cas d'étude du système de contrôle d'un drone autonome. Puis, nous exposerons les modèles en Event-B de l'architecture en couches de ce système ainsi que les propriétés associées. Ensuite, nous évoquerons les premières vérifications réalisées pour garantir l'exactitude de ces modèles et le respect des objectifs de sécurité.

2 Présentation du cas d'étude

2.1 Contexte opérationnel

Le cadre applicatif de cette étude concerne le système de contrôle avionique du drone autonome RMAX développé par l'ONERA. Cette avionique implémente trois modes de pilotage qui sont sélectionnés manuellement ou de manière autonome. Le premier mode de pilotage correspond à un pilotage manuel. Le second mode est relatif à un pilotage automatique grossier, tandis que le troisième mode 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 et la supervision 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, que l'on détaillera dans la sous-section suivante.
La mission de ce drone est constituée de plusieurs phases opérationnelles combinées aux différents modes de pilotage et à des zones survolées : décollage automatique, vol d'avancement à vue en zone non habitée, vol stationnaire hors vue en zone habitée, atterrissage manuel.
Notre objectif étant d'évaluer qualitativement et quantitativement les conditions de faute sur notre système, on considère alors un scénario opérationnel depuis l'allumage moteur jusqu'à l'atterrissage de l'engin, en se focalisant sur la phase opérationnelle la plus critique, qui est celle d'un vol hors vue au-dessus d'une zone habitée.

2.2 Architecture en couches

D'après de Rosnay (1977), la définition d'une 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 (voir Alami et al. (1998)). Dans notre cas, l'architecture décrite par Chaudemar et al. (2009), est constituée de 3 couches fonctionnelles : opération, fonction, équipement. 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 du drone. 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 guidage. 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). 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 mais 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 immédiatement inférieure. De même, la couche fonction contrôle et surveille les activités des équipements.
L'architecture en couches joue également un rôle essentiel dans la sécurité du système selon Troubitsyna et Laibinis (2004). Dans notre approche, des barrières de sécurité en termes de détection de fautes, d'isolation et de reconfiguration (mécanismes communément appelés FDIR pour Fault Detection Isolation and Reconfiguration) 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 vers un mode dégradé (*Back*) en poursuivant la mission, ou bien vers un mode avorté (*Abort*) en effectuant un "retour à la base" ou un retour vers un point géographique préétabli, ou bien 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.
La figure 1 montre bien la dualité du comportement du système dans le cas nominal et dans le cas anormal au sein de notre architecture en couches. La partie droite de cette figure représente les automates de modes de défaillances associés à chaque composant situé dans la partie gauche suivant de la couche considérée.
Dans la sous-section suivante, on présente les propriétés de sécurité à vérifier par notre système en s'appuyant sur l'architecture en couches.

2.3 Propriétés de sécurité du système

Le processus d'analyse de sécurité permet de créditer le système d'une plus grande confiance sur sa sûreté de fonctionnement et sur son innocuité. Cette analyse repose sur des propriétés ou exigences de sécurité à satisfaire par le système. 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. Ne pouvant pas se prévaloir d'une expertise statistique détaillée sur les types d'anomalies et de catastrophes liées aux drones, on se base sur la réglementation internationale appliquée aux avions civils. Dans notre cas d'étude, on considère l'événement ayant l'effet le plus catastrophique comme étant l'événement de crash du drone dans une zone habitée. De

