B-RAIL – Passage d'une modélisation semi-formelle à une modélisation formelle
B-RAIL – From a semi-formal model to a formal model

Boulanger J-L
Université de Technologie de Compiègne
Laboratoire Heudiasyc UMR CNRS 6599
Centre de recherche de Royallieu
BP 20529
60205 Compiègne cedex

Bon P. