La Validation dans le Processus de Développement

Imen Sayar, Jeanine Souquières

La Validation dans le Processus de Développement

Imen Sayar — Jeanine Souquières

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

RÉSUMÉ. *L'amélioration de la qualité d'un logiciel commence par l'expression de ses exigences en langage naturel. Notre objectif est de combler l'écart entre le cahier des charges, celui du client, et la spécification formelle, celle de l'informaticien. Dans ce papier, nous abordons la validation tout au long du processus de développement, en tenant compte des exigences et la spécification en Event-B du logiciel. La vérification peut aussi détecter des incohérences dans les exigences. La place des outils, notamment avec la plateforme Rodin, est importante au long de ce processus, améliorant sa qualité et sa documentation. Notre approche est illustrée par l'étude de cas d'un système de contrôle du train d'atterrissage d'un avion.*

ABSTRACT. *The amelioration of the quality of a system begins by the requirements elicitation. Our goal is to bridge the gap between requirements, those of the client, and the specification, this of the computer scientist. In this paper, we talk about the validation all along the development of a system, taking into account its requirements and its Event-B specification. The verification may also detect incoherences in the requirements. The Rodin platform is important all along to improve the quality and the documentation of the system, both of its specification and its development process. We illustrate our approach on the case study of an aircraft landing system.*

MOTS-CLÉS : *exigences, spécification, raffinement, validation, vérification, outils*

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

Congrès INFORSID, 34ème édition, Grenoble, 31 mai - 3 juin 2016, pages 1 à 16

1. Introduction

Un 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 Standisch 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. Ces exigences sont souvent très pauvres (Abrial, 2009), voire inexistantes.

Le processus de développement de systèmes à l'aide du raffinement utilisé dans Event-B 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 : 1) toutes les exigences sont explicitées pour décrire le modèle initial ; 2) le modèle initial est une description formelle répondant aux exigences de sécurité et fonctionnelles et 3) toutes les décisions prises lors d'un raffinement doivent être mémorisées relativement à une exigence. En réalité, peu de développements décrivent entièrement ces propriétés. Les exigences évoluent avec le développement, toutes les exigences 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 à chaque étape du développement. 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), 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.

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. Ces liens sont mis à jour tout au long du processus du développement, depuis le cahier des charges jusqu'à la spécification terminée. Différentes actions entre l'évolution des deux "mondes" sont présentées, voir figure 1. Le choix d'une ou plusieurs exigences 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.

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.