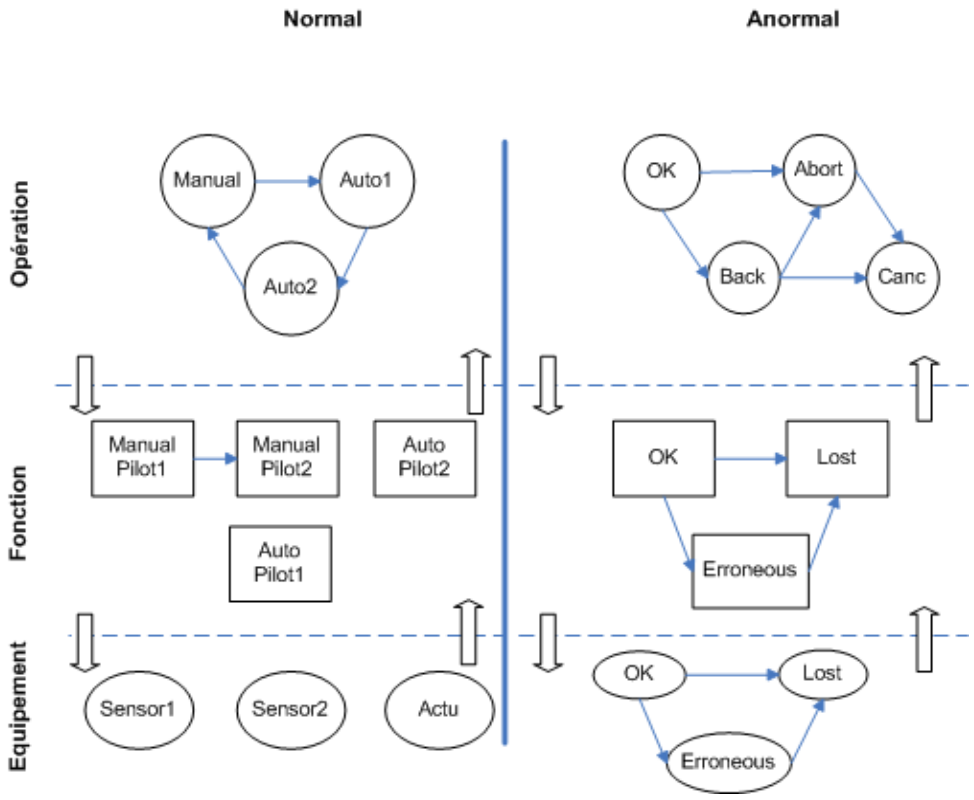

FIG. 1 – *Architecture en couches du drone.*

ce fait, on déduit deux principales propriétés de sécurité à satisfaire, l'une étant qualitative, l'autre quantitative. La propriété de sécurité qualitative est ainsi formulée : «une faute simple ne doit pas conduire à l'événement redouté de crash dans une zone habitée». De même, la propriété de sécurité quantitative s'exprime en ces termes : «la probabilité d'occurrence de l'événement redouté de crash doit être inférieure à 10^{-9} par heure de vol». Les normes de développement des systèmes évoluant, l'utilisation de méthodes formelles rend plus aisées la vérification et la validation de ces propriétés. Par conséquent, pour la spécification et la formalisation du système de contrôle du drone, on a choisi d'utiliser conjointement les méthodes formelles AltaRica, spécialisé dans l'étude de propagation de défaillances (voir Arnold et al. (2000); Rauzy (2002)), et Event-B (voir Abrial (2010)).

Dans cet article, on s'intéresse essentiellement à la spécification en Event-B du système de contrôle pour le respect de la propriété de sécurité qualitative énoncée ci-dessus.

Cependant, d'autres propriétés interviennent dans cette architecture pour maintenir l'intégrité fonctionnelle du système. L'importance de ces propriétés transparaît au niveau des relations entre les couches. En effet, la coordination des activités entre les couches est primordiale dans le comportement sûr du drone. Cette coordination se traduit par la fourniture de "services"

d'une couche pour le besoin de la couche immédiatement supérieure. Un service correspond à l'activité coordonnée et collective de certains composants de la couche fournissant ce service. Toutes activités des éléments de la couche en question ne sont pas des services : seules les activités qui sont visibles par la couche supérieure sont des services. Un service peut être vu comme une machine abstraite qui génère une fonctionnalité spécifique de la couche supérieure. La fourniture d'un service peut nécessiter l'utilisation de services de plus bas niveau. Ainsi, les services de FDIR utilisés dans une couche peuvent par exemple reposer sur l'existence d'équipements redondants ou indépendants que l'on décide d'activer ou pas.
En rapport avec ces services, on distingue trois grandes familles de propriétés utiles pour garantir la sécurité dans chaque couche :

1. Propriétés relatives aux aspects statiques d'un service
   - "Sélection" des événements utiles à l'établissement d'un service ;
   - "indépendance" entre événements dans une couche ;
   - "équivalence" entre événements dans une couche ;
   - "redondance" définie comme étant la conjonction des propriétés d'indépendance et d'équivalence.
2. Propriétés relatives aux aspects dynamiques d'un service
   - "Contraintes temporelles" entre événements décrites dans Allen (1983) et, Bestougeff et Ligozat (1989).
3. Propriétés relatives à la connaissance des composants
   - "Observabilité" de l'état des composants.

Dans la section suivante, on donne un aperçu de la méthode envisagée pour spécifier ces propriétés et garantir leur respect pendant la conception du système de contrôle du drone.

3 Modélisation en Event-B

3.1 Approche adoptée

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 permettant de maîtriser la complexité du système par un développement progressif et sûr.
Par ailleurs, ces principes de la méthode Event-B sont adaptés pour la spécification de notre architecture et ont été appliqués dans une étude de spécification en B de la tolérance aux fautes dans une architecture en couches décrite par Troubitsyna et Laibinis (2004). L'approche abordée par les auteurs de cette étude consiste à considérer la couche la plus haute comme étant une abstraction du système. La modélisation est raffinée en ajoutant progressivement les couches inférieures successives. Dans une couche, les composants gèrent le comportement des composants de la couche inférieure. Le principe du raffinement dans cette étude consiste à y inclure (clause "includes" en B) le modèle du composant de la couche immédiatement inférieure et ainsi de suite.
Par contre, dans notre approche, on considère que les propriétés définies dans la section précédente sont primordiales pour traduire les relations entre les entités des couches de l'architecture. La notion de "services" impose alors un raffinement dont la machine abstraite est le

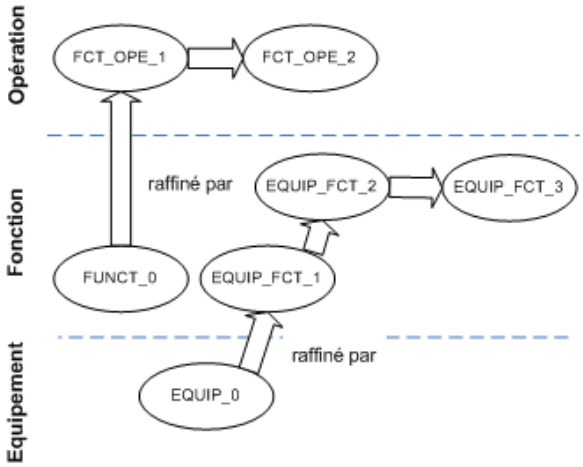

FIG. 2 – *Mécanisme de raffinement adopté.*

modèle de la couche de niveau inférieur. En effet, dans une relation entre deux couches successives, 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 2 met en évidence l'approche de modélisation Event-B adoptée pour spécifier les relations entre les couches. En partant de modèles abstraits (*EQUIP_0*, *FUNCT_0*), on raffine notre spécification jusqu'à considérer le cas concret du système de contrôle du drone autonome RMAX (éventuellement jusqu'aux modèles *EQUIP_FCT_3*, *FCT_OPE_2*). De plus, l'application d'un mécanisme de fusion de modèles (par exemple, entre les modèles *FUNCT_0* et *EQUIP_FCT_1*) permettra de consolider la spécification de notre architecture en couches comme indiqué par Abrial (2006) et Butler (2009).

3.2 Modélisation de la couche Fonction

Comme énoncé précédemment, la philosophie de modélisation de la couche fonction repose sur le modèle de départ *EQUIP_0* qui consiste à spécifier des services fournis par la couche inférieure qui est dans ce cas la couche équipement. Ces services correspondent à des activités des équipements utiles pour les fonctions, qu'elles soient liées à un fonctionnement nominal ou à un dysfonctionnement. Ces services sont :

- soit sollicités explicitement par des fonctions comme par exemple les services relatives aux actionneurs ;
- soit sollicités implicitement car de tels services dépendent de phénomènes externes comme dans le cas de mesures de capteurs ou des surveillances d'équipement.

Ensuite, ce modèle de départ, constituant le modèle abstrait de notre modélisation, est raffiné afin de mettre en évidence des activités de fonctions concernées par les services ci-dessus. Parallèlement, un modèle de fonction *FUNCT_0* complète ce modèle raffinant *EQUIP_FCT_1* en spécifiant les activités caractéristiques des fonctions utiles pour la couche opération.

