La validation dans les premières étapes du processus de développement

Imen Sayar, Jeanine Souquières

LORIA – CNRS UMR 7503 – Université de Lorraine Campus Scientifique
BP 239, 54506 Vandœuvre lès Nancy cedex, France
{Firstname.Lastname}@loria.fr

RÉSUMÉ. *L'amélioration de la qualité d'un système commence par l'expression de ses besoins en langage naturel. Notre objectif est de prendre en compte la validation dès la compréhension des exigences et tout au long du développement de la spécification en Event-B. Pour combler l'écart entre le cahier des charges et sa spécification formelle, nous explicitons les interactions entre ces deux mondes. La validation est étudiée pour les modèles formels relativement aux exigences. La vérification permet de détecter des incohérences et des contradictions dans le cahier des charges et dans la spécification Event-B. La place des outils disponibles, notamment avec la plateforme Rodin, est importante tout au long du développement, améliorant sa qualité et sa documentation. Rodin et le plugin ProR permettent de gérer la trace des besoins en lien avec la spécification en cours de construction. L'ensemble des documents disponibles et les retours des outils de validation et vérification sont disponibles tout au long du développement. Notre approche est illustrée par l'étude de cas d'un système de contrôle du train d'atterrissage d'un avion.*

ABSTRACT. *Improving the quality of a system begins by the requirements elicitation. Our goal is to take into account the validation since the understanding of the requirements and all along the development of their Event-B specification. Our challenge is to bridge the gap between requirements, those of the client, and the specification, that of the computer scientist. We make explicit the interactions between the requirements and the specification under construction. The validation is studied for formal models with regard to the requirements. The verification may detect incoherences and contradictions in both requirements and formal specification. The Rodin platform tools are important all along the development to improve the quality and the documentation of the system. Rodin and the ProR plugin allow us to manage the trace of the requirements and their specification. The documentation and the feedback of the different used tools for the validation and verification are available at any time of the development. We illustrate our approach to the case study of an aircraft landing system.*

MOTS-CLÉS : *besoins, spécification, raffinement, validation, vérification, outils.*

KEYWORDS: *requirements, specification, refinement, validation, verification, tools.*

DOI:10.3166/ISI.22.4.11-41

1. Introduction

Le travail de recherche sur l'ingénierie des exigences présenté dans (van Lamsweerde, 2008) insiste sur deux activités à résoudre : l'analyse du domaine et la modélisation des exigences. Le groupe Standish a conduit des études par l'interview d'entreprises dans le domaine du logiciel. Il a récemment publié une dernière version du Rapport CHAOS dont le premier date de 1994[1] ; l'une des principales causes des difficultés dans le développement de systèmes réside dans la prise en compte des exigences. Celles-ci sont souvent très pauvres, voire inexistantes.

Depuis 1998, Abrial (Abrial, 1998) propose des bases pour la rédaction de la spécification. Un premier *texte explicatif* se consacre à l'exposé des besoins. Il s'intéresse au pourquoi ou à sa compréhension. Ce document peut être oublié lorsqu'il n'apporte plus de nouvelles informations. Le deuxième texte appelé *texte de référence* du futur système expose les connaissances nécessaires à la réalisation du futur système. Ces deux textes peuvent contenir des redondances.

Le processus de développement de systèmes à l'aide du raffinement utilisé dans Event-B (Abrial, 2010) est comparable au processus de la cascade. Le modèle initial précise son invariant que le système doit garantir et chaque raffinement est à nouveau prouvé par son invariant. Cette approche assume les propriétés suivantes :

– les exigences sont explicitées pour décrire le modèle initial ;

– le modèle initial est une description formelle répondant aux exigences de sécurité et fonctionnelles et ;

– les décisions prises lors d'un raffinement sont mémorisées en liaison avec les exigences associées.

En réalité, peu de développements décrivent entièrement ces propriétés. Les exigences évoluent avec le développement, celles-ci ne sont pas forcément exprimées dans le modèle initial, de nombreux raffinements introduisent de nouvelles informations dans le modèle (Abrial, 2006).

Les activités de validation et de vérification sont utilisées dans les premières étapes du développement depuis les exigences jusqu'à la spécification formelle. Parmi les techniques de validation de modèles, leur exécution est plus attrayante, particulièrement dans le cadre de la modélisation à base de modèles Event-B. La plus grande difficulté vient du non-déterminisme de la plupart des modèles raffinés. En fait, il est recommandé de réduire l'abstraction et le non-déterminisme pas-à-pas. Tandis que des outils actuels, tels que ProB (Leuschel *et al.*, 2003, 2008), peuvent animer des modèles avec un non-déterminisme modéré, ils ne peuvent pas toujours traiter l'ensemble des problèmes soulevés. Des stratégies d'exploration exhaustives tombent rapidement dans l'explosion combinatoire.

1. Rapport CHAOS du Standish Group (http ://www.standishgroup.com).

Dans notre approche, nous mémorisons le cahier des charges, nous prenons en compte ses différentes mises à jour et nous explicitons sa place dans le processus de développement (Abrial, 2009). Pour cela, un ensemble de liens, ou relations, entre exigences et spécifications a été identifié et réalisé avec la plateforme Rodin (Abrial *et al.*, 2010) et le plugin ProR (Jastram, 2010). Ces liens sont mis à jour tout au long du processus du développement, depuis le cahier des charges jusqu'à sa spécification terminée. Différentes actions entre l'évolution de ces deux « mondes » sont présentées (Sayar *et al.*, 2016). Le choix d'un ou plusieurs besoins nous amène à la spécification en cours de développement. Depuis la spécification, les actions modifiant les exigences permettent :

– d'ajouter des termes formels dans une exigence,

– d'ajouter, supprimer et mettre à jour une exigence,

– de valider et vérifier la spécification vis-à-vis des exigences à l'aide des outils disponibles. Ces outils permettent de découvrir des incohérences et des imprécisions aussi bien dans la spécification en cours que dans les besoins.

La spécification est en cours de développement et décrite pas à pas. L'ensemble des outils disponibles, tels que les outils de validation et de vérification, sont utilisés tout au long de ce développement, et non seulement lorsque la spécification est terminée. Dans ce papier, nous nous intéressons plus particulièrement à l'activité de validation en tant que processus rigoureux. Cette activité démarre avec la structuration des besoins, avant que la spécification associée ne soit introduite, jusqu'à l'animation des modèles Event-B. Ces modèles sont validés relativement aux besoins du client, en vue de détecter des problèmes.

Dans la suite de cet article, la partie 2 présente brièvement la méthode Event-B et la plateforme Rodin. Nous utilisons les outils suivants : ProR pour gérer les besoins informels en lien avec la spécification en Event-B, les outils de vérification, de validation et Event-B Statemachine. La partie 3 présente notre approche en termes du cahier des charges et de sa spécification en cours de développement. La partie 4 concerne la préparation de la validation dès la première étape du processus de développement, celle de l'expression de ses exigences en langage naturel. La partie 5 concerne l'application de notre approche à une étude de cas. Nous présentons quelques étapes de développement depuis les besoins jusqu'à la spécification en Event-B. La validation est abordée dans la partie 6 en vue de prouver la correction de la spécification relativement aux besoins à tout moment du développement. La partie 7 aborde la détection d'incohérences dans les exigences en liaison avec la vérification. Les travaux connexes sont présentés dans la partie 8. Une conclusion est proposée dans la partie 9 avec nos travaux futurs.

Notre approche est illustrée par l'étude de cas d'un système de contrôle du train d'atterrissage d'un avion (Boniol *et al.*, 2014). Son architecture est présentée dans la figure 1. Il s'agit d'un système hybride combinant :

– une partie mécanique et hydraulique actionnée par des électrovannes et observée par des capteurs. Elle contient trois ensembles d'atterrissage,

– une partie digitale incluant le logiciel de commandes et,

– l'interface de contrôle pour le pilote.

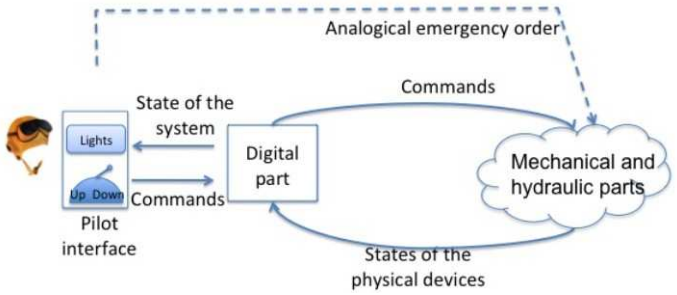


Figure 1. Architecture globale

2. Cadre formel et outils support

2.1. La méthode Event-B

Event-B est une méthode formelle pour spécifier et modéliser des systèmes complexes à partir de la notion de machines abstraites et du raffinement (Abrial, 2010). Une spécification est décomposée en deux parties :

– le contexte pour décrire la partie statique du modèle à l'aide des ensembles, des constantes, des théorèmes et des axiomes ;

– la machine pour décrire la partie dynamique à l'aide des variables et des événements.

Event-B permet de modéliser un système et son environnement à travers la notion d'observation des événements. Le système est décrit à l'aide des trois axes suivants permettant de définir la correction des différents modèles : le modèle final est une implantation correcte du modèle initial lorsque les obligations de preuve ont été démontrées.

Description du système. Il est modélisé par un état ou une fonction associant des noms à des valeurs, et contraint par un invariant qui circonscrit l'ensemble des valeurs licites que peut prendre cet état. L'invariant est une formule logique du premier ordre portant sur les valeurs des variables et des constantes. Un événement est une substitution gardée sur l'état. La garde est un prédicat du premier ordre sur l'état. Une machine et son contexte en Event-B sont présentés dans la figure

Sémantique. Elle est liée à la notion de correction. Le modèle doit être réalisable, l'ensemble des états licites n'est pas vide et les événements relient des états licites. L'invariant doit être préservé lorsqu'un événement est déclenché depuis un état licite. Le raffinement maintient l'invariant abstrait : il existe une fonction d'abstraction

reliant l'état du modèle concret au modèle abstrait, et chaque événement concret maintient l'invariant abstrait. Les propriétés de chaque modèle ainsi que celles des raffinements sont données par un ensemble d'obligations de preuve, appelées OP.

CONTEXT *Landing_Ctx*
SETS
Positions
Doors_Position
CONSTANTS
g_extended
g_retracted
open
closed
ext_seq
ret_seq
no_seq
rev_seq
AXIOMS
Pos−axm: *partition* (*Positions* , {*g_extended*}, {*g_retracted*})
Doors_pos−axm: *partition(Doors_ Position*, {*open*}, {*closed*})
rev_seq_typ : *rev_seq* = {*ret_seq* ↦ *ext_seq*, *ext_seq* ↦ *ret_seq*, *no_seq* ↦ *no_seq*}
END

MACHINE *Landing_System*
SEES *Landing_Ctx*
VARIABLES
gears_pos
gears_moving
doors_pos
INVARIANTS
inv1 : *gears_pos* ∈ *Positions*
inv2 : *doors_pos* ∈ *Doors_Position*
inv3 : *gears_moving* ∈ *BOO*
EVENTS
INITIALISATION
Event extend_gears
Event retract_gears
Event reverse
Event open_doors
Event close_doors
...
END

Figure 2. Exemple d'une spécification en Event-B de l'étude de cas choisie

Développement. Le développement de spécifications Event-B s'effectue par une approche générale progressive. Il s'agit du raffinement ou relation qui lie deux modèles par l'enrichissement de l'un par l'autre. Un raffinement peut être vu comme une ou plusieurs opérations élémentaires :

– l'extension de l'état. Elle introduit de nouvelles variables et constantes sans relation avec l'état abstrait ;

– le renforcement de l'invariant. Il restreint l'espace et se rapproche de celui du système final ;

– la réification de variables appelée raffinement de données ;

– la décomposition d'un événement. Elle correspond à l'introduction d'un autre niveau d'observation du système.

La correction des raffinements est définie par les OP qui garantissent que les invariants de la machine précédente sont préservés par le raffinement.

2.2. La plateforme Rodin

Rodin[2] est une plateforme autour d'Event-B développée avec Eclipse. Elle est étendue à l'aide de *plugins* et fournit un soutien pour le raffinement et la preuve mathématique. Elle permet d'éditer, animer, prouver et contre-prouver les modèles.

ProR. C'est un *plugin* de Rodin pour exprimer une structure hiérarchique des exigences (Jastram, 2010). L'approche proposée commence par l'élicitation des exigences initiales et les hypothèses du domaine. Elle ne propose pas de notation particulière mais un classement des artefacts, voir figure 3.

ID	*Description*
FUN-G	The landing system goal is maneuvering gears and their associated doors
FUN-G-1	Maneuvering gears consists on extending or retracting them and reversing their movement
FUN-G-2	Maneuvering doors consists on opening or closing them
...	...

Figure 3. Exemple d'exigences avec ProR

Cette approche est basée sur le modèle de référence WRSPM (Gunter *et al.*, 2000). Elle crée manuellement les liens entre exigences et éléments du modèle Event-B en cours de construction, ces liens pouvant être annotés. Pour gérer la traçabilité entre exigences et modèles formels, ProR permet de :

– définir des liens depuis la spécification Event-B vers les exigences texte sous ProR. Initialement, ces exigences sont informelles,

– insérer des éléments formels dans les exigences avec ProR, ces éléments étant issus de la spécification Event-B.

La vérification. L'objectif est de formaliser les propriétés du système. Ces propriétés de nature diverses sont abstraites et générales au début du développement puis elles se précisent au fur et à mesure de l'évolution de la spécification. La correction des propriétés est prouvée via des preuves au sens mathématiques : le modèle est correct, sans erreur et bien conçu ; ses propriétés sont explicitées. La preuve est pratiquée tout au long du développement (Abrial, 2003). Les obligations de preuve (OP), sont générées et déchargées automatiquement ou semi-automatiquement par les outils de preuve sous Rodin.

La validation. Cette activité est utilisée tout au long du processus formel de développement avec l'outil ProB (Leuschel *et al.*, 2003 ; Bendisposto *et al.*, 2008 et Leuschel *et al.*, 2008) sous Rodin. Parmi les plugins, des outils complémentaires à la preuve sont indispensables pour la validation des modèles Event-B. Le premier

2. http://wiki.event-b.org

concerne un animateur et un *model-checker* permettant de détecter un ensemble de problèmes comme l'interblocage ou des comportements non autorisés. Le deuxième outil est un contre-prouveur qui aide à la preuve interactive avec ProB pour trouver des contre-exemples, *via* un accès direct aux obligations de preuve. Le troisième concerne l'outil proposé par Yang (Yang *et al.*, 2011) permettant d'écrire automatiquement la propriété d'absence de blocage du modèle formel. Il évite d'utiliser l'animation pour laquelle l'espace d'états correspondant à ce modèle peut être exhaustif.

3. Présentation de notre approche

Notre approche fait intervenir deux mondes différents, celui du semi-formel pour décrire le cahier des charges, et le formel pour présenter la spécification. Nous décrivons un système informatique à l'aide de trois composants :

– *CdC.* Le cahier des charges est présenté à l'aide de l'outil ProR.

– *Spec.* Sa spécification formelle est définie en Event-B.

– *Liens.* Les liens entre le *CdC* et la *Spec* sont réalisés à partir des termes formels introduits dans la spécification en cours de définition. Ces termes formels sont communs aux deux.

Le système est défini par une suite d'étapes de développement. Une étape consiste à passer d'un état à un autre, suite à un choix dans le développement. Ce choix concerne :

– la prise en compte de besoins exprimés dans le *CdC* existant,

– la définition de termes formels dans la spécification *Spec* en cours de développement,

– un *lien* existant entre le *CdC* et sa *Spec* et/ou,

– l'utilisation d'un *patron de développement*.

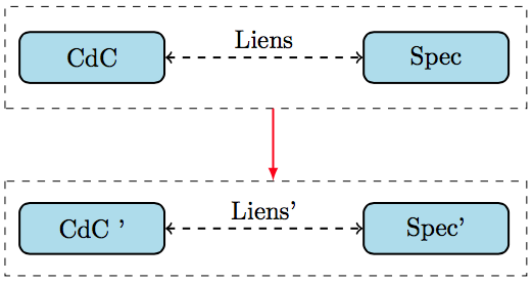


Figure 4. Présentation d'une étape

L'état résultant présenté dans la figure 4 est défini par ses trois composants :

– *CdC'*. Il est obtenu par introduction de termes formels issus de la spécification en cours de développement et des liens existants entre *Spec* et *CdC*.

– *Spec'*. Sa spécification formelle en Event-B évolue via le raffinement, des modifications ou corrections.

– *Liens'*. Les liens sont automatiquement mis à jour à partir des nouveaux termes formels introduits et communs au *CdC'* et à la *Spec'*.

L'évolution de l'ensemble est mise à jour en permanence. Le *CdC* n'est pas oublié, au contraire, il est renforcé et affiné grâce aux liens entre le *CdC* et sa *Spec*. Les différents outils existants sont disponibles en permanence.

3.1. Le CdC

Le cahier des charges, informel au démarrage du développement, est réécrit sous la forme de phrases courtes et étiquetées. Sa trace est sauvegardée et permet de suivre son évolution tout au long du développement. Le *CdC* est défini par une suite de phrases hiérarchisées et typées. Au fur et à mesure de son évolution, les termes issus du monde formel, celui de la spécification, sont introduits dans le *CdC*. Donc, une phrase peut contenir des termes formels intégrés dans le texte informel. Un terme peut avoir un type. Ces termes proviennent de la spécification formelle associée au *CdC*. Ses opérations sont celles des types utilisés, suites, ensembles et produit cartésien noté CP.