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

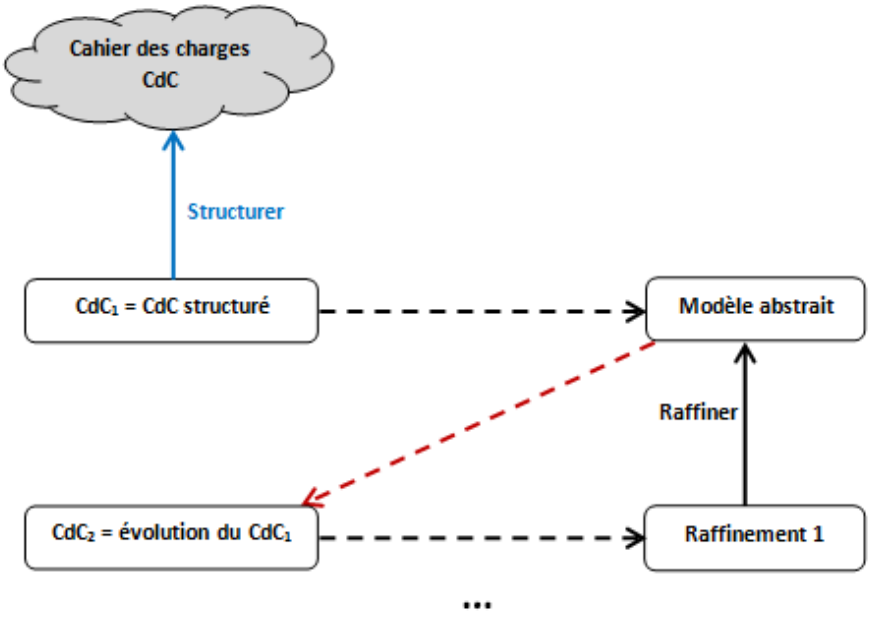

Figure 1. *Notre approche*

Dans la suite de cet article, la partie 2 présente brièvement la méthode Event-B, la plateforme Rodin et l'outil ProR pour prendre en compte les besoins informels en lien avec la spécification en Event-B. La partie 3 décrit la prise en compte de la validation dès la première étape du processus de développement, celle de l'expression de ses exigences en langage naturel. La partie 4 aborde la détection d'incohérences dans les exigences en liaison avec la vérification. 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). L'état de l'art sur le sujet est abordé dans la partie 5. Une conclusion est proposée dans la partie 6 avec nos travaux futurs.

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 : (1) le contexte pour décrire la partie statique du modèle à l'aide des ensembles énumérés, des constantes et des axiomes ; (2) 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.

– 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. La structure générale d'une machine en Event-B est la suivante :

MACHINE <*Machine_identifier*> **REFINES** <*Abstract_machine_identifier*> **SEES** < *Context_identifier_list* > **VARIABLES** < *variable_identifier_list* > **INVARIANTS** <label>: <*predicate*> ...	**EVENTS** *Event* < *event_identifier* > **REFINES** < *abstract_event_identifier* > **WHEN** <label>: <*predicate*> **THEN** <label>: <*action*> **END** **END**

– 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 est 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 POs.

– 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 un autre. La correction des raffinements est définie par les POs 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 "plug-ins" et fournit un soutien pour le raffinement et la preuve mathématique. Elle permet d'éditer, animer, prouver et contre-prouver les modèles. Parmi les plug-ins, deux outils complémentaires à la preuve sont indispensables pour la validation des modèles Event-B. Il s'agit d'un animateur permettant de détecter un ensemble de problèmes comme l'inter-blocage 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.

ProR est un plug-in 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. Elle s'exprime de la manière suivante :

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

ID	Description
FUN-G	The landing system's goal is maneuvring gears and their associated doors
FUN-G-1	Maneuvring gears consists on extending or retracting them and reversing their mouvement

Elle 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 manuellement des liens depuis la spécification Event-B vers les exigences texte sous ProR. Initialement, les exigences sont informelles,
- insérer des éléments formels dans les exigences de ProR, ces éléments étant issus de la spécification Event-B.

Cet outil enregistre des informations importantes pour les activités de validation et de vérification.

3. Validation

Nous abordons la validation tout au long du développement. Un premier travail commence par la compréhension des exigences et à les re-structurer, si nécessaire. Au fur et à mesure du développement de la spécification sous Rodin et en utilisant ProR, les exigences structurées sont mises à jour pour :

- ajouter des liens entre exigences et éléments formels de la spécification en cours,
- introduire des termes formels dans les exigences structurées, termes issus de la spécification. Ces termes entre crochets [] sont distingués du reste de l'exigence.

3.1. *Structuration des exigences*

La structuration du cahier des charges (CdC) a pour objectif l'obtention d'un document lisible et accessible par toutes ses parties prenantes. Dans notre étude de cas, celui d'un système de contrôle du train d'atterrissage d'un avion, le CdC est compréhensible en l'état. Par conséquent, notre activité consiste à ré-écrire ce document sous forme de phrases étiquetées, en utilisant la proposition de (Abrial, 2010). La validation est prise en compte dès la structuration du CdC et tout au long du processus de développement du système. Elle concerne les futurs modèles formels en Event-B par rapport aux exigences du client. Ces exigences s'expriment sous forme des :

- données du futur système (notées Fact dans la suite de ce document),
- fonctionnalités attendues (Functionality),
- comportements (Behavior) ou séquences de fonctions prévues ou non autorisées,
- conditions (Obligation) avec lesquelles le système fonctionne sous forme de pré-conditions et post-conditions (Hoare, 1969).

Cette phase du développement concerne les deux étapes suivantes :

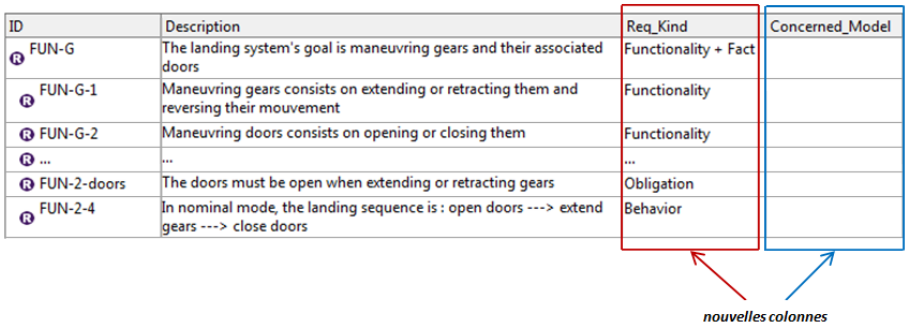

Figure 2. *Prise en compte de la validation avec ProR*

Req_Kind	**Extracted element**	**ID**
Fact	- gears - associated doors	FUN-G FUN-G
Functionality	- maneuvring gears and doors - maneuvring gears : extend gears, retract gears, reverse gears mouvement - maneuvring doors : open doors, close doors	FUN-G FUN-G-1 FUN-G-2
Behavior	- landing sequence : open doors –> extend gears –> close doors	FUN-2-4
Obligation	- *pre-condition* doors = open - *post-condition* extend gears, retract gears	FUN-2-doors

Tableau 1. *Eléments pour la validation*

1) La première consiste à ajouter des paramètres au document structuré avec ProR. La structure du CdC dispose de deux nouvelles colonnes contenant des informations relatives à la validation :

- La colonne *Req_Kind*. Un élément de cette colonne indique les catégories de l'exigence correspondante : *Fact*, *Functionality*, *Behavior* ou *Obligation*.
- La colonne *Concerned_Model*. Un élément de cette colonne fait le lien entre l'exigence et le nom du modèle Event-B correspondant, via des machines et des contextes.

Exemple. La figure 2 est décrite à l'aide de ProR. Elle contient ces deux nouvelles colonnes du CdC structuré. Le travail de spécification n'a pas encore commencé ; donc, nous n'avons pas encore introduit de modèles lors de cette étape.

2) La deuxième consiste à extraire les éléments destinés à la validation du futur modèle formel, présents dans les exigences du client. Ces éléments de type *Fact*, *Functionality*, et *Obligation* sont extraits en se posant certaines questions :

- Quelles sont les données présentes dans le futur système ?

- Quelles sont ses fonctionnalités attendues ?
- Sous quelles conditions le système devra fonctionner ?

Les comportements extraits du CdC sont décrits en termes de scénarios de validation. Ces scénarios décrivent les différents états du modèle Event-B en animant des évènements. Ils apparaissent dans le CdC structuré sous forme de phrases décrivant un enchaînement d'actions.

Exemple. Les éléments de validation extraits de la figure 2 sont présentés dans le tableau 1, relativement à chaque catégorie des exigences.

3.2. *Évolution des exigences et de la spécification*

Cette partie est illustrée par l'étude de cas à l'aide des différentes sortes d'exigences proposées dans la partie 3.1. Nous introduisons la machine `Gears_M0` et son contexte `Gears_Ctx0`. La colonne Concerned_Model introduite dans la figure 2 est mise à jour par les deux nouveaux noms de modèles `Gears_Ctx0` et `Gears_M0`. Ces noms correspondent au `context` et à la `machine` d'Event-B liés aux deux exigences FUN-G et FUN-G-1. L'état du développement est présenté dans la figure 3. La spécification Event-B en cours est proposée dans l'annexe.

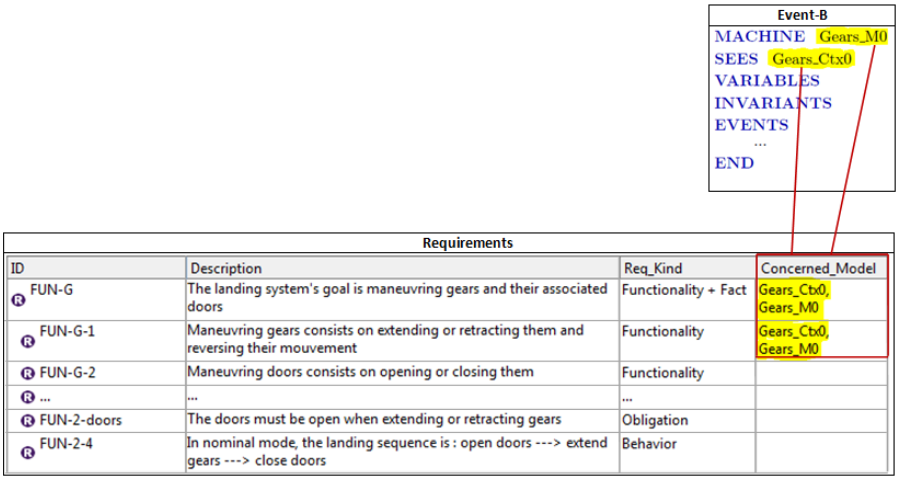

Figure 3. *Introduction d'une machine et son contexte*

3.2.1. *Introduction d'une donnée*

L'introduction de la variable `gears_pos` dans la machine `Gears_M0` se répercute dans la description des exigences : le terme informel `gears` dans les exigences FUN-G et FUN-G1 est remplacé par le terme formel `[gears_pos]` issu de la spécification `Gears_M0`. Les liens entre la spécification Event-B et les besoins sous ProR sont mis à jour. La figure 4 montre l'introduction de la variable `gears_pos`.

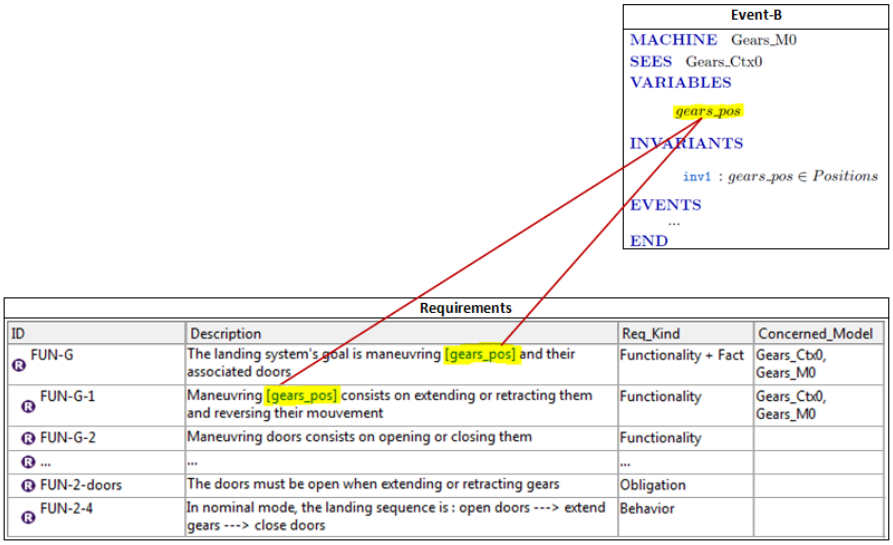

Figure 4. *Introduction d'une donnée*

3.2.2. *Introduction d'une fonctionnalité*

La figure 5 présente l'introduction de l'événement `open_doors` dans la spécification existante et dans le texte des exigences FUN-G-2 et FUN-2-4. La mise à jour du développement concerne :
- la spécification avec un nouvel événement,
- les nouveaux éléments formels dans les exigences,
- les liens entre eux, via ProR.