3.2.1 Modèle abstrait de départ

Caractéristiques statiques. La composante "Context" *EQUIP_C0* du modèle abstrait de départ décrit les paramètres statiques contenus dans le modèle abstrait de la couche *équipement* de l'architecture du système de contrôle du drone. Elle définit des ensembles d'objets abstraits et des relations entre certains éléments de ces ensembles, qui sont des caractéristiques statiques de la couche considérée.
Par exemple, on définit un ensemble abstrait *EQUIPMENT* composé des sous-ensembles disjoints *SENSOR* et *ACTUATOR*.
Les états fonctionnels distincts de l'ensemble énuméré *E_FLAG* sont caractérisés de la façon suivante :

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 suites à des défaillances ;
idle	état signifiant que l'équipement est en attente (par exemple, un actionneur attend 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.

Les états dysfonctionnels distincts de l'ensemble énuméré *E_STATUS* représentent les valeurs des mots de surveillance de chaque équipement :

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, des fonctions représentent les liens entre les éléments des ensembles ci-dessus. La fonction bijective *act_cmd* signifie que chaque actionneur est associé à un mot de commande.
La fonction bijective *eqt_surv* exprime que tous les équipements possèdent un mot de surveillance de leur état de santé.
La fonction bijective *eqt_mode* stipule que tous les équipements possèdent un mot associé à leur état de fonctionnement.
La fonction bijective *eqt_obs* exprime que tous les équipements possèdent un mot d'observation.
La fonction partielle *e_fault_class* permet d'assurer les principes d'identification et d'isolation d'une faute. Une faute est associée à un équipement. Certaines fautes peuvent affecter les couches supérieures de l'architecture du système.

La fonction totale *eqt_cat* indique que chaque équipement appartient à une catégorie d'équipements équivalents contenant soit l'équipement seul soit plusieurs équipements redondants.

Caractéristiques dynamiques. La composante "Machine" *EQUIP_0* de ce modèle décrit les caractéristiques dynamiques de la couche *équipement*.
On y décrit trois événements traduisant le comportement nominal des équipements et trois événements représentant des situations de défaillance.
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 ($value(eqt\_obs(sen)) := v$) 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 données, mais il peut être dans un état nominal ou erroné).
L'événement *a_activate* précise l'activation d'un actionneur dépendant des sollicitations d'une fonction représentées par une condition de commande. On a comme paramètres un actionneur *act*. L'actionneur est activé à condition qu'il soit préalablement dans un état d'attente ($flag(eqt\_mode(act)) = idle$), que la commande correspondante à cet actionneur soit disponible avec une valeur correcte ($e\_cmd(act\_cmd(act)) = TRUE$) et que l'actionneur soit dans un état nominal ou erroné ($e\_status(eqt\_surv(act)) \neq lost$).
L'événement *a_return* indique le retour d'un actionneur *act* à un régime permanent du point de vue de l'automatique ($flag(eqt\_mode(act)) := idle$) 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 ($flag(eqt\_mode(act)) = active$), que cette commande associée à cet actionneur ne soit plus valide ($e\_cmd(act\_cmd(act)) = FALSE$) et que l'actionneur soit dans un état nominal ou erroné ($e\_status(eqt\_surv(act)) \neq lost$).
Les événements suivants de diagnostic et de reconfiguration d'une faute sont indépendants de sollicitations d'une quelconque fonction. 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 ou défaillance *fault*. La faute est détectée et l'équipement passe dans l'état erroné à condition qu'il soit préalablement dans un état actif nominal ($flag(eqt\_mode(eqt)) = active$ et $e\_status(eqt\_surv(eqt)) = ok$) et que la faute fasse partie des fautes impactant cet équipement ($fault \mapsto eqt \in e\_fault\_class$).
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 ou défaillance *fault*. La faute est détectée et l'équipement passe dans l'état perdu à condition qu'il soit préalablement dans un état actif nominal ou erroné ($flag(eqt\_mode(eqt)) = active$ et $e\_status(eqt\_surv(eqt)) \neq lost$) et que la faute fasse partie des fautes impactant cet équipement ($fault \mapsto eqt \in e\_fault\_class$).
L'événement *recover* constitue l'étape de reconfiguration suite à une faute. On a comme paramètres un équipement *eq* et un équipement redondant associé *eq_red*. L'événement est déclenché à condition que l'équipement concerné soit dans un état perdu ($flag(eqt\_mode(eq)) = active$ et $e\_status(eqt\_surv(eq)) = lost$), que l'équipement redondant distinct soit inactive ($flag(eqt\_mode(eq\_red)) = spare$). Les actions engendrées concernent une activation de