CdC is Sequence(*Sentence*)

Sentence is CP [ID: *TEXT*,
[Order: *INTEGER*,]
Description: *Set*(*Term*),
[*Sequence*(*Sentence*),]
[*Set*(*Term_Type*)]]

Term is Informal_Term | *Formal_Term*

Term_Type is Fact | *Functionality* | *Behavior* | *Obligation*

La figure 5 présente un exemple du *CdC* de l'étude de cas proposée dans ce papier. La phrase *FUN-G-1* est un fils de la phrase *FUN-G*. Sa description est un texte informel et son type est *Functionality*.

3.2. La Spec

La spécification est définie en Event-B en utilisant le raffinement, représentant la formalisation des exigences du *CdC*. La correspondance entre le *CdC* et sa *Spec* est définie par les paramètres suivants (figure 5) :

Requirements		
ID	*Description*	*Term_Type*
FUN-G	The landing system goal is maneuvering gears and their associated doors	Functionality Fact
FUN-G-1	Maneuvering gears consists on extending or retracting them and reversing their movement	Functionality
FUN-G-2	Maneuvering doors consists on opening or closing them	Functionality
...	...	
FUN-2-doors	The doors must be open when extending or retracting gears	Obligation
FUN-2-4	In nominal mode, the landing sequence is : open doors --> extend gears --> close doors	Behavior

Figure 5. Description partielle du CdC

– *ID*. Il s'agit d'un commentaire ;

– *Order*. L'ordre entre les événements dans la spécification est décrit par leurs gardes ;

– *Informal_Term*. Il s'agit d'un commentaire ;

– *Formal_Term*. Un terme formel introduit dans le *CdC* correspond à un élément d'Event-B ; il s'agit du nom d'une variable, d'une constante, d'un ensemble, d'un événement ou d'un modèle formel ;

– *Term_Type*. Le type d'un terme est :

- *Functionality* pour des événements,
- *Fact* pour des constantes, des ensembles et des variables,
- *Obligation* pour des axiomes, invariants, théorèmes et gardes,
- *Behavior* pour l'animation et la simulation de machines de la spécification.

3.3. Les liens

La liaison entre le *CdC* et sa *Spec* s'effectue par l'introduction de *termes formels*. Ceux-ci sont initialement introduits dans la spécification formelle et propagés automatiquement dans le *CdC* :

$$CdC \leftarrow \textit{terme formel} \rightarrow Spec$$

Ces liens sont automatiquement mis à jour et disponibles tout au long du développement. Initialement, nous n'avons pas de spécification formelle et il n'y a pas de terme formel dans le *CdC*. Au cours du développement, les termes formels introduits dans la *Spec* sont intégrés dans le *CdC*. Les liens, gérés par l'outil ProR, concernent :

– les termes formels issus de la *Spec* et introduits dans le *CdC*. Ils sont dénotés entre [ ] ;

– chaque phrase du *CdC*. Elle a des liens vers ses éléments formels dans la *Spec* ;

– les éléments du glossaire. Ils expriment la correspondance entre chaque terme formel et sa définition informelle introduite dans le *CdC*.

Illustrons ces liens sur l'étude de cas :

– initialement, la *Spec* est vide et la figure 5 correspondante ne contient pas de terme formel. Il en résulte que le *CdC* n'a pas de lien avec la *Spec* et le glossaire est vide ;

– regardons la figure 11 présentée dans la partie 5.3. Au cours du développement, la Spec contient les deux termes formels [Landing_System] et [gears_pos] introduits dans les phrases FUN-G et FUN-G-1 issus de la Spec. Ces deux phrases du CdC ont des liens avec ces deux éléments formels de la Spec. Le glossaire contient les deux couples (Landing_System, landing system) et (gears_pos, gears).

3.4. Les patrons de développement

Dans notre approche, nous utilisons des techniques existantes, telles que le raffinement et la réutilisation. La lecture du texte de référence du *CdC* présentée par (Abrial, 2009) nous a invités à étudier la notion de patrons pour réutiliser une solution dans le développement. Un patron avec ses paramètres permet de décrire un sous-problème identifié et sa solution en réutilisant des connaissances acquises par l'expérience. La présentation des besoins dans l'étude de cas de l'hémodialyse (Mashkoor, 2016) ont souvent la même forme. Elle nous a conduit à définir plusieurs patrons ou modèles génériques adaptés à notre approche dans laquelle nous manipulons des triplets <*CdC, Liens, Spec*>. Un patron générique est mémorisé avec ses paramètres. La définition du patron utilise le raffinement en Event-B, les preuves correspondantes et précise les éléments restant à décrire.

Le patron proposé dans (Sayar *et al.*, 2017) génère automatiquement une partie de la spécification en cours de construction. La notion de collage est présentée *via* le *CdC* et la *Spec*. L'effort de vérification est réduit. Les patrons utilisés dans cette étude de cas peuvent être utilisés et adaptés à d'autres études de cas, comme celle utilisée dans ce papier.

3.5. Les outils de vérification et de validation

Dans notre approche, les outils de vérification et de validation jouent un rôle important et s'intègrent complètement dans le processus du développement. La validation est prise en compte dès la gestion des exigences, c'est-à-dire avant la formalisation *via* la spécification et au fur et à mesure de la construction du modèle

formel (Abrial, 2010). Cette activité s'effectue simultanément avec la preuve du modèle Event-B et la complète. La vérification nécessite un texte formel en cours de développement et prouve que le logiciel décrit possède bien les propriétés attendues et tout au long du développement.

La vérification. Elle répond à la question « avons-nous construit le système correctement ? ».

Dans notre approche, avec Event-B, la vérification est statique, sans exécution et concerne aussi bien le monde formel que le monde informel, en vue d'améliorer la qualité du logiciel. Les retours de cette activité peuvent apporter des indications sur les lacunes du *CdC*, par exemple avec des contradictions ou bien des oublis. Nous nous intéressons particulièrement au cas où la preuve ne réussit pas et l'énoncé à prouver n'est probablement ni favorable ni réfutable. Le modèle défini est à revoir : il n'est pas forcément erroné mais il est plutôt probablement trop pauvre et correspond à un *CdC* incomplet, peu précis.

La validation. Elle répond à la question « avons-nous construit le bon système ? ». Ce système doit faire ce dont l'utilisateur a besoin.

Dans notre approche, la validation commence dès le traitement des exigences. Elle commence par une étape de préparation en extrayant et mémorisant, depuis le *CdC*, des éléments typés : données, fonctionnalités, comportements et obligations. Ces informations sont mises à jour par des termes formels tout au long du développement. La validation s'effectue au fur et à mesure du développement du modèle Event-B par la preuve de propriétés. Il s'agit de raisonner en termes de propriétés auxquelles le futur système doit obéir. A nouveau, il s'agit de définir le quoi et le pourquoi, et non le comment. L'outil ProB (Leuschel *et al.*, 2003 ; Leuschel *et al.*, 2008) permet l'animation et le model-checking des spécifications.

4. Préparation à la validation

Nous abordons la validation tout au long du développement. Un premier travail commence par la compréhension des besoins, leur structuration et leur analyse. La figure 6 présente des informations nécessaires à la validation extraites de chaque phrase du *CdC*.

ID	*Term_Type*	*Terms*	*Event-B Model*
...	...	...	...

Figure 6. Eléments de validation

ID. La structuration du *CdC* a pour objectif l'obtention d'un document lisible et accessible par toutes ses parties prenantes. Pour cela, les besoins sont décomposés en

phrases courtes et accessibles *via* un identifiant en ProR, appelé *ID*. L'objectif est d'améliorer le texte de référence (Abrial, 2010 ; Su *et al.*, 2011).

Term_Type. La validation est prise en compte dès la structuration du *CdC*. Elle concerne les futurs modèles formels en Event-B par rapport aux exigences du client. Les phrases du *CdC* sont typées, *Term_Type*, et s'expriment sous forme de :

– données du futur système appelées *Fact*,

– fonctionnalités attendues, appelées *Functionality*,

– conditions appelées *Obligation* avec lesquelles le système fonctionne sous forme de pré-conditions et post-conditions (Hoare, 1969),

– comportements appelés *Behavior*. Un comportement est défini à l'aide des fonctions existantes.

Terms. En étudiant chaque besoin à l'aide des types décrits ci-dessus, notre objectif est de préciser les éléments destinés à la validation du futur modèle formel. Ces éléments typés extraits du *CdC* concernent :

– les données présentes dans le futur système,

– les fonctionnalités attendues,

– les conditions de fonctionnement du système,

– les comportements décrits en termes de scénarios de validation. Ces scénarios permettent de décrire les différents états du modèle Event-B en animant ses événements. Ceux-ci apparaissent dans le *CdC* structuré sous forme de phrases décrivant un enchaînement d'actions.

Pour disposer du maximum d'informations disponibles dans le *CdC*, nous mémorisons chaque élément, présenté dans la colonne *Terms* de la figure 6. Cet élément est utilisé dans le besoin en fonction de son type, comme indiqué dans la colonne *Term_Type*.

Event-B Model. Un lien entre le besoin et le nom du modèle Event-B correspondant est introduit, via des machines et des contextes. Il est présenté dans la colonne *Event-B Model* de la figure 6.