3.2.3. *Introduction d'une condition*

La figure 6 présente l'introduction de la garde `doors = open` pour les deux événements `extend_gears` et `retract_gears` dans la spécification existante. La mise mise à jour de l'exigence FUN-2-doors concerne :
- une garde dans deux événements existants dans la spécification,
- une nouvelle variable, `doors_pos`,
- les nouveaux éléments formels dans les exigences,
- les liens entre eux, via ProR.

3.3. *Validation de la spécification relativement au CdC*

Afin de valider chacun des modèles Event-B en cours de développement, relativement aux éléments de validation issus lors de la phase de structuration du CdC, nous procédons aux deux étapes suivantes.

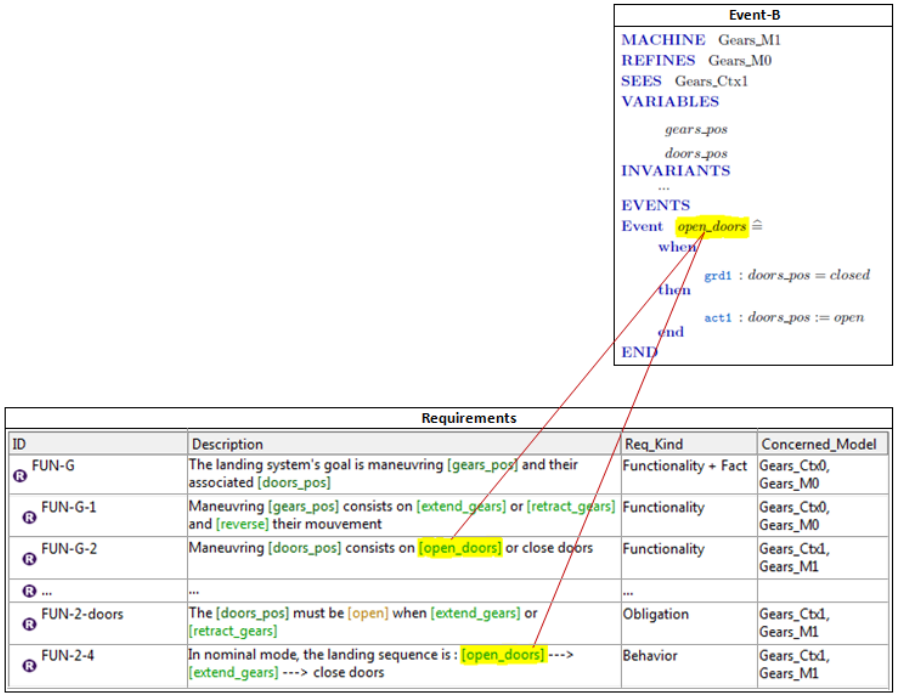

Figure 5. *Introduction de la fonction open_doors*

- Étape 1. Recherche d'éléments de types Fact, Functionality et Obligation dans le modèle décrit précédemment.

  Exemple. Dans la machine `Gears_M1`, les deux variables `gears_pos` et `doors_pos` correspondent aux éléments extraits de type Fact du tableau 1. Concernant la validation des éléments de type Functionality,

  - l'élément *maneuvring gears and doors* n'a pas été modélisé par notre choix de stratégie de développement,
  - les autres éléments de type Functionality ont été pris en compte.

