HAL
archives-ouvertes.fr

Premières leçons sur la spécification d'un train d'atterrissage en B événementiel

Jean-Pierre Jacquot

▶ **To cite this version:**

Jean-Pierre Jacquot. Premières leçons sur la spécification d'un train d'atterrissage en B événementiel. C. Dubois; R. Laleau. AFADL 2014, Jun 2014, Paris, France. 2014. <hal-00982982>

HAL Id: hal-00982982
https://hal.inria.fr/hal-00982982

Submitted on 24 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Premières leçons sur la spécification d'un train d'atterrissage en B événementiel

Jean-Pierre Jacquot

LORIA – équipe DEDALE – Université de Lorraine
Vandoeuvre lès Nancy, France
Jean-Pierre.Jacquot@loria.fr

Résumé Ce papier présente les leçons préliminaires obtenues en traitant en B Événementiel l'étude de cas proposée par la conférence ABZ 2014. Le problème consiste à modéliser le logiciel de contrôle du train d'atterrissage d'un avion. L'utilisation de B Évémentiel sur cette étude pose des questions intéressantes quant à la nature des invariants, quant au moment de leur introduction, ainsi que quant à l'expression et la vérification des propriétés fonctionnelles. Le raffinement est organisé en niveaux d'observation structurés par la description du matériel. Le système est vu comme un automate assez simple piloté par des capteurs externes. La description d'un tel système en B Événementiel est simple mais sa validation est beaucoup plus difficile. Cette étape utilise JeB, un simulateur de B Événementiel en JavaScript. L'émulation des capteurs est un point crucial.

1 Introduction

Pour promouvoir l'usage des méthodes formelles, il est important d'analyser chacune en tant qu'outil. Dans cette perspective, une méthode est moins caractérisée par ses propriétés intrinsèques, par exemple le type de logique utilisée, que par la problématique de son emploi. Sur quel type de problèmes est-elle adaptée ? Peut-on l'adapter à d'autres contextes ? Quelles phases et activités du développement permet-elle de traiter ? Les environnements supports sont-ils adéquats ? En délimitant ainsi le périmètre de chaque méthode, nous facilitons le choix par les développeurs de la « meilleure » pour leur projet en cours.

Dans la veine de [7] qui explorait l'usage de B Événementiel [2] pour la modélisation de domaine, ce papier propose une première analyse de cette méthode sur l'étude de cas proposée par V. Wiels et F. Boniol [5]. Cette étude propose de modéliser le logiciel de contrôle de manœuvre du train d'atterrissage d'un avion. L'architecture du système comporte trois grands éléments :

- une interface de contrôle pour le pilote (levier et indicateurs lumineux),
- la partie mécanique et hydraulique actionnée via des électrovannes et observée par des capteurs, et
- le logiciel de commande.

B Événementiel n'est pas, *a priori,* le langage de choix pour ce type de système. De fait, trois caractéristiques introduisent des difficultés sensibles :

– Invariant faible. En première analyse, le système peut se concevoir comme un petit automate fini. B Événementiel permet d'écrire facilement un automate, états et transitions, mais ne permet pas de vérifier, c'est-à-dire, de prouver, que la spécification est bien celle de l'automate et qu'il a bien les bonnes propriétés. La raison tient principalement au fait que le parcours de l'automate s'exprime très mal à travers un invariant.
– Interface avec la partie matérielle. Le matériel possède un comportement propre et autonome dont doit tenir compte le logiciel de contrôle. La difficulté est moins dans l'expression des « communications » entre logiciel et matériel que dans la vérification/validation des bons comportements.
– Exigences temporelles. De nombreuses exigences sont exprimées comme des réponses à la (non) réalisation d'actions matérielles dans un délai donné. B ne possède pas de moyen d'expression natif pour ce type d'invariants qu'il est néanmoins possible de modéliser [11].

Ce papier traite uniquement des deux premières caractéristiques. Elles ont en commun le fait que si les preuves restent indispensables pour garantir le développement, il faut leur adjoindre des activités qui relèvent de la *validation.* Il convient en effet d'observer le comportement des modèles pour s'assurer que les états s'enchaînent bien, qu'il n'y a pas de blocage, etc. Il faut donc utiliser des outils d'animation et de *model-checking* comme AnimB [9] ou ProB[4]. Ce papier utilise JeB, un outil de simulation développé pour permettre la validation de modèle B Événementiels à tous les niveaux de raffinement [13].

La suite est structurée de la façon suivante. B Événementiel et JeB sont d'abord brièvement décrits. Puis, la stratégie générale de développement suivie sur l'étude de cas est présentée. La simulation des modèles est ensuite décrite. Enfin, une analyse des observations des différents outils est proposée.