5. Application de notre approche

L'étude de cas concerne le développement d'un système hybride contrôlant le train d'atterrissage d'un avion. Le point de départ est le cahier des charges présenté dans l'article (Boniol *et al.*, 2014). Il décrit le système, sous forme d'un texte explicatif, en termes de son architecture, voir la figure 1, du comportement de l'équipement hydraulique et de la description du logiciel. Le dernier chapitre de cet article présente quelques exigences et propriétés ; il s'agit d'un texte de référence.

5.1. Initialisation du développement

La figure 7 présente l'initialisation de l'état de son développement avec ses besoins structurés et typés.

Figure 7. Première étape dans le développement du système

L'état du système est décrit par :

– *CdC.* Il est réécrit sous la forme de phrases courtes et étiquetées, comme présenté dans la figure 5. Chaque phrase est typée et peut utiliser plusieurs types.

Exemple. La phrase *FUN-G* décrit deux types de termes, *Functionality* et *Fact*. La phrase *FUN-2-doors* décrit une *Obligation* et la phrase *FUN-2-4* décrit un *Behavior*.

– *Spec.* La spécification est vide.

– *Liens.* Il n'y a pas de terme formel. Il n'y a pas de lien car aucun terme formel n'est introduit et le glossaire est vide.

ID	*Term_Type*	*Terms*	*Event-B Model*
FUN-G	Functionality	- maneuvering	
	Fact	- gears - doors	
FUN-G-1	Functionality	- extend gears - retract gears - reverse gears	
FUN-G-2	Functionality	- open doors - close doors	
...	...	...	
FUN-2-doors	Obligation	*pre-condition :* - doors must be open *post-condition :* - extend gears - retract gears	
FUN-2-4	Behavior	open doors → extend gears → close doors	

Figure 8. Prise en compte de la validation à l'initialisation du système

Exemple. Les informations nécessaires à la validation sont résumées dans la figure 8. Les types introduits dans la figure 5 sont associés aux éléments informels de la phrase. Par exemple, l'*Obligation* de la phrase *FUN-2-doors* se précise : sa pré-condition concerne *doors must be open* et sa post-condition concerne les termes *extend gears* et *retract gears*.

5.2. Initialisation de la spécification

De manière générale, la prochaine étape est décrite dans la figure 9.

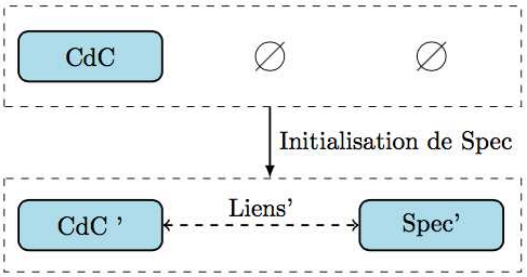


Figure 9. Nouvelle étape dans le développement

L'état de départ du développement contient les besoins structurés et typés, comme présenté dans la figure 5. Notre choix est de prendre en compte le besoin informel *FUN-G*. Pour cela, la spécification est initialisée par l'introduction de la machine *Landing_System* et son contexte *Landing_Ctx*.

L'état résultant présenté dans la figure 10 est résumé par :

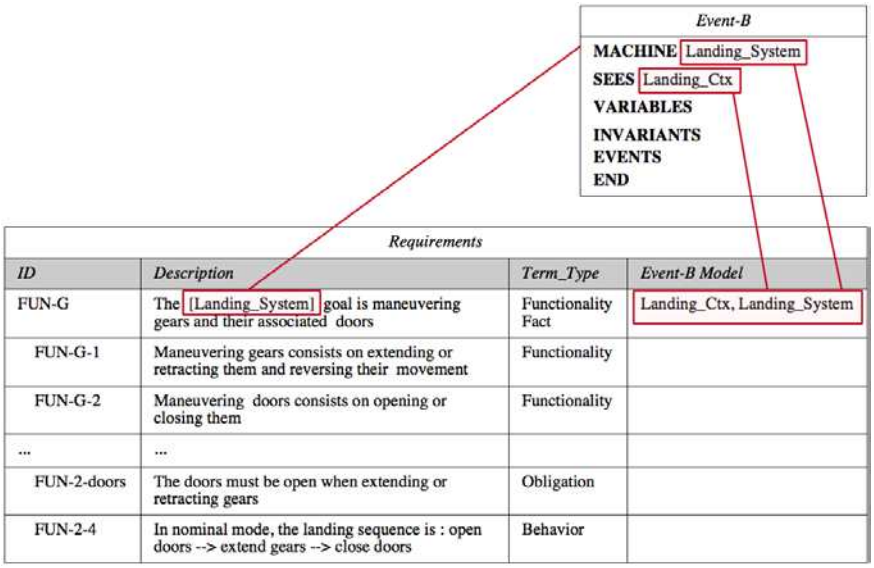
– *CdC'*. Le besoin *FUN-G* est partiellement pris en compte. Le terme formel *[Lan- ding_System]* remplace le terme informel correspondant, *landing system*.

– *Spec'*. La spécification est initialisée par la machine *Landing_System* et son contexte *Landing_Ctx* lié à l'utilisation d'Event-B.

– *Liens'*. Ils sont automatiquement mis à jour. Le terme formel *Landing_System* issu de la spécification Event-B est introduit dans le *CdC* et remplace le terme informel *landing system*. Le glossaire est initialisé, voir tableau 1.

La colonne *Event-B Model* introduite dans la figure 8 est mise à jour par les deux nouveaux noms de modèles introduits, comme présenté dans la dernière colonne de la figure 10.

REMARQUE.– Le contexte *Landing_Ctx* introduit dans la spécification Event-B n'est pas présent dans les besoins. C'est un élément technique lié au langage utilisé.



Requirements

ID	*Description*	*Term_Type*	*Event-B Model*
FUN-G	The [Landing_System] goal is maneuvering gears and their associated doors	Functionality Fact	Landing_Ctx, Landing_System
FUN-G-1	Maneuvering gears consists on extending or retracting them and reversing their movement	Functionality	
FUN-G-2	Maneuvering doors consists on opening or closing them	Functionality	
...	...		
FUN-2-doors	The doors must be open when extending or retracting gears	Obligation	
FUN-2-4	In nominal mode, the landing sequence is : open doors --> extend gears --> close doors	Behavior	

Figure 10. Introduction de la machine et son contexte

Tableau 1. Initialisation du glossaire

Formal_Term	*Informal_Term*
Landing_System	landing system

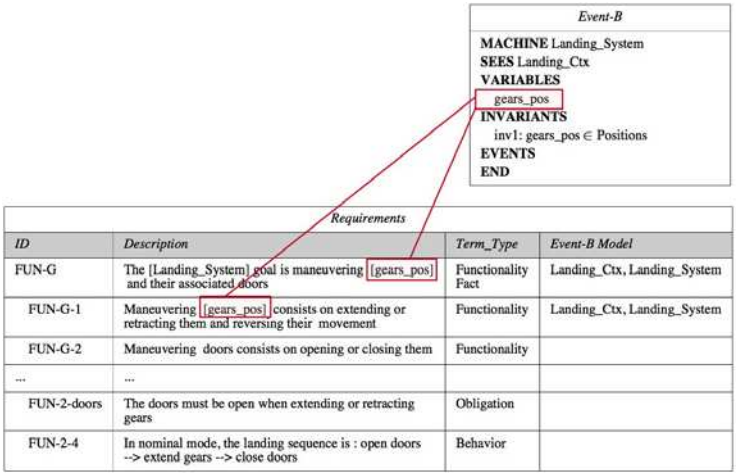
5.3. Introduction d'une donnée

En partant de l'état obtenu, nous prenons en compte les deux besoins informels *FUN-G* et *FUN-G-1* par introduction de la variable *gears_pos* dans la machine *Landing_System*. L'état résultant présenté dans la figure 11 est résumé par :

– *CdC'*. L'introduction de la variable *gears_pos* se répercute dans la description des besoins. Le terme informel *gears* utilisé dans les deux besoins *FUN-G* et *FUN-G-1* est remplacé par le terme formel *[gears_pos]* issu de la spécification de la machine *Landing_System*.

– *Spec'*. La variable *gears_pos* et son type sont introduits dans la machine *Landing_System*.

– *Liens'*. Les différents liens sont automatiquement mis à jour. Le glossaire est mis à jour par le couple (*gears_pos*, *gears*).