  Ce résultat est présenté dans le tableau 2. L'élément de type Obligation a été modélisé par une garde nommée `grd_FUN2-2-doors` (voir la figure 6).

- Étape 2. Validation par animation du comportement du modèle exprimé par les éléments de type Behavior. L'animation est réalisée avec ProB en utilisant des scénarios.

  Exemple. Pour animer le modèle `Gears_M1` par rapport au scénario décrit dans le tableau 1, nous avons simulé la séquence d'événements Event-B suivante :

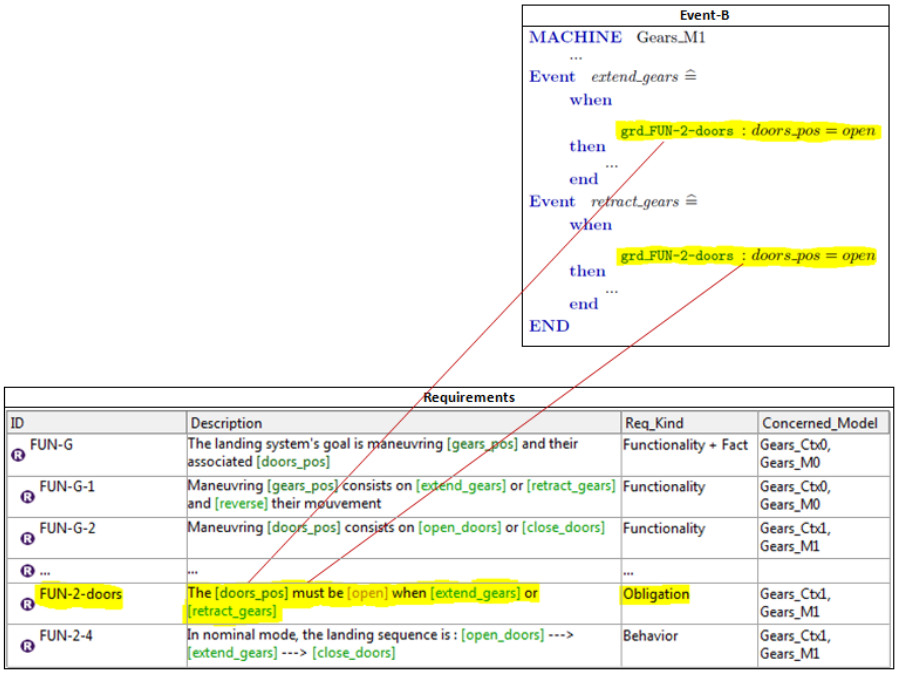

Figure 6. *Introduction d'une condition dans un événement*

Functionality	**Event-B element**
maneuvring gears and doors	non modélisé (choix de développement)
extend gears	event `extend_gears`
retract gears	event `retract_gears`
reverse gears mouvement	event `reverse`
open doors	event `open_doors`
close doors	event `close_doors`

Tableau 2. *Prise en compte des éléments de type Functionality*

```
INITIALISATION -> open_doors -> retract_gears -> close_doors
```

3.4. *Scénarios de validation pour l'animation*

Le CdC peut contenir plusieurs scénarios décrivant les comportements du futur système. Cependant, ces scénarios ne sont pas suffisants pour couvrir tous les comportements possibles. Afin de définir de nouveaux scénarios non prévus dans le CdC,