2 Cadre formel et outils

2.1 B Événementiel

B Événementiel est un cadre de spécification basé sur trois idées :
– un système est décrit par un modèle formel constitué d'un état et d'un ensemble d'événements qui agissent sur l'état,
– un développement est une suite de modèles qui sont liés par une relation de raffinement formel,
– la sémantique de chaque modèle ainsi que celle des raffinements est donnée par un ensemble d'obligations de preuve.

La combinaison de ces trois éléments permet de définir une notion de correction pour chaque modèle qui garantit que le modèle final est une implantation

correcte du modèle initial lorsque toutes les obligations de preuves ont été démontrées. Ce cadre formel s'inscrit dans la démarche de production de logiciels « corrects par construction ».

Modèle Un état est une fonction associant des noms à des valeurs. Les valeurs sont des expressions ensemblistes construites à partir d'entiers, de symboles, des ensembles $\mathbb{N}$ et $\mathbb{Z}$, et d'ensembles de symboles (*carrier sets*). Des notations spécifiques permettent de construire facilement les relations–dont les fonctions– entre ensembles ainsi que les opérations telles que l'image d'un sous-ensemble par une relation ou l'inverse d'une relation. Le typage est défini comme l'appartenance à un ensemble. Syntaxiquement, B Événementiel distingue les constantes des variables qui décrivent respectivement les parties statiques et les parties dynamiques de l'état.

L'élément essentiel du modèle est *l'invariant* qui circonscrit l'ensemble des valeurs licites que peut prendre l'état. L'invariant est une formule logique du premier ordre portant sur les valeurs des variables et des constantes. Syntaxiquement, c'est une conjonction de prédicats. Un état licite est un état pour lequel l'invariant est vrai.

Un événement est une substitution gardée. La garde est un prédicat du premier ordre sur l'état. La substitution est une affectation parallèle de nouvelles valeurs à un sous-ensemble des variables. Un événement peut comporter des *paramètres* qui sont des variables libres susceptibles de prendre n'importe quelle valeur qui rend la garde vraie. Un événement est *déclenchable* si sa garde est vraie ; le déclenchement d'un événement provoque une évolution de l'état.

Raffinement Du point de vue du développeur, le raffinement est une opération qui consiste à transformer un modèle abstrait en un modèle plus concret par l'adjonction de nouveaux éléments qui rapprochent d'une implantation par un langage de programmation.

Concrètement, un raffinement en B Événementiel peut être vu comme une ou plusieurs opérations élémentaires.

- L'extension de l'état. Cette opération introduit de nouvelles variables et constantes sans relation avec l'état abstrait. Elle permet de décrire graduellement les différentes facettes d'un système.
- Le renforcement de l'invariant. Cette opération restreint l'espace dans lequel le modèle évolue à un espace plus proche de celui du système final.
- La réification de variables. Cette opération correspond au classique « raffinement de données » par lequel une variable abstraite est codée par une ou des variables plus proches d'une structure de données programmable.

– La décomposition d'un événement. Cette opération décompose un événement « global » en un ensemble d'événements « locaux ». Elle correspond à l'introduction d'un nouveau niveau d'observation du système.

Techniquement, chaque opération se traduit par une évolution de la plupart des composants du texte de la spécification : invariants, axiomes, gardes et actions. L'intérêt de la typologie des opérations est d'ordre méthodologique. Elle permet de mieux comprendre la différence entre le raffinement vu par la méthode B [1], la description plus précise d'un ensemble fixe de fonctions, et le raffinement vu par B Événementiel, l'enrichissement d'un modèle. Cette distinction n'a pas d'impact sur la définition formelle du raffinement. En revanche, elle induit la nécessité de nouveaux outils d'aide au développement.

Sémantiques La sémantique axiomatique de B Événementiel est structurée autour de trois grands principes. Un, le modèle doit être réalisable : l'ensemble des états licites n'est pas vide. Deux, l'invariant est préservé lorsqu'un événement est déclenché depuis un état licite. Une autre façon de penser cette propriété est que les événements ne permettent pas de rejoindre un état non licite. Trois, 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.

Grâce à une syntaxe adaptée, ces principes peuvent être traduits en obligations de preuve qui, appliquées à une spécification, génèrent un ensemble de théorèmes dont la démonstration garantit la correction du modèle. Cette sémantique est particulièrement adaptée pour le développement de systèmes dont l'implantation doit garantir le respect de propriétés exprimables comme une invariance dans un espace abstrait.