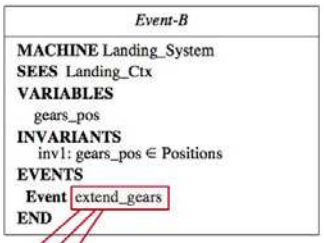


Requirements

ID	*Description*	*Term_Type*	*Event-B Model*
FUN-G	The [Landing_System] goal is maneuvering [gears_pos] and their associated doors	Functionality Fact	Landing_Ctx, Landing_System
FUN-G-1	Maneuvering [gears_pos] consists on extending or retracting them and reversing their movement	Functionality	Landing_Ctx, Landing_System
FUN-G-2	Maneuvering doors consists on opening or closing them	Functionality	
...	...		
FUN-2-doors	The doors must be open when extending or retracting gears	Obligation	
FUN-2-4	In nominal mode, the landing sequence is : open doors --> extend gears --> close doors	Behavior	

Figure 11. Introduction d'une donnée

5.4. Introduction d'une fonctionnalité



Requirements

ID	*Description*	*Term_Type*	*Event-B Model*
FUN-G	The [Landing_System] goal is maneuvering [gears_pos] and their associated doors	Functionality Fact	Landing_Ctx, Landing_System
FUN-G-1	Maneuvering [gears_pos] consists on [extend_gears] or retracting them and reversing their movement	Functionality	Landing_Ctx, Landing_System
FUN-G-2	Maneuvering doors consists on opening or closing them	Functionality	
...	...		
FUN-2-doors	The doors must be open when [extend_gears] or retracting gears	Obligation	Landing_Ctx, Landing_System
FUN-2-4	In nominal mode, the landing sequence is : open doors --> [extend_gears] --> close doors	Behavior	Landing_Ctx, Landing_System

Figure 12. Introduction de la fonctionnalité extend_gears

Prenons en compte l'introduction de la fonctionnalité *extend_gears* dans la machine *Landing_System*. Elle répond aux besoins *FUN-G-1*, *FUN-2-doors* et *FUN-2-4*.

L'état résultant présenté dans la figure 12 est résumé par :

– *CdC'*. L'élément formel *extend_gears* introduit dans la spécification est automatiquement introduit dans le *CdC*, dans les trois besoins *FUN-G-1*, *FUN-2-doors* et *FUN-2-4*. Ces trois besoins sont des fils du besoin *FUN-G*. Le terme *extend_gears* correspond aux termes informels *extending* et *extend gears* du *CdC*.

– *Spec'*. L'événement *extend_gears* est introduit dans la machine *Landing_System*.

– *Liens'*. Ils sont automatiquement mis à jour.

Le glossaire mis à jour est présenté dans le tableau 2 dans lequel le terme formel *extend_gears* désigne deux termes informels, *extending* et *extend gears* utilisés dans les besoins *FUN-2-doors* et *FUN-2-4*. La colonne *Event-B Model* est mise à jour automatiquement pour les trois besoins traités.

Tableau 2. Glossaire mis à jour

Formal_Term	*Informal_Term*
Landing_System	landing system
gears_pos	gears
extend_gears	extending
extend_gears	extend gears

5.5. Introduction d'une condition

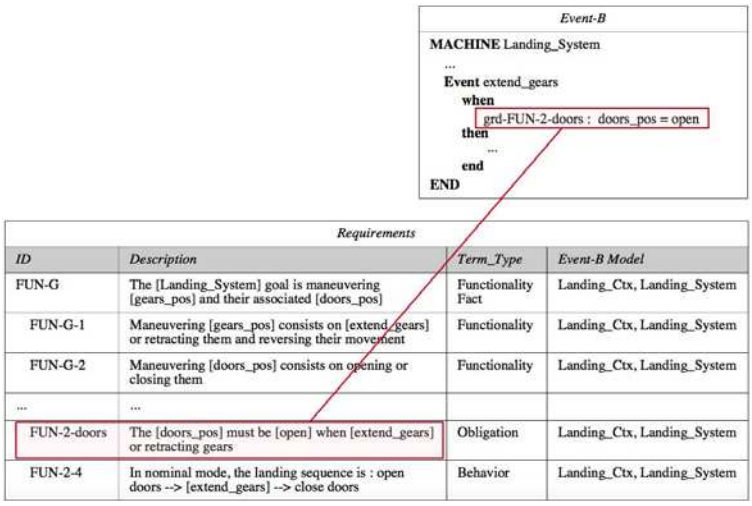


Figure 13. Introduction d'une condition dans un événement existant

Le développement a évolué depuis l'état présenté dans la figure 12 avec introduction de la variable *doors_pos*. Guidées par le besoin *FUN-2-doors*, nous définissons les conditions de fonctionnement du système pour la fonction *extend_gears*.

L'état résultant présenté dans la figure 13 est résumé par :

– *CdC'*. Le besoin *FUN-2-doors* est mis à jour par l'introduction du terme formel *[open]* dans les besoins. Celui-ci remplace le terme informel *open*, valeur de la variable *doors_pos*.

– *Spec'*. La garde de l'événement *extend_gears* est introduite. Elle concerne la va- riable *doors_pos* :

$$doors\_pos = open$$

– *Liens'*. Ils sont mis à jour automatiquement.

La figure 14 présente l'évolution de la figure 8 par les nouveaux termes formels. Par exemple, la colonne *Terms* contient un terme informel, *maneuvering* dans la phrase FUN-G du *CdC*. Les deux autres termes de cette phrase sont des termes formels introduits par la spécification, à savoir les termes formels *[gears_pos]* et *[doors_pos]* introduits respectivement dans les parties 5.3 et 5.5.

ID	*Term_Type*	*Terms*	*Event-B Model*
FUN-G	Functionality	- maneuvering	Landing_System Landing_Ctx
	Fact	[gears_pos] [doors_pos]	
FUN-G-1	Functionality	[extend_gears] - retract gears - reverse gears	Landing_System Landing_Ctx
FUN-G-2	Functionality	- open doors - close doors	Landing_System Landing_Ctx
FUN-2-doors	Obligation	*pre-condition :* [doors_pos] must be [open] *post-condition :* [extend_gears] - retract gears	Landing_System Landing_Ctx
FUN-2-4	Behavior	open doors → [extend_gears] → close doors	Landing_System Landing_Ctx

Figure 14. Prise en compte de la validation

6. Validation

La validation est utilisée tout au long du développement de la spécification. La machine *Landing_System* est présentée dans la figure 15.

Cette machine est validée relativement aux différents types de termes introduits, *Fact*, *Functionality*, *Obligation* et *Behavior* présentés dans la figure 16. Tous les termes introduits sont formels sauf le terme *maneuvering* pour l'identificateur *FUN-G* de la figure.

MACHINE *Landing_System*
SEES *Landing_Ctx*
VARIABLES
gears_pos
doors_pos
gears_moving
INVARIANTS
inv1: *gears_pos* ∈ *Positions*
inv2: *doors_pos* ∈ *Doors_Position*
inv3: *gears_moving* ∈ *BOOL*
EVENTS
INITIALISATION
BEGIN
act1 : *gears_pos* := *g_extented*
act2 : *doors_pos* := *closed*
END
Event extend_gears
WHEN
grd1: *gears_pos* = *g_retracted*
grd_FUN–2–doors: *doors_pos* = *open*
THEN
FUN G 1: *gears_pos* := *g_extended*
END
Event retract_gears
WHEN
grd1: *gears_pos* = *g_retracted*
grd_FUN–2–doors: *doors_pos* = *open*
THEN
FUN–G–1: *gears_pos* := *g_ retracted*
END
Event reverse
THEN
...
END
Event open_doors
WHEN
grd1: *doors_pos* = *closed*
THEN
FUN–G–2: *doors_pos* := *open*
END
Event closes_doors

.....

END

Figure 15. Machine Landing_System

6.1. Validation relativement aux données, fonctionnalités et obligations

Pour l'étude de cas, ces différents cas sont :

– les deux variables *gears_pos* et *doors_pos* correspondent à des éléments de type *Fact* ;

– les éléments de type *Functionality* sont des événements Event-B, sauf *maneuvering* qui n'a pas été modélisé par notre choix de stratégie de développement ;

– l'élément de type *Obligation* a été modélisé par la garde nommée *grd_FUN-2-doors* présenté dans l'événement *extend_gears* dans la figure 13.

6.2. Validation relativement au comportement

Les scénarios de validation pour l'animation décrivent les comportements du futur système. Ces scénarios ne sont pas toujours suffisants pour couvrir tous les comportements possibles.

ID	*Term_Type*	*Terms*	*Event-B Model*
FUN-G	Functionality	- maneuvering	Landing_System Landing_Ctx
	Fact	[gears_pos] [doors_pos]	
FUN-G-1	Functionality	[extend_gears] [retract_gears] [reverse]	Landing_System Landing_Ctx
FUN-G-2	Functionality	[open_doors] [close_ doors]	Landing_System Landing_Ctx
...	...	...	
FUN-2-doors	Obligation	*pre-condition :* [doors_pos] must be [open] *post-condition :* [extend_gears] [retract_gears]	Landing_System Landing_Ctx
FUN-2-4	Behavior	[open_doors] → [extend_ gears] → [close_doors]	Landing_System Landing_Ctx

Figure 16. Prise en compte de la validation pour la machine Landing_System

6.2.1. Simulation d'un scénario existant

Pour animer le modèle *Landing_System* par rapport au scénario décrit dans le besoin *FUN-2-4* et présenté dans la figure 16, nous simulons la séquence d'événements Event-B suivante, avec l'outil ProB :

open_doors → extend_gears → close_doors

avec l'état des variables suivantes :

gears_pos = g_retracted

doors_pos = closed

6.2.2. Définition de nouveaux scénarios avec le client

Les scénarios existants dans le *CdC* sont insuffisants. De nouveaux scénarios peuvent être proposés.