l'équipement redondant et une mise hors service de l'équipement perdu
($flag := flag \lhd\!\!+ \{eqt\_mode(eq\_red) \mapsto active, eqt\_mode(eq) \mapsto off\}$) et une diminution du nombre d'équipements redondants restant ($nb(eqt\_cat(eq)) := nb(eqt\_cat(eq)) - 1$).

Propriétés. Les principales propriétés structurantes de notre modèle sont de deux ordres. D'une part, on trouve des propriétés relatives aux caractéristiques statiques au travers des axiomes définis dans la composante "Context" du modèle. Ce sont principalement des axiomes de liaison entre éléments d'ensembles abstraits. Par exemple, les axiomes suivants :

axm12 : $act\_cmd \in ACTUATOR \rightarrowtail\!\!\!\!\!\rightarrow COMMAND$

axm13 : $eqt\_surv \in EQUIPMENT \rightarrowtail\!\!\!\!\!\rightarrow E\_SURVEILLANCE$

axm18 : $e\_fault\_class \in DEFAULT \nrightarrow EQUIPMENT$

définissent des propriétés garantissant un mot de commande pour chaque actionneur modélisé et un mot de surveillance pour chaque équipement modélisé, ainsi qu'une propriété d'identification de classes de fautes associées aux équipements.
D'autre part, certaines propriétés sont étroitement liées à la dynamique du système. Elles sont décrites par des invariants au niveau de la composante "Machine" du modèle. Les propriétés importantes relatives aux caractéristiques dynamiques concernent la quantification (*nb*) des équipements redondants ou équivalents (sous-ensemble *SS_EQUIPMENT*) en fonction des états de fonctionnement de ces équipements :

inv6 : $\forall sse \cdot ((sse \in SS\_EQUIPMENT \land nb(sse) > 1) \Leftrightarrow (sse \in SS\_EQUIPMENT \land (\exists a, b \cdot (a \in sse \land b \in sse \land a \neq b \land flag(eqt\_mode(b)) = spare \land (flag(eqt\_mode(a)) = active \lor flag(eqt\_mode(a)) = idle)))))$

inv7 : $\forall sse \cdot ((sse \in SS\_EQUIPMENT \land nb(sse) = 1) \Leftrightarrow (sse \in SS\_EQUIPMENT \land (\exists a \cdot (a \in sse \land flag(eqt\_mode(a)) \neq off) \land (\forall b \cdot ((b \in sse \land a \neq b) \Rightarrow (b \in sse \land flag(eqt\_mode(b)) = off))))))$

inv8 : $\forall sse \cdot ((sse \in SS\_EQUIPMENT \land nb(sse) = 0) \Leftrightarrow (sse \in SS\_EQUIPMENT \land (\forall a \cdot (a \in sse \land flag(eqt\_mode(a)) = off))))$

3.2.2 Modèle de raffinement

Caractéristiques dynamiques. La machine *EQUIP_FCT_1* raffinant ce modèle d'équipements abstrait spécifie les relations ou les échanges entre les couches équipement et fonction. Cette machine complète les événements abstraits par de nouveaux événements précisant les mises à jour des données véhiculées entre ces couches. On y trouve par exemple des événements décrivant la mise en disponibilité (valeur de commande à TRUE) et la fin (valeur de commande à FALSE) de commandes vers les actionneurs.

Cet événement *send_cmd_t* (voir ci-après) décrit l'affectation de la valeur TRUE à la commande d'un actionneur par une fonction *fct* donnée.
On décrit également dans ce raffinement des événements de détection et de reconfiguration dans le cas où l'on a un équipement déclaré *ok* alors que les valeurs retournées sont erronées.

Propriétés. Parmi les invariants de ce modèle, la principale propriété relative aux relations inter couches concerne la confirmation des valeurs erronées envoyées par un équipement dans l'état *erroneous*.