Le développeur a également besoin d'une vision plus opérationnelle des modèles afin de définir ou analyser les comportements du système. La sémantique opérationnelle de B Événementiel est intuitivement décrite par :

```
déclencher l'événement INITIALISATION
calculer l'ensemble des événements déclenchables (Ed)
tant que Ed ≠ ∅
     choisir un événement e dans Ed
     fixer les valeur des paramètres de e
     exécuter la substitution de e
     calculer Ed
```

L'arrêt peut correspondre à une anomalie dans le modèle (*deadlock*) ou au résultat souhaité (terminaison du calcul).

La caractéristique principale des sémantiques de B Événementiel concerne le non-déterminisme. Celui-ci est présent à trois niveaux : lors du choix des

paramètres des événements, lors du choix de l'événement à déclencher et lors du choix des valeurs à substituer. Hors extension de l'état, les opérations de raffinements sont généralement guidées par la diminution du non-déterminisme.

2.2 JeB

La conception du raffinement en B Événementiel comme un enrichissement du modèle formel implique que la question de la correction du développement doit être posée différemment. Si la preuve, c'est-à-dire la vérification, garantit toujours que les invariants fondamentaux sont maintenus, elle ne dit rien quant au fait que les modèles raffinés sont conformes aux exigences ou aux contraintes réelles. Assurer cette conformité est le rôle de la validation.

Parmi les techniques de validation, l'exécution du modèle formel est un outil de choix pour vérifier que les comportements spécifiés couvrent les scénarios prévus ou qu'il n'y a pas de comportement anormal (*deadlock* par exemple). L'exécution du modèle formel peut être réalisée de plusieurs façons. Le modèle peut être traduit en un programme rédigé dans un langage exécutable : E2B [12] ou EB2ALL [8] sont de tels compilateurs. Le modèle peut être directement interprété par un outil qui réalise la sémantique opérationnelle : AnimB[9] ou ProB [6] utilisent ce principe. Traduction et interprétation, encore appelée animation, sont efficaces sur des modèles suffisamment déterministes. En effet, les choix nécessitent d'énumérer des ensembles de valeurs. Même lorsque les ensembles sont suffisamment définis pour que leurs éléments soient énumérables, la technique bute souvent sur une explosion combinatoire.

JeB est un environnement conçu pour aider à la validation des modèles lorsque l'animation ou la traduction sont inopérantes. L'idée est de construire des *simulateurs* du modèle, c'est-à-dire des versions exécutables composées :

- d'une traduction automatique du modèle en JavaScript,
- d'un moteur d'exécution comportant la boucle de base, une bibliothèque des opérations ensemblistes en JavaScript et une interface HMTL de visualisation et de contrôle des simulations,
- de fragments de code en JavaScript fournis par l'utilisateur pour éliminer ou réduire les non-déterminismes massifs.

L'idée sous-jacente à JeB est qu'un développeur peut utiliser la démarche des « petits pas à partir d'un modèle très abstrait » défendue dans [3] tout en ayant une idée assez précise de la nature du résultat qu'il obtiendra. Il lui est ainsi possible de proposer très tôt des implantations raisonnables d'ensembles laissés indéfinis, ou d'imposer les trajectoires de comportement qui sont les plus pertinentes. L'implantation de JeB est décrite dans la thèse de Yang [13].

La question de la fidelité de la simulation au modèle se pose naturellement dès qu'une intervention humaine est nécessaire. Une définition formelle de la

notion de fidélité est proposée dans [14]. Elle donne lieu à la création d'obligations de preuves qui permettent de garantir que les comportements observés sur une simulation sont bien spécifiés par le modèle.

3 Stratégie de développement

3.1 Présentation de l'étude de cas

L'étude de cas traite du développement d'un système de contrôle du train d'atterrissage d'un avion. Le point de départ est un cahier des charge informel qui décrit l'architecture matérielle, le comportement des différents composants et les exigences pour le logiciel de commande.

Au niveau le plus externe, le train d'atterrissage se compose de trois atterrisseurs et d'une interface à usage du pilote. Chaque atterrisseur est contitué d'une trappe, d'une jambe et de verrous. L'interface comporte le levier de commande et des indicateurs lumineux informant de l'état du train complet. Au niveau le plus interne, chaque atterrisseur se compose des pistons hydrauliques de manœuvre de la jambe et de la trappe, ainsi que des micro-capteurs qui détectent la position de la trappe et de la jambe. Le niveau intermédiaire se compose des électro-vannes de commande des pistons, des capteurs virtuels agrégeant les micro-capteur et des capteurs globaux (pression, alimentation électrique, etc.).

Les comportements sont décrits à travers les séquences d'états observées en fonctionnement nominal et la caractérisation des états pour les capteurs et les indicateurs. Une importance particulière est donnée à la description du temps de latence des manœuvres élémentaires des équipements hydrauliques. Des contraintes temporelles, liées aux temps de latence, entre différents commandes sont décrites. La définition de la « santé » du système et des actions de traitement des fautes détectées est détaillée.

Les exigences sont données sous la forme de deux listes. La première liste concerne le fonctionnement nominal. Elle contient des items d'ordre fonctionnel, c'est-à-dire des états à atteindre. Par exemple (page 18 de [5]) :

> *(R_{11}bis) When the command line is working (normal mode), if the landing gear command handle has been pushed DOWN and stays DOWN, then eventually the gears will be locked down and the doors will be seen closed.*

Les items d'ordre non-fonctionnel sont des contraintes sur l'enchaînement de certaines actions (page 18 de [5]) :

> *(R_{31}) When the command line is working (normal mode), the stimulation of the gears outgoing or the retraction electro-valves can only happen when the three doors are locked open.*

La seconde liste contient la définition des états anormaux qui doivent être détectés sur de critères temporels, principalement le dépassement de délais (page 19 de [5]) :

(R_{61}) If one of the three doors is still seen locked in the closed position more than 0.5 second after simulating the opening electro-valves, then the boolean output normal_mode *is set to* false

Au final, le cahier des charges pose 19 exigences, dont 12 (ou 10 dans une version simplifiée des exigences fonctionnelles) concernent le respect de délais.

3.2 Développement et raffinements

L'approche « naturelle » des méthodes basées sur le raffinement est d'exprimer les exigences comme des invariants sur un état. La démarche débute alors par la définition d'une abstraction de l'espace où évoluera le système à définir. Si le choix est judicieux, les invariants et les fonctions peuvent être décrits de façon compacte et complète. Cette compacité rend possible la validation du modèle par des procédures de lecture attentive par des experts du domaine.

Dans le cas présent, les exigences et contraintes sont décrites sur des éléments de bas niveau impliqués dans le fonctionnement des composants physiques. En fait, celles-ci sont des contraintes sur la dynamique interne du système. Il y a peu de possibilité d'abstraction du système physique pour exprimer tous les invariants. En conséquence, la démarche usuelle a de fortes chances d'amener à la construction d'un modèle initial de grande taille, donc très difficile à valider. Par ailleurs, la modélisation explicite du temps en B Événementiel ne fait qu'ajouter à la difficulté.

Une approche différente, conforme à l'esprit d'introduction la plus progressive possible de la complexité, a été choisie. Elle repose sur une stratégie générale de description de « l'extérieur vers l'intérieur » correspondant à l'introduction progressive des composants matériels. Les exigences sont donc introduites parallèlement à la progression des raffinements.

Ce papier utilise les cinq premiers modèles du développement (modèle initial et 4 raffinements) [1].

Modèle initial Ce modèle décrit l'état macroscopique de l'avion : fonctionnement nominal ou non, train déployé ou non. Il possède trois événements : prepare_landing, prepare_flight et alert. La vérification et la validation de ce modèle sont triviales.

Raffinement 1 Le premier raffinement est un changement de niveau d'observation. Il est guidé par les composants matériels externes qui participent à l'action : trappe, jambe et verrous, levier de commande. Les mouvements se décrivent facilement par des automates, la spécification encode simplement ceux-ci par un

1. Le texte B Événementiel est disponible sous forme d'archive Rodin à l'adresse : http://dedale.loria.fr/?q=en/etude-train-atterrissage

état (au sens de B) comportant les états (au sens des automates) des composants et des événements modélisant les transitions. Ainsi, prepare_landing se décompose en

- switch_handle_down : commande du pilote,
- initiate_prepare_landing : mise en activité du train,
- landing_unlock : déverrouillage et ouverture des trappes,
- landing_start_lowering : descente de la jambe,
- landing_stop_lowering : jambe descendue,
- landing_lock_down : verrouillage et fermeture des trappes.

Le modèle comporte une vingtaine d'événements dont six correspondent à l'exigence (implicite) que les manœuvres puissent être interrompues et inversées à tout moment. La vérification est triviale puisque l'invariant ne concerne que le typage des nouvelles variables. La simulation avec JeB ne pose pas de difficulté et il est aisé de valider le modèle en exécutant tous les scénarios possibles.

Raffinement 2 Le deuxième raffinement est une extension de l'état pour introduire les indicateurs lumineux. Les contraintes sur les indicateurs sont décrites dans l'invariant, par exemple :

" no_light_failure " (operating_mode=normal) ⇒ (light_failure= light_off)
" light_failure " (operating_mode=emergency) ⇒ (light_failure=light_on)

Les gardes des événements sont inchangées. Comme les indicateurs lumineux changent de valeur à l'occasion de certaines transitions, la partie action des événements les codant est modifiée. La vérification du modèle n'est pas complexe mais se révèle fastidieuse du fait de la multiplication des analyse par cas. La validation est facile une fois développée une visualisation graphique et colorée des indicateurs et atterrisseurs.