6.2.2.1. Animation à partir d'une machine à états

La validation est relative au *CdC*. Nous utilisons un *plugin* de la plateforme Rodin, Event-B Statemachines[3]. Une machine à états appelée *SM* est associée à chaque modèle Event-B :

– chaque état de la *SM* représente une valeur des variables Event-B et

– chaque transition entre deux états représente un événement.

Nous prenons un état initial dans lequel les portes sont fermées et le train d'atterrissage est étendu.

3. http ://wiki.event-b.org/index.php/Event-B_Statemachines

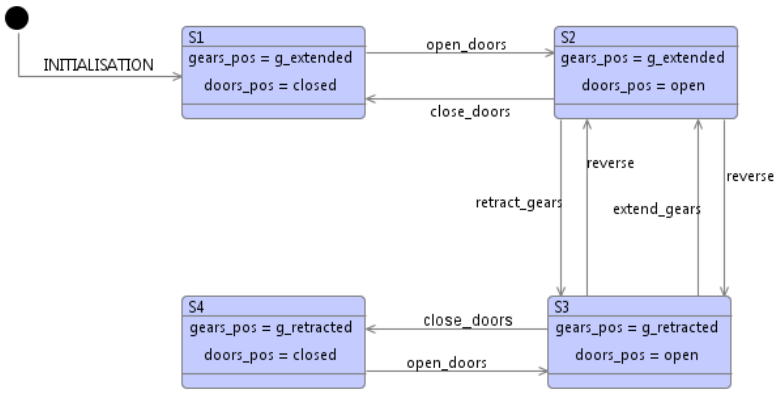


Figure 17. Une sous-machine à états de Landing_System

La figure 17 présente la *SM* correspondante à la machine Event-B *Landing_System*. Cette *SM* décrit la séquence de rétraction et d'extension des trains. Ses états sont décrits par les valeurs de ses variables *gears_pos* et *doors_pos*. Le passage de l'état *S1* à l'état *S2* s'effectue par l'événement *open_doors* ; la variable *doors_pos* passe de *closed* à *open*.

A partir de cette *SM*, de nouveaux scénarios sont proposés dans lesquels des enchaînements d'états ne sont pas autorisés :

– *INITIALISATION* → *retract_gears*

Ce scénario passe de *S1* à *S3*. Il est interdit car il n'est pas possible de rétracter les trains sans ouvrir les portes ;

– *INITIALISATION* → *close_doors*

Il n'y a pas de passage immédiat entre un état de trains étendus (état *S1*) à un autre un état de trains pliés (état *S4*) sans passer par la séquence de rétraction. Ou bien, il n'est pas possible de fermer les portes car elles sont déjà fermées ;

– *INITIALISATION* → *open_doors* → *retract_gears* → *retract_gears* → *close_doors*

Il n'est pas possible de rétracter les trains plus d'une fois dans cette séquence.

Ces scénarios ne sont pas permis par la machine *Landing_System* lors de son animation avec ProB.

6.2.2.2. Construction de scénarios non autorisés

A partir de la compréhension des exigences, nous introduisons une action décrivant un comportement non souhaité dans un scénario décrivant un comportement existant. A partir du scénario de validation décrit dans la figure 16 :

open_doors → extend_gears → close_doors,

nous introduisons à nouveau l'action *open_doors* avant que la porte soit fermée. L'objectif de ce scénario est de vérifier si l'ouverture des portes des trains d'atterrissage se réalise une deuxième fois dans une séquence d'extension des trains. Ce scénario a la forme suivante :

open _doors → extend_gears → ***open_doors*** *→ close_doors*

La machine *Landing_System* a interdit l'animation de ce scénario, à l'aide de ProB. La garde de l'événement *open_doors* interdit sa simulation lorsque les portes sont déjà ouvertes.

6.3. Validation des modèles raffinés

Nous abordons la problématique de l'espace d'états en termes de l'explosion combinatoire. La mémorisation de l'historique et de la trace des modèles validés aide à réduire l'effort de validation des modèles complexes dans différents raffinements.

Exemple. A partir de la figure 13, nous prenons en compte l'introduction :

– des cylindres appartenant à la partie hydro-mécanique du système hybride et,

– de la manette et des voyants lumineux présentés dans l'interface pilote.

Les besoins associés sont présentés dans la figure 18. Le besoin *FUN-2-4-Cyln* est un enfant du besoin *FUN-2-4* décrit après évolution de la figure 13. En l'état de la compréhension des besoins, nous ne pouvons pas préciser la hiérarchie des deux besoins *FUN-2-P-I* et *FUN-2-P-light* relativement au besoin *FUN-2-4*.

6.3.1. Prise en compte du besoin FUN-2-4-Cyln

Ce besoin de type comportement décrit en détail le déroulement interne de la séquence d'extension des trains présenté dans le besoin *FUN-2-4* en termes des cylindres. Nous avons :

– le besoin *FUN-2-4*. Il a introduit l'événement *[extend_gears]* dans le premier modèle abstrait *Landing_System*. Ce modèle a été validé dans des étapes précédentes relativement au comportement décrit dans la section 6.2 ;

– le besoin *FUN-2-4-Cyln*. Il s'agit d'un comportement décrit par la suite d'éléments suivante, certains sont formels et certains sont informels :

[open_doors] → unlock up gears cylinders → move down gears cylinders →
lock down gears cylinders → [close_doors]

ID	*Description*	*Term_Type*	*Event-B Model*
FUN-2-4	In nominal mode, the landing sequence is : [open_doors] → [extend_gears] → [close_doors]	Behavior	Landing_System Landing_Ctx
FUN-2-4-Cyln	Using cylinders movement, the extending sequence is : [open_doors] → unlock up gears cylinders → move down gears cylinders → lock down gears cylinders → [close_doors]	Behavior	
FUN-2-P-I	The pilot interface is composed of green, orange and red lights and an up/down handle	Fact	
FUN-2-P-light	When gears cylinders are locked down, the green light is on	Obligation Behavior	

Figure 18. Besoins relatifs aux cylindres et à l'interface pilote

L'état résultant du traitement du besoin *FUN-2-4-Cyln* est résumé par :

– *Spec'*. Le modèle *Landing_System_Hybrid* spécifie le besoin *FUN-2-4-Cyln*. Il est défini par un raffinement du modèle abstrait *Landing_System* comme présenté dans la figure 19. Le raffinement de l'événement *extend_gears* est présenté dans la figure 20.

– *CdC'*. Les termes formels venant de *Spec'* sont introduits dans le *CdC'*

– *Liens'*. Ils sont mis à jour automatiquement.

ID	*Description*
FUN-2-4-Cyln	Using cylinders movement, the extending sequence is : *[open_doors]→[unlock_up_gears_cylinders]→[move_ down_gears_cylinders] → [lock_down_gears_cylinders]→ [close_doors]*

Le comportement abstrait du besoin *FUN-2-4* a été validé dans le paragraphe 6.2.1. La figure 20, créée avec le plugin Flows[4] sous Rodin, présente son raffinement en termes des trois événements *[unlock_up_gears_cylinders]*, *[move_down_gears_cylinders]* et *[lock_down_gears_cylinders]*.

6.3.2. Communication entre les différents composants

Regardons un nouveau raffinement dans lequel la communication s'exprime par la notion de collage :

4. http ://wiki.event-b.org/index.php/Flows

CONTEXT *Landing_Hybrid_Ctx* **SETS** *Movement* **CONSTANTS** *locked_up* *locked_down* *moving_up* *moving_down* *unlocked_up* *unlocked_down* **AXIOMS** Cyln−axm: *partition* (*Movement*, {*locked_up*}, {*locked_down*}, {*moving_up*}, {*moving_down*} {*unlocked_up*}, {*unlocked_down*}) **END**	**MACHINE** *Landing_System_Hybrid* **SEES** *Landing_Hybrid_Ctx* **REFINES** *Landing_System* **VARIABLES** *gears_pos* *gears_moving* *doors_pos* *gears_cylinders* **INVARIANTS** inv_cyln : *gears_cylinders* ∈ *Movement* **EVENTS** **INITIALISATION** // *Event extend_gears est defini* // *par les* 3 *Event−B suivants* *Event unlock_up_gears_cylinders* *Event move_down_gears_cylinders* *Event lock_down_gears_cylinders* *refines extend_gears* *Event retract_gears* *Event reverse* *Event open_doors* *Event close_doors* ... **END**

Figure 19. Machine Landing_System_Hybrid et son contexte

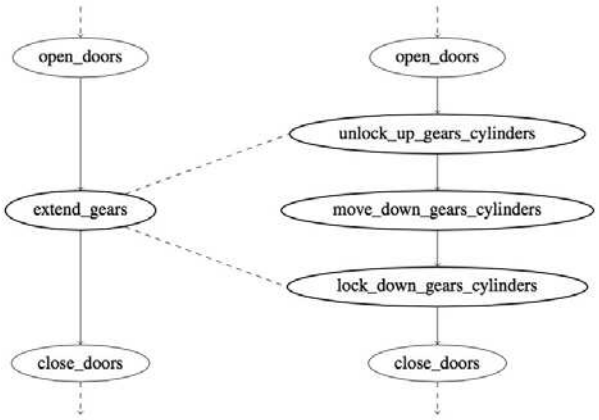


Figure 20. Raffinement de l'événement extend_gears

– *CdC'*. Nous prenons en compte les deux besoins *FUN-2-P-I* et *FUN-2-P-light*.