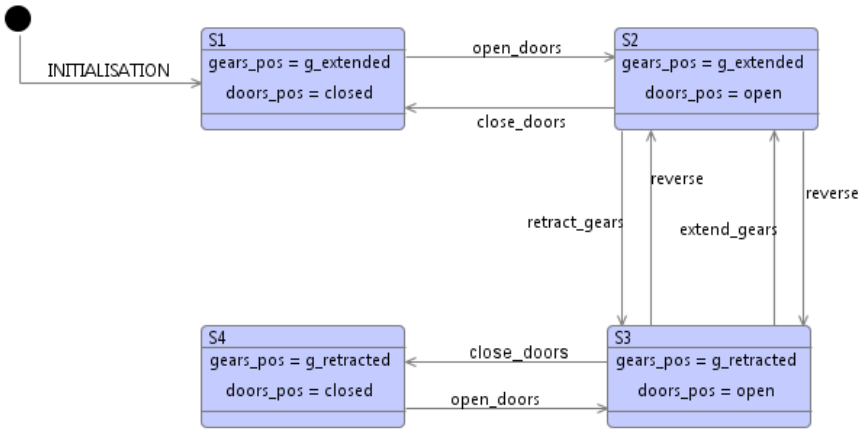

Figure 7. *Une sous-machine à états de* `Gears_M1`

nous utilisons les machines à états, State Machine SM, et une recherche de scénarios non permis par le futur système.

3.4.1. *Utilisation des machines à états*

Afin de construire la SM correspondante à une machine Event-B, nous utilisons Event-B Statemachines [3], un plugin de la plateforme Rodin. A chaque modèle, une SM est associée :
- chaque état représente une valeur donnée à chacune des variables Event-B,
- chaque transition entre deux états représente un événement.

Exemple 1. La figure 7 présente la SM correspondante à la machine Event-B `Gears_M1` de la figure 8. 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`.

Exemple 2. A partir de la figure 7, 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.

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

- *initialisation -> open doors -> retract gears -> retract gears -> close doors* : il n'est pas possible de rétracter les trains plus d'une fois dans la même séquence.

Ces scénarios ne sont pas permis par la machine `Gears_M1` lors de son animation avec ProB.

3.4.2. *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.

Exemple. A partir du scénario de validation décrit dans le tableau 1,

open doors –> extend gears –> close doors,

l'action *open doors* est introduite à nouveau 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 `Gears_M1` 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.

4. Vérification

L'activité de vérification permet de découvrir des incohérences dans les exigences. Celles-ci peuvent correspondre à des contradictions, des répétitions ou des oublis. Dans l'étude de cas, nous avons ajouté l'exigence suivante :

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

Cette exigence appelée FUN-i est modélisée en Event-B par l'invariant suivant :

```
gears_moving = TRUE => doors_pos = closed
```

Dans cet invariant, `gears_moving` est une variable booléenne qui indique si les trains sont en déplacement ou non (voir figure 8).

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 :

- Une contradiction entre la garde et cet invariant :
  - garde : `doors_pos = open`
  - invariant : `gears_moving = TRUE => doors_pos = closed`

- Cette garde et cet invariant modélisent deux exigences du CdC. Une contradiction entre ces exigences est détectée à l'aide des prouveurs de Rodin.

5. Etat de l'art

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 ré-écrire 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.

Notre approche commence par la re-structuration du document des exigences proposée par Abrial pour exprimer les liens entre les exigences et les spécifications ; nous ajoutons pas-à-pas des liens avec les modèles Event-B en cours de développement. La première étape de structuration est celle 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 spécification Event-B et les exigences.

Dans l'approche présentée dans (Heisel *et al.*, 1999), les exigences et la spécification sont clairement identifées. 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. Notre approche est une évolution 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 multi-vues 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. Dans notre approche, les exigences doivent être comprises avant de les structurer. Dans l'approche proposée dans (Ponsard *et al.*, 2015), les auteurs utilisent plusieurs sortes de diagrammes pour aider à comprendre les exigences.

L'évolution des besoins est prise en compte (Hallerstede *et al.*, 2014). Cela signifie que les exigences doivent être fréquemment changée et incorporée 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; Mashkoor *et al.*, 2016b) 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 la deuxième étape, des détails sont introduits graduellement dans les modèles formels. La troisième étape est d'utiliser l'outil UML-B et l'outil de décompostion 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.