Raffinement 3 Le troisième raffinement est également une extension de l'état qui introduit les capteurs. Un capteur est ici l'entité logique perçue par le système ; sa définition à partir des micro-capteurs sera réalisée dans un raffinement ultérieur. Comme lors du raffinement précédent, seules les parties action des événements sont concernées. Par exemple, pour l'événement prepare_landing, le capteur d'ouverture de trappe doit indiquer `faux` et l'interrupteur général `ouvert` :

"door_open" door_open := LANDING_SET ∗∗ {sfalse}
"switch" analog_switch := open

Il peut sembler surprenant que les capteurs se voient affectés des valeurs dans la partie action des événements. En fait, il faut interpréter ces affectations comme « les valeurs qui devront être lues après que l'événement ait eu lieu ». Ceci sera utilisé pour la vérification du raffinement qui introduira la lecture effective

des micro-capteurs. La vérification et la validation sont dans la continuité du raffinement précédent.

Raffinement 4 Le quatrième raffinement est un nouveau changement de niveau d'observation. Il est guidé par l'introduction des micro-capteurs et de leur lecture. Le cahier des charges indique que chaque capteur logique est le résultat de l'agrégation de trois micro-capteurs. À ce stade, les fonctions d'agrégation sont simplement postulées. La stratégie de modélisation suivie est d'introduire un seul événement de lecture par groupe de capteurs : un pour les capteurs liés aux atterrisseurs et aux circuit généraux, et un pour le capteur lié au levier de commande dans le cockpit. Ce dernier est exprimé comme :

```
event read_handle_sensor
any h_s
where
 "flipped−state" flipped = strue
 "handle_phase−in" handle_phase = reading
 "h_s−type" h_s ∈ 1..3 → HANDLE_STATES
then
 "μ_handle−out" μ_handle := h_s
 "handle_phase−out " handle_phase := read
```

La phase code l'état d'un protocole qui permet de décomposer les événements et de synchroniser les lectures. Le protocole peut se schématiser ainsi :

$$Evt_{R_3} \triangleq init\_Evt;\ do\_Evt;\ read\_sensors;\ Evt_{R_4}$$

où la gauche de $\triangleq$ est un événement du Raffinement 3 et la droite l'enchaînement des événements du Raffinement 4 qui le réalisera.

Concrètement, le Raffinement 3 est transformé de manière systématique par le méta-algorithme suivant :

```
Pour chaque évènement Evt
            introduire init_Evt et do_Evt
            mettre les actions sur les capteurs dans les gardes
            (les affectations devenant des égalités)
            remplacer les valeurs dans les actions par
            l'agrégation des valeurs lues des micro-capteurs
```

Cette structure pourrait être simplifiée, mais elle présente trois avantages : l'indépendance des mouvements des atterrisseurs est conservée, la synchronisation des événements clés est garantie, le terrain pour l'introduction des contraintes de temps de chaque mouvement est préparé. En particulier, l'introduction des événements *Init_Evt* servira à la mise en place des *timers.* Il faut noter que la synchronisation est ici une propriété émergente (implicite) qui modélise le fait qu'un seul circuit hydraulique physique assure les mouvements.

La vérification est fastidieuse (longues analyses par cas) mais ne présente pas de difficulté majeure. La validation a posé quelques dificultés qui seront exposées dans la section suivante.

4 Validation des comportements

En B Événementiel, la démonstration des obligations de preuve garantit la correction des modèles et du développement. Néanmoins, celles-ci traitent presque exclusivement de la partie fonctionnelle du système. Dans les cinq premiers modèles, seule la gestion des indicateurs du cockpit et des valeurs des capteurs relève de cette partie fonctionnelle. Les preuves ne garantissent pas que les comportements spécifiés sont bien ceux attendus. Ainsi, les premières versions du Raffinement 4 étaient prouvées bien qu'elles ne respectaient pas l'exigence de réversibilité de la manœuvre. Il faut donc des outils d'analyse des comportements.

4.1 La visualisation graphique

La figure 1 montre l'exécution d'une simulation du Raffinement 4 avec JeB. Le bas de la fenêtre visualise le modèle formel dans toutes ses composantes : état à gauche, événements au centre et historique du scénario à droite. Cette partie est générée automatiquement. La partie haute présente une vue graphique qui facilite l'analyse des comportements. La réalisation de cette partie est sous la responsabilité totale des utilisateurs.

La programmation du graphisme est réalisée en utilisant l'API `canvas` de HTML5. JeB fournit les points d'accès nécessaires pour observer les valeurs dans l'état. Il est également possible d'agir sur la simulation en « basculant » le levier ou en désignant certains micro-capteurs pour les rendre défectueux.