Event $send\_cmd\_t \mathrel{\widehat{=}}$

any

fct

where

grd1 : $fct \in SL\_FUNCT$

grd2 : $f\_flag(fct\_mode(fct)) = f\_executing$

grd3 : $e\_cmd(fct\_cmd(fct)) = FALSE$

then

act1 : $e\_cmd(fct\_cmd(fct)) := TRUE$

end

inv9 : $\forall eq \cdot ((eq \in EQUIPMENT \wedge flag(eqt\_mode(eq)) = active \wedge$
$e\_status(eqt\_surv(eq)) = erroneous) \Rightarrow (f\_eval\_e(value(eqt\_obs(eq))) = FALSE))$

Cet invariant *inv9* exprime le fait que si un équipement actif est erroné alors l'évaluation des observations de cet équipement indique obligatoirement que les valeurs retournées sont fausses.

3.2.3 Modèle de fonctions abstrait

Parallèlement, on définit un modèle abstrait de comportement des fonctions *FUNCT_0* dans la couche fonction en mettant en évidence les étapes de l'activité d'une fonction : activation par une requête, acquisition de ressources et exécution. De même, on y décrit le comportement anormal des fonctions à l'aide d'événements de détection et de reconfiguration (*f_recover* ci-après).

La modélisation de ce comportement est assez similaire à celle réalisée pour le comportement abstrait des équipements.

3.3 Synthèse et vérification

La figure 3 récapitule la constitution actuelle des couches Équipement et Fonction en termes d'événements. Le premier modèle de fonctions ainsi réalisé est issu de la composition (non encore validée sous Rodin) des modèles *FUNCT_0* et *EQUIP_FCT_1*.

Vérification. Notre modélisation est réalisée avec la plateforme Rodin qui est un support de la méthode Event-B comme indiqué par Abrial et al. (2006). 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. Dans notre modèle, certaines preuves concernant les règles de faisabilité d'une action d'initialisation ne sont pas déchargées automatiquement en raison de l'indéterminisme associé à certaines initialisations introduisant des cas où l'initialisation de fonctions peut conduire à des fonctions vides. Pour la décharge de ces preuves non passées, on a la possibilité soit de modifier le modèle pour corriger d'éventuelles erreurs d'écriture,

Event $f\_recover \mathrel{\widehat{=}}$

any

fct

fct_eq

where

grd1 : $fct \in L\_FUNCT$

grd2 : $fct\_eq \in L\_FUNCT$

grd3 : $fct \neq fct\_eq$

grd4 : $f\_status(fct\_surv(fct)) = f\_lost$

grd5 : $f\_status(fct\_surv(fct\_eq)) = f\_ok$

grd6 : $f\_nb(fct\_cat(fct)) > 1$

grd7 : $fct\_eq \in fct\_cat(fct)$

then

act1 : $f\_nb(fct\_cat(fct)) := f\_nb(fct\_cat(fct)) - 1$

end

soit de réaliser interactivement les preuves à l'aide des prouveurs intégrés, soit de démontrer manuellement la satisfaction de ces règles non validées.
La satisfaction de l'ensemble de ces règles d'obligation de preuves permet d'assurer la correction du modèle par rapport aux connaissances a priori du système ou par rapport aux spécifications établies au début de la conception du système.
Actuellement, on a réussi à valider la totalité des règles d'obligation de preuves du modèle abstrait des équipements. Comme indiqué ci-dessus, les erreurs rencontrées portaient surtout sur une écriture incorrecte de certains invariants ou axiomes. Les actions correctrices ont consisté à modifier ces invariants ou axiomes, et à ajouter de nouveaux axiomes permettant de mieux préciser les caractéristiques des ensembles utilisés. Par exemple, on a défini des capteurs et actionneurs, qui ont ensuite été rangés dans des catégories d'équipements redondants bien identifiées. De plus, on a défini un théorème de "non blocage" garantissant l'activation en permanence d'un événement parmi les événements définis. Ce théorème a également été validé pour ce modèle abstrait de départ.