6. Conclusion et perspectives

Dans ce papier, nous avons pris en compte la validation tout au long du processus de développement, depuis la structuration des exigences jusqu'à la spécification en Event-B d'un logiciel. Mémoriser les exigences implique sa mise à jour et sa place dans le processus de développement (Abrial, 2009). Nous avons utilisé la plateforme Rodin et illustré notre approche par l'étude de cas d'un système de contrôle du train d'atterrissage d'un avion [4]. Nous avons aussi développé une étude de cas d'une machine d'hémodialyse (Mashkoor, 2015).

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 ré-écrivons ces exigences sous forme de phrases étiquetées. Nous ajoutons de nouveaux paramètres au document structuré avec ProR et décrivons des scénarios de validation.

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. 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

4. Ce développement est accessible à l'adresse http ://dedale.loria.fr

des exigences et celui des spécifications formelles. Si nous regardons notre approche présentée dans la figure 1, le raffinement utilisé en Event-B pourrait être généralisé à l'ensemble des couples (besoins, spécifications) et des liens entre eux. 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. Concernant la validation, un travail sur des outils manuels et semi-automatiques est nécessaire en lien avec les différentes catégories des exigences.

7. Bibliographie

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

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

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

Alkhammash E., Butler M., Fathabadi A. S., Cirstea C., « Building Traceable Event-B Models from Requirements », *Science of Computer Programming (Special Issue on Automated Verification of Critical Systems, AVoCS 2013)*, vol. 111, Part 2, p. 318-338, 2015.

Boniol F., Wiels V., « Landing Gear System case Study », *ABZ Conference, Communications in Computer and Information Science, Springer*, vol. 433, p. 1-18, 2014.

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

Gunter C. A., Gunter E. L., Jackson M., Zave P., « A Reference Model for Requirements and Specifications- Extended Abstract », *Proceedings of the 4th International Conference on Requirements Engineering*, IEEE Software, p. 17 : 37-43, 2000.

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

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

Heisel M., Souquieres J., « 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, 1999.

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

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

Leuschel M., Butler M. J., « ProB : A Model Checker for B », *in* K. Araki, S. Gnesi, D. Mandrioli (eds), *International Symposium of Formal Methods Europe, Pisa, Italy, Proceedings*, vol. 2805 of *LNCS*, Springer, p. 855-874, 2003.

Mashkoor A., The Hemodialysis Machine Case Study, Technical report, Tecnical report SCCH-TR-1542, Austria : Software Competence center Hagenberg GmbH, 2015.

Mashkoor A., Jacquot J.-P., « Validation of Formal Specifications through Transformation and Animation », *Requirements Engineering, Springer Verlag, 16 pages*, 2016a.

Mashkoor A., Yang F., Jacquot J.-P., « Refinement-based Validation of Event-B Specifications », *Software and Systems Modeling, Springer Verlag, 33 pages*, 2016b.

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

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

van Lamsweerde A., « 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, 2008.

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

A. Annexe

MACHINE *Gears_M1*
REFINES *Gears_M0*
SEES *Gears_Ctx1*
VARIABLES
gears_pos
doors_pos
gears_moving
INVARIANTS
inv1: $doors\_pos \in Doors\_Positions$
FUN−i_inv: $gears\_moving = TRUE \Rightarrow doors\_pos = closed$

EVENTS
INITIALISATION
BEGIN
act1: $gears\_pos := g\_extended$
act2: $doors\_pos := closed$
END
Event extend_gears
REFINES
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
REFINES
retract_gears
WHEN
grd1: $gears\_pos = g\_extended$
grd_FUN−2−doors: $doors\_pos = open$
THEN
FUN−G−1: $gears\_pos := g\_retracted$
END
Event reverse
REFINES
reverse
THEN
...
END
Event open_doors
WHEN
grd1: $doors\_pos = closed$
THEN
FUN−G−2: $doors\_pos := open$
END
Event close_doors
...

END

Figure 8. *Machine* `Gears_M1`