– *Spec'*. Nous introduisons :

- les deux variables *handle* et *light_state*,
- les constantes *green*, *on* et *down*,

La spécification contient les invariants de collage suivants :

INVARIANTS

glu_inv1 : $gears\_pos = g\_extended \Rightarrow gears\_cylinders = locked\_down \wedge doors\_pos = closed$

glu_inv2 : $light\_state(green) = on \Rightarrow gears\_cylinders = locked\_down \wedge handle = down$

– *Liens'*. Ils sont automatiquement mis à jour.

7. Vérification

Les outils de vérification de spécifications formelles, tels que les prouveurs et les générateurs d'obligations de preuves sous Rodin, permettent de :

– détecter des omissions dans le *CdC* concernant :

- l'état initial,
- certains besoins implicites et non décrits dans le *CdC*,
- l'absence de scénarios dans le *CdC*,

– détecter des contradictions entre la spécification mathématiquement correcte et les besoins. La spécification ne correspond pas aux besoins lorsque le *CdC* est trop pauvre.

La validation aide à déceler des problèmes liés à la vérification, par exemple en utilisant des contre-prouveurs, comme ProB, avec des contre-exemples relativement aux obligations de preuve non déchargées. L'activité de vérification permet de découvrir des incohérences dans les exigences. Celles-ci peuvent correspondre à des contradictions, des répétitions ou des oublis.

Exemple. Ajoutons la nouvelle exigence présentée dans la figure 21.

ID	*Description*	*Term_Type*
FUN-i	The gears can move only if doors are closed	Obligation

Figure 21. Introduction d'une nouvelle exigence

Elle est modélisée en Event-B par l'invariant :

$$FUN\text{-}i\_inv : \ gears\_moving = TRUE \Rightarrow doors\_pos = closed$$

dans lequel la variable booléenne *gears_moving* indique si les trains sont en déplacement.

Avant l'ajout de cet invariant, le modèle en Event-B était mathématiquement correct. Après l'introduction de cet invariant, les prouveurs de Rodin ont donné lieu à un ensemble d'obligations de preuve non déchargées. Une de ces obligations de preuve a précisé que l'événement *extend_gears* ne respecte pas ce nouvel invariant *FUN-i_inv* :

– une contradiction est décelée entre sa garde et cet invariant :

- garde : $doors\_pos = open$

- invariant : $gears\_moving = TRUE \Rightarrow doors\_pos = closed$,

– cette garde a été modélisée à partir l'exigence *FUN-2_doors* et cet invariant modélisé à partir l'exigence *FUN-i* du *CdC*. Une contradiction entre ces exigences est détectée à l'aide des prouveurs de Rodin.

8. Travaux connexes

Notre approche prend en compte la demande du client et ses besoins. Pour améliorer la qualité du système demandé, le triplet < *CdC, Liens, Spec* > est défini dans les premières étapes du développement. Chacun de ces trois éléments évolue en tenant en compte l'introduction de termes formels issus de la spécification : un terme formel remplace un terme informel dans le *CdC*. Nous renforçons l'existence du *CdC* et de son évolution. Elle est facilitée par l'utilisation de la plateforme Rodin qui propose :

– le langage Event-B. Un concept important est celui du raffinement qui permet de décrire la spécification pas à pas,

– un ensemble d'outils disponibles :

- gestion des besoins avec ProR,

- validation à l'aide d'un animateur, model-checker, contre-prouveur,

- vérification avec les prouveurs,

- visualisation des modèles définis à l'aide de Event-B Statemachines.

Les documents des exigences utilisés dans l'industrie sont souvent pauvres et difficiles à comprendre (Abrial, 2006). Dans (Su *et al.*, 2011), les auteurs recommandent de réécrire ce document sous la forme de deux textes différents. L'un est destiné à la compréhension du problème et l'autre contient les définitions et les besoins en termes de phrases courtes dans un langage naturel. Les systèmes sont de plus en plus hybrides (Su *et al.*, 2014). La notion de patrons est introduite dans (Hoang *et al.*, 2013) comme un modèle générique. Cette notion de patron, initialement utilisée dans une approche orientée objet (Gamma *et al.*, 1995), peut être vue comme un moyen élaboré de réutiliser des connaissances acquises par

l'expérience.

Notre approche commence par l'étape de re-structuration du document des exigences proposée par Abrial. Les autres étapes utilisent l'outil ProR (Jastram, 2010). Nous proposons une évolution de la structure des exigences avec la préparation de l'activité de la validation. De plus, nous ajoutons de nouveaux liens entre la spécifica- tion Event-B et les exigences.

Dans l'approche présentée dans (Heisel *et al.*, 1999), les exigences et la spécification sont clairement identifiées. Un guide méthodologique détaillé pour ces deux activités est proposé et n'introduit pas de nouveaux langages ou formalismes. Le processus d'élicitation des besoins est indépendant du langage de spécification utilisé. Un lien de traçabilité entre les exigences et la spécification est maintenu *via* les agendas qui expriment le sommaire de la méthode. Les agendas décrivent et guident le processus de développement, assurant une certaine garantie de la qualité du produit obtenu grâce aux conditions de validation associées aux différentes étapes. Notre approche en est une évolution récente liée à l'utilisation de la plateforme Rodin et l'outil ProR.

L'approche KAOS (van Lamsweerde, 2009) est orientée buts ; elle permet de les identifier et de les raffiner progressivement jusqu'à l'obtention des contraintes. La méthode associée propose de dériver les besoins en termes d'objets et d'actions. Un modèle multivue intègre tous les concepts du langage utilisé, pour articuler les exigences et les spécifications (van Lamsweerde, 2008). A partir de plusieurs applications industrielles décrites à l'aide de KAOS, la présentation de (Ponsard *et al.*, 2015) considère les interactions entre les artifacts des exigences ; des outils supports sont nécessaires et plusieurs sortes de diagrammes sont utilisées pour aider à comprendre les exigences. Dans notre approche, les exigences doivent être comprises avant de les structurer.

L'évolution des besoins est prise en compte (Hallerstede *et al.*, 2014). Cela signifie que les exigences doivent être fréquemment modifiées et incorporées incrémentalement dans la spécification formelle. Dans notre approche, nous mémorisons les liens entre exigences et spécifications tout au long du développement avec les actions qui les font évoluer.

Driss (2014) propose un modèle pivot basé sur une ontologie, pour faire le lien entre exigences et spécifications formelles. Dans notre approche, nous utilisons les liens offerts par ProR et la plateforme Rodin pour relier les exigences avec le modèle formel en cours de développement.

Les outils d'aide à la validation de modèles Event-B (Mashkoor *et al.*, 2016a, 2016b ; Jacquot, 2015) ont mis en évidence la possibilité de définir des sémantiques mathématiques qui garantissent la correction de modèles exécutables. Ils ont esquissé une extension de la notion de raffinement en tant qu'étape dans le processus de développement. Suite à ce travail, notre approche propose une meilleure intégration entre le formel et le semi-formel dans le processus de développement.

Dans l'approche décrite dans (Alkhammash *et al.*, 2015), des structures semi-formelles sont utilisées pour combler l'écart entre les exigences et les modèles Event-B. Cette approche conserve la trace entre les exigences et les modèles Event-B. Dans une première étape, les exigences sont classées et affectées à des composants Event-B. Dans une deuxième étape, des détails sont introduits graduellement dans les modèles formels. Une troisième étape est d'utiliser l'outil UML-B et l'outil de décomposition atomique pour générer des modèles Event-B. Une différence entre notre présentation et celle-ci concerne le fait que notre approche n'est pas seulement pour des modèles Event-B, mais elle est basée sur la combinaison des exigences évolutives alternativement avec les étapes de raffinement des modèles Event-B.

Dans (Hallerstede *et al.*, 2011), un algorithme en ProB a été décrit pour l'animation du raffinement de plusieurs niveaux simultanés. Cette animation permet de détecter une variété d'erreurs introduites avec le raffinement. Ces résultats sont empiriques ; la preuve et l'animation se complètent relativement à la validation. Nous utilisons cette proposition pour raisonner tout au long du développement en utilisant le raffinement.