4 Conclusion et perspectives

Dans le cadre de la modélisation d'un drone capable d'autonomie, l'étude en cours consiste à spécifier en Event-B une architecture pour un système de contrôle intégrant des contraintes de sécurité. La structuration du système en couches fonctionnelles est l'approche couramment rencontrée dans la littérature (voir Alami et al. (1998)) vis-à-vis des niveaux d'autonomie décrits par Fargeon et Quin (1993). Dans le cas de certains satellites, l'autonomie est associée à des critères de fiabilité et de sécurité qui imposent de renforcer l'architecture fonctionnelle en introduisant dans les couches des composants permettant de réaliser des mécanismes de FDIR

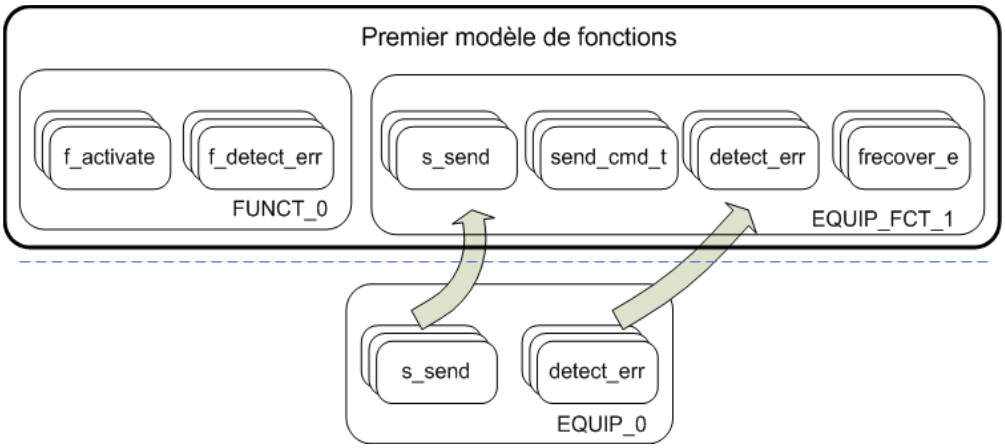

FIG. 3 – *Modélisation de la couche Fonction.*

comme indiqué par Lemai et al. (2006). Pour notre cas d'étude, on s'est inspiré de l'architecture proposée par Lemai et al. (2006), puis on a étendu ce concept en identifiant des propriétés entre les couches pour mieux satisfaire des objectifs qualitatifs de sécurité tels que « une panne simple ne conduit pas à la perte totale du système ». De plus, on a également associé des modes de défaillances à chaque composant de notre architecture.
Toutefois, l'apport de nos travaux se situe principalement dans la modélisation formelle de l'architecture retenue. La modélisation en B d'une architecture en couches de Troubitsyna et Laibinis (2004), et la modélisation en B du concept de FDIR de Sabatier et al. (2008) se sont focalisées plus particulièrement sur la spécification des applications logicielles et moins sur la définition du système au regard des contraintes de sécurité imposées. Par exemple, le nombre d'équipements redondants peut être utilisé sans mettre en évidence son impact sur la qualité de l'architecture du point de vue de la fiabilité et de la sécurité comme dans la modélisation de Sabatier et al. (2008). D'autres modélisations formelles permettent de prendre en compte conjointement les aspects matériels et logiciels d'un système embarqué relatifs aux critères de fiabilité et de sécurité. D'après Owre et al. (1995), PVS 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. De même, le langage formel AltaRica est de plus en plus utilisé dans le monde industriel pour l'évaluation de la sécurité de systèmes complexes (voir Arnold et al. (2000) et Seguin et al. (2007)). On a entrepris une première modélisation de notre architecture en AltaRica afin de tirer profit de ses outils d'analyse de la propagation de pannes (voir Chaudemar et al. (2009)). Ce type de modèle permet d'évaluer aisément une architecture donnée du point de vue de la sûreté de fonctionnement. Par contre, il ne permet pas de raisonner sur des contraintes génériques comme un nombre fixé de redondance. On a alors décidé de compléter cette modélisation. Cependant, notre modélisation de l'architecture du système de contrôle du drone en Event-B présenté dans cet article est encore incomplète et n'est que partiellement vérifiée. Le choix d'une telle modélisation se justifie par la possibilité offerte par Event-B d'exprimer explicitement des propriétés générales sur le système modélisé et de vérifier ces propriétés