La réalisation du graphisme ne présente aucune difficulté autre que d'obtenir une interface visuellement plaisante, lisible et portant l'information nécessaire en utilisant des primitives graphiques de bas niveau.

La table 1 donne la répartition des lignes de code JavaScript pour la réalisation de la simulation du Raffinement 4. Elle permet de comparer le nombre de lignes de JavaScript selon leur provenance (générées par JeB ou programmées par l'utilisateur). La partie programmée est elle-même décomposée selon leur fonction dans la simulation. Il faut noter que le code utilisateur est développé incrémentalement : toutes les constantes introduites pour un raffinement sont nécessaires pour les suivants, une grande partie du code graphique et des fonction d'acquisitions des paramètres est conservée entre les raffinements.

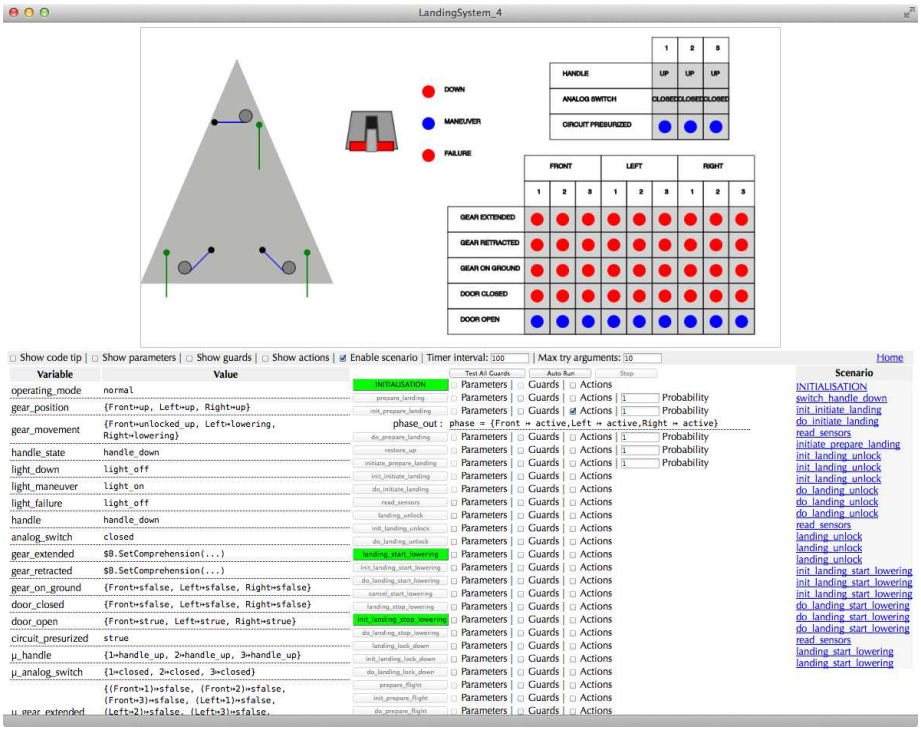

FIGURE 1. Simulation du Raffinement 4

Générées par JeB		10 100
Programmées à la main		
	Valuation des constantes	410
	Graphisme	410
	Émulation capteurs	630
Total manuel		1 450
Total simulation		11 550

TABLE 1. Répartition de l'effort de programmation

4.2 L'émulation du matériel

Les chiffres de la table 1 montrent que l'émulation du comportement du système matériel est ce qui a requis le plus d'effort. La difficulté est de convevoir un émulateur qui possède le comportement attendu tout en autorisant la simulation à être pilotée via l'interface générée par JeB. En effet, il faut que l'utilisateur puisse choisir librement l'événement à déclencher au cas ou il y en a plusieurs, il faut aussi qu'il y ait une synchronisation entre la boucle de la sémantique opérationnelle du modèle et la lecture des valeurs fournies par l'émulateur.

Techniquement, les micro-capteurs sont codés comme des objets. Leur état interne contient une table qui associe à chaque état du mouvement du train la valeur nomimale du capteur et l'état courant. Ils ont deux méthodes : la lecture qui donne la valeur associée à l'état *suivant* et l'avancement de l'état. L'événement read_sensor utilise ces deux méthodes : la lecture lors de la génération des valeurs des paramètre et l'avancement lors de la réalisation des actions.

Cet émulateur d'un système parfait a permis de valider la spécification qui décrit les comportements nominaux. Il a été indispensable pour mettre au point le comportement de réversibilité. C'est par l'exécution de différents scénarios avec JeB et l'émulateur que le petit protocole sur les capteurs a été augmenté d'une phase d'attente pour la re-synchronisation lors de l'interruption de la manœuvre. C'est également à travers ces exécutions que les événements de gestion de l'interruption ont été rétroactivement introduits dans le Raffinement 1.