9. Conclusion et perspectives

Dans cet article, nous avons pris en compte la validation dès les premières étapes du processus de développement, depuis la structuration des exigences jusqu'à la spécification en Event-B d'un logiciel. Mémoriser les exigences implique leur mise à jour et leur place dans ce processus. Nous avons utilisé la plateforme Rodin et illustré notre approche pour le développement de trois études de cas[5] :

– un système hybride de contrôle du train d'atterrissage d'un avion (Boniol *et al.*, 2014). Cette étude de cas est présentée dans ce papier,

– un système de l'hémodialyse, système critique, hybride et interactif contrôlant sa machine (Mashkoor, 2016) et,

– un système de contrôle hybride et interactif du distributeur automatique de billets.

A tout instant du développement, nous disposons d'un seul cahier des charges, celui qui évolue. Au début du processus, il désigne celui du client qui ne contient pas de termes formels. En effet, les termes formels sont issus de la spécification et celle-ci n'est pas encore initialisée. A toute étape du développement, le *CdC* contient son évolution avec les termes formels inclus, suite à leur introduction dans la spécification. La validation est prise en compte dès la première étape du processus de développement, celle de l'expression de ses exigences en langage naturel. Nous réécrivons ces exigences sous forme de phrases étiquetées. Nous ajoutons de nouveaux paramètres au document structuré avec ProR et décrivons des informations liées à la validation.

5. Leur développement est accessible à l'adresse http ://dedale.loria.fr

Dans les étapes suivantes, nous ajoutons des termes formels dans les exigences à partir de la spécification Event-B. La mise à jour des exigences est prise en compte. La préparation à la validation est guidée par les types des éléments formels ainsi que les modèles raffinés. L'animation et l'aide à la validation sont guidées par la définition de nouveaux scénarios dans lesquels des enchaînements d'actions ne sont pas autorisés ou non précisés dans les exigences. La vérification permet de détecter des incohérences dans les exigences.

Aujourd'hui, la compréhension et l'utilisation directe du cahier des charges, lorsqu'il existe, n'a pas encore atteint la maturité suffisante pour proposer une démarche de développement. L'évolution des outils disponibles, comme par exemple la plateforme Rodin permettant d'éditer, animer, prouver et contre-prouver les modèles, apporte un point de vue important pour combler l'écart entre ces deux mondes, celui des exigences et celui des spécifications formelles. Si nous regardons notre approche présentée dans la figure 4, le raffinement utilisé en Event-B pourrait être généralisé aux trois composants *CdC*, *Liens et Spec*. Une partie du développement pourrait être outillée, voire automatisée.

La notion de liens introduite dans ce papier est importante dans le processus de développement d'un logiciel et dans sa validation. A ce jour, ces liens sont extraits des différents documents disponibles, qu'ils soient formels ou informels. Ils sont utilisés manuellement dans le processus. Cette notion doit être clarifiée et rendue outillée automatiquement. Concernant la validation, un travail sur des outils manuels et semi-automatiques est nécessaire en lien avec les différentes catégories des exigences.

Les limites de notre approche concernent les points suivants :

– tout dépend du *CdC* disponible : plus le *CdC* est pauvre et plus l'apport de notre approche est minime ;

– un travail important et fastidieux doit être réalisé sur le *CdC*, que ce soit sa compréhension ou sa réécriture ;

– l'approche proposée est lourde à gérer et ;

– la phase de préparation à la validation n'est pas automatisable.

Bibliographie

Abrial J.-R. (1998). Le Cahier des Charges: Contenu, Forme et Analyse (en vue de la Formalisation), Technical report, Connsultant.

Abrial J.-R. (2003). B : passé, présent, futur. *Technique et Science Informatiques*, vol. 22, n° 1, p. 89-118.

Abrial J.-R. (2006). Formal Methods in Industry : Achievements, Problems, Future. *28th International Conference on Software Engineering, Shanghai*, L. J. Osterweil, H. D. Rombach, M. L. Soffa (Eds), China, ACM, p. 761-768.

Abrial J.-R. (2009). Faultless Systems: Yes We Can! *IEEE Computer*, vol. 42, n° 9, p. 30-36.

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

Abrial J.-R., Butler M. J., Hallerstede S., Hoang T. S., Mehta F., Voisin L. (2010). Rodin: An open toolset for modelling and reasoning in Event-B. *STTT*, vol. 12, n° 6, p. 447-466.

Alkhammash E., Butler M., Fathabadi A. S., Cirstea C. (2015). Building Traceable Event-B Models from Requirements. *Science of Computer Programming (Special Issue on Automated Verification of Critical Systems, AvoCS'13)*, vol. 111, Part 2, p. 318-338.

Bendisposto J., Leuschel M., Ligot O., Samia M. (2008). La validation de modèles Event-B avec le plugin ProB pour RODIN. *Technique et Science Informatiques*, vol. 27, n° 8, p. 1065-1084.

Boniol F., Wiels V. (2014). Landing Gear System case Study. *ABZ Case Study, Berlin, Communications in Computer and Information Science, Springer*, vol. 433, p. 1-18.

Driss S. (2014). From natural language specifications to formal specifications via an ontology as a pivot model, Theses, Université Paris Sud - Paris XI, June.

Gamma E., Helm R., Johnson R., Vlissides J. (1995). *Design Patterns : Elements of Reusable Object- oriented Software*, Addison-Wesley Longman Publishing Co., Inc., Boston, USA,

Gunter C. A., Gunter E. L., Jackson M., Zave P. (2000). A Reference Model for Requirements and Specifications - Extended Abstract. *Proceedings of the 4th International Conference on Requirements Engineering*, *IEEE Software*, p. 37-43.

Hallerstede S., Jastram M., Ladenberger L. (2014). A Method and Tool for Tracing Requirements into Specifications. *Sciences Computer Program*, vol. 82, p. 2-21.

Hallerstede S., Leuschel M., Plagge D. (2011). Validation of Formal Models by Refinement Animation. *Sci. Comput. Program.*, vol. 78, n° 3, p. 272-292.

Heisel M.. Souquieres J. (1999). A Method for Requirements Elicitation and Formal Specification. *Proc. 18th Int. Conference on Conceptual Modeling. ER'99*. n° 1728 in *LNCS* Springer-Verlag, p. 309-324.

Hoang T. S., Fürst A., Abrial J. (2013). Event-B Patterns and their Tool Support. *Software and System Modeling*, vol. 12, n° 2, p. 229-244.

Hoare C. A. R. (1969). An Axiomatic Basis for Computer Programming. *Commun. ACM*, vol. 12, n° 10, p. 576-580 & 583.

Jacquot J. (2015). Premières leçons sur la spécification d'un train d'atterrissage en B Événementiel. *Technique et Science Informatiques*, vol. 34, n° 5, p. 549-573.

Jastram M. (2010). ProR, an Open Source Platform for Requirements Engineering based RIF. *SEISCONF*.

Leuschel M., Butler M. (2008). ProB : an Automated Analysis Toolset for the B Method. *International Journal on Software Tools for Technology Transfer*, vol. 10, n° 2, p. 185-203.

Leuschel M., Butler M. J. (2003). ProB: A Model Checker for B. *FME 2003: Formal Methods, International Symposium of Formal Methods Europe*, Pisa, Italy, Proceedings, p. 855-874.

Mashkoor A. (2016). The Hemodialysis Machine Case Study. *Abstract State Machines, Alloy, B, TLA, VDM, and Z. 5th International Conference, ABZ 2016*, Linz, Austria, May 23-27, Proceedings, p. 329-343.

Mashkoor A., Jacquot J.-P. (2016a). Validation of Formal Specifications through Transformation and Animation. *Requirements Engineering*, Springer Verlag.

Mashkoor A., Yang F., Jacquot J.-P. (2016b). Refinement-based Validation of Event-B Specifications. *Software and Systems Modeling*, Springer Verlag.

Ponsard C., Darimont R., Michot A. (2015). Combining Models, Diagrams and Tables for Efficient Requirements Engineering: Lessons Learned from the Industry. *Actes du XXXIIIe Congrès INFORSID*, Biarritz, France, May 26-29, p. 235-250.

Sayar I., Souquières J. (2016). La validation dans le processus de développement. *34e Congrès INFORSID*, Grenoble, France.

Sayar I., Souquières J. (2017). Du cahier des charges à sa spécification. *16e journées AFADL*, 13 au 16 Juin 2017, Montpellier, France.

Su W., Abrial J.-R., Huang R., Zhu H. (2011). From Requirements to Development : Methodology and Example. *13th International Conference on Formal Engineering Methods*, Durham, UK, p. 437-455.

Su W., Abrial J.-R., Zhu H., (2014). Formalizing hybrid systems with Event-B and the Rodin Platform. *Sci. Comput. Program*, vol. 94, p. 164-202.

van Lamsweerde A. (2008). Requirements Engineering: from Craft to Discipline. *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Atlanta, Georgia, USA, p. 238-249.

van Lamsweerde A. (2009). *Requirements Engineering - From System Goals to UML Models to Software Specifications*, Wiley.

Yang F., Jacquot J. (2011). An Event-B Plug-in for Creating Deadlock-Freeness Theorems, *14th Brazilian Symposium on Formal Methods*, Brazilian Computer Society.