tout au long du processus de modélisation en Event-B. Ainsi, l'expression d'une propriété de redondance sur des équipements a un impact sur la configuration des modes opérations. Par exemple, le système disposera de modes dégradés que s'il existe des équipements redondants ou des fonctions équivalentes, ou bien la défaillance d'une fonction de commande non reconfigurable inhibe l'activité de l'actionneur associé. Notre objectif est d'aboutir à une modélisation "correcte" par construction, qui vérifie des propriétés de sécurité clairement identifiées.
La généralisation de notre modélisation est rendue possible par la généricité de la modélisation Event-B. Par exemple, le nombre de composants dans une couche est facilement modifiable. De même, les fonctionnalités associées à des composants peuvent s'appliquer aussi bien à des drones qu'à des systèmes autonomes de satellites. Mais la validation est une étape importante à passer en raffinant suffisamment nos modèles pour parvenir à une implémentation testable sur un simulateur ou sur notre drone. La combinaison de cette modélisation en Event-B avec une modélisation en AltaRica est également un autre point sur lequel on doit continuer à travailler pour renforcer la confiance sur la sûreté de fonctionnement de notre système de contrôle.

Références

Abrial, J.-R. (2006). Refinement, Decomposition and Instantiation of Discrete Models. *Fundamentae Informatica 77*, 2006.

Abrial, J.-R. (2010). *Modeling in Event-B : System and Software Engineering*. Cambridge University Press.

Abrial, J.-R., M. Butler, S. Hallerstede, et L. Voisin (2006). An Open Extensible Tool Environment for Event-B. In Z. Liu et J. He (Eds.), *Formal Methods and Software Engineering*, Volume LNCS 4260/2006, pp. 588–605. Springer Berlin / Heidelberg.

Alami, R., R. Chatila, S. Fleury, M. Ghallab, et F. Ingrand (1998). An Architecture for Autonomy. *International Journal of Robotics Research 17*, 315–337.

Allen, J. (1983). Maintaining Knowledge about Temporal Intervals. In *Communications of the ACM*, Volume 26, pp. 832–843.

Arnold, A., A. Griffault, G. Point, et A. Rauzy (2000). The Altarica formalism for describing concurrent systems.

Bestougeff, H. et G. Ligozat (1989). *Outils logiques pour le traitement du temps: de la linguistique à l'intelligence artificielle*. Masson.

Butler, M. (2009). Incremental Design of Distributed Systems with Event-B.

Chaudemar, J.-C., E. Bensana, C. Castel, et C. Seguin (2009). Altarica and Event-B Models for Operational Safety Analysis: Unmanned Aerial Vehicle Case Study. In *Workshop on Integration of Model-based Formal Methods and Tools*. Düsseldorf, DE.

de Rosnay, J. (1977). *Le macroscope: Vers une vision globale*. Seuil.

Fargeon, C. et J.-P. Quin (1993). *Robotique mobile*. Teknea.

Lemai, S., M.-C. Charmeau, et X. Olive (2006). 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.

Owre, S., J. Rushby, N. Shankar, et F. von Henke (1995). Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering 21*, 107–125.

Rauzy, A. (2002). Mode automata and their compilation into fault trees.

Sabatier, D., B. Dellandrea, et D. Chemouil (2008). FDIR Strategy Validation with the B Method. In *The International Space System Engineering Conference - DASIA*.

Seguin, C., S. Humbert, J.-M. Bosc, C. Castel, P. Darfeuil, Y. Dutuit, et E. Focone (2007). 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*.

Troubitsyna, E. et L. Laibinis (2004). Fault Tolerance in a Layered Architecture: a General Specification Pattern in B.

Summary

This paper aims at describing safety architectures of autonomous systems by using Event-B formal method. The Event-B formalism well supports the rigorous design of this kind of systems. Its refinement mechanism allows a progressive modelling by checking the correctness and the relevance of the models by discharging proof obligations. The application of the Event-B method within the framework of layered architecture specification enables the emergence of desired global properties with relation to layer interactions. The safety objectives are derived in each layer and they involve static and dynamic properties such as an independence property, a redundant property. In our modelling, we distinguish nominal behaviour and abnormal behaviour in order to well establish failure propagation in our architecture.