5 Observations sur les outils

Au moment où ce papier est écrit, le modèle comporte 55 événements, environ 1 300 lignes pour le Raffinement 4 et 260 lignes pour les différents contextes. Bien que des parties importantes de l'étude de cas restent à aborder, notamment la modélisation des délais, certaines leçons peuvent être tirées dès maintenant.

5.1 B Événementiel

B Événementiel n'a pas été, *a priori,* conçu pour traiter ce type de problème, néanmoins, son usage n'a pas soulevé de difficulté particulière. L'absence de moyen d'exprimer facilement un automate (il faut le coder explicitement) est compensée par la simplicité du modèle opérationnel.

La plus grosse difficulté d'usage de B Événementiel tient surtout à la gestion des remises en cause. La démarche méthodologique associée au raffinement formel est un modèle en cascade : d'abord on fixe les exigences, puis on les formalise abstraitement, puis on construit en garantissant la conformité aux

exigences. Ce modèle de développement ne prévoit pas de retour en arrière et suppose que le modèle initial est complet. Or, peu de développements respectent ces contraintes : les exigences évoluent [10] et il est rare de trouver tout de suite la « bonne » solution.

Dans le cadre de ce développement, la gestion de l'interruption des manœuvres est un cas typique. La solution initiale, introduite au Raffinement 1, fonctionnait jusqu'au Raffinement 3. Ensuite, il est apparu qu'elle ne pouvait pas être raffinée pour intégrer la re-synchronisation de la lecture des capteurs. D'un point de vue théorique, il faut effacer et refaire les raffinement depuis le point où est introduite la nouvelle solution. D'un point de vue pratique, il est plus facile de penser en terme de modification, c'est-à-dire en terme d'une opération qui fait quelques petits changement et garde le reste. Circonscrire le changement, évaluer les conséquences, propager la modification : ces opérations ne sont pas prises en compte aujourd'hui. Ceci ralentit beaucoup le développement.

5.2 Rodin

Le Raffinement 4 génère environ 500 obligations de preuves visibles, c'est-à-dire, qui ne sont pas trivialement vérifiées. De l'ordre d'un tiers requièrent une intervention du développeur, souvent limitée au choix d'un démonstrateur. Un quart environ nécessite de guider la preuve, principalement en organisant des analyses par cas et des factorisations d'expressions. Rodin et les démonstrateurs associés sont assez efficaces.

Deux points techniques méritent une amélioration. Lors des modifications rétroactives, le système complet est re-vérifié : toutes les obligations de preuves sont considérées comme non prouvées et l'ancienne preuve est rejouée. Trop souvent, les outils laissent des obligations dans un état *à prouver* alors que la simple ouverture de l'obligation dans le démonstrateur interactif conclut la démonstration. L'outil a donc conservé assez d'information mais ne l'a pas exploitée. Cette situation fait perdre énormément de temps. Sauf dans les cas où on fait de la « mise au point » de gardes, d'invariants ou de comportement compliqués, il est nécessaire de s'assurer que le modèle est vérifié, c'est-à-dire prouvé, avant de générer les simulateurs. Il est inutile de valider un modèle incorrect.

Le second point concerne la fragilité de Rodin devant les spécifications de grande taille. Les blocages de l'environnement ne sont pas rares. Il arrive, malheureusement, que les fichiers soient corrompus au point de nécessiter une recréation complète du raffinement.

Enfin, la longueur du texte de la spécification commence à poser des difficultés pratiques. Les vues proposées soit totalement structurelles, soit totalement aplaties, ne sont plus suffisantes pour une lecture et une édition efficaces.

5.3 La validation et JeB

La proportion du code programmé à la main est d'environ 13% du total (en nombre de lignes de JavaScript). C'est un peu plus important que dans les cas que nous avions traités précédemment : environ 4%. Il est possible d'analyser ces chiffres en fonction du découpage par domaine de la table 1.

La valuation des constantes correspond à une partie de code qu'il est possible de générer. Ceci nécessiterait de faire une analyse des valeurs pour détecter les cas fréquents tels que la création d'ensembles de symboles par exemple. Ce cas se répète plusieurs fois dans l'étude et le code JavaScript reproduit directement la spécification.

Le graphisme occupe une part importante. La leçon, peu originale, est qu'il y a besoin de bibliothèques de formes, d'images et d'interacteurs simples tels que des boutons pour faciliter la réalisation des interfaces. Un des objectifs de l'interface était d'étudier la possibilité de contrôler le déroulement d'une simulation à partir du graphisme. La faisabilité, avec un coût raisonnable, de cette idée confirme la pertinence du choix HTML/JavaScript pour les simulations.

L'émulation des capteurs est la partie la plus novatrice. Vis-à-vis de la sémantique du modèle, l'émulateur produit des valeurs de paramètres pour deux événements. L'affectation de valeurs aux paramètres étant une des principales causes de non-determinisme des simulations, JeB propose l'infrastructure pour créer et utiliser des fonctions de génération d'arguments. Dans les modèles précédemment traités avec JeB, les générateurs d'arguments étaient pour l'essentiel des algorithmes tirant aléatoirement des valeurs. Pour ce modèle, la démarche est différente : les valeurs proviennent d'un mécanisme déterministe qui est totalement extérieur au modèle.

La configuration simulateur et émulateur repose la question de la fidélité des simulations par rapport à la spécification. Lorsque l'émulateur est produit, comme ici, en regard de la spécification, il est légitime d'interroger la correction de l'implantation vis-à-vis du mécanisme réel. Lorsque l'émulateur provient d'une autre source, il est possible d'admettre sa conformité, mais il faudra l'interfacer avec le simulateur. Concrètement, une couche logicielle doit être développée. Cette couche a pour fonction d'abstraire la vue du comportement concret de l'émulateur au niveau abstrait attendu par le simulateur. L'interrogation porte alors sur la transparence de l'interface vis-à-vis des comportements.

6 Conclusion

Trois leçons se dégagent à ce stade du développement de l'étude de cas :

– Le choix de B Événementiel est raisonnable pour développer ce type de système, à condition d'avoir de bons outils de validation.

– JeB, malgré quelques défauts de jeunesse, confirme son potentiel pour construire rapidement des simulations fidèles.
– L'émulation et, plus généralement, la connexion de système concrets externes pour la validation de modèles abstraits posent des questions auxquelles la communauté devra apporter des réponses.

Le travail sur l'étude doit se poursuivre pour aborder deux questions. Les patrons d'introduction du temps « tiendront-ils la charge » dans une spécification de grande taille ? Quelle part de l'investissement mis dans la validation peut être conservée dans les raffinements à venir ?

Références

1. Abrial, J.R. : The B Book. Cambridge University Press (1996)
2. Abrial, J.R. : Modeling in Event-B : System and Software Engineering. Cambridge University Press (2010)
3. Abrial, J.R. : Formal methods in industry : achievements, problems, future. In : Proc. of the 28th int. conf. on Software engineering. pp. 761–768. ICSE '06, ACM, New York, USA (2006).
4. Bendisposto, J., Leuschel, M., Ligot, O., Samia, M. : La validation de modèles Event-B avec le plug-in ProB pour RODIN. Technique et Science Informatiques 27(8), 1065–1084 (2008)
5. Boniol, F., Wiels, V. : Landing gear system. http://www.irit.fr/ABZ2014/landing_system.pdf (2013), case-study for ABZ'2014
6. Hallerstede, S., Leuschel, M., Plagge, D. : Refinement-animation for event-b – towards a method of validation. In : Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) Abstract State Machines, Alloy, B and Z, LNCS, vol. 5977, pp. 287–301. Springer Berlin Heidelberg (2010),
7. Mashkoor, A., Jacquot, J.P. : Utilizing Event-B for Domain Engineering : A Critical Analysis. Requirements Engineering 16(3), 191–207 (2011)
8. Méry, D., Singh, N. : Automatic Code Generation from Event-B Models. In : Proc. Symposium on Information and Communication Technology. ACM, Hanoi, Vietnam (2011)
9. Métayer, C. : B model animator. Website (2012), http ://www.animb.org
10. Nakatani, T., Tsumaki, T., Tsuda, M., Inoki, M., Hori, S., Katamine, K. : Requirements Maturation Analysis by Accessibility and Stability. In : Software Engineering Conference (APSEC), 18th Asia Pacific. pp. 357–364 (2011)
11. Rehm, J. : Gestion du temps par le raffinement. Ph.D. thesis, Nancy-Université — Université Henri Poincaré (2009)
12. Wright, S. : Automatic Generation of C from Event-B. In : IM_FMT, Workshop on Integration of Model-based Formal Methods and Tools. Dusseldorf, Germany (2009)
13. Yang, F. : A Simulation Framework for the Validation of Event-B Specifications. Ph.D. thesis, Université de Lorraine (November 2013)
14. Yang, F., Jacquot, J.P., Souquières, J. : Proving the Fidelity of Simulations of Event-B Models. In : 15th IEEE International Symposium on High Assurance Systems Engineering (HASE), Miami, USA (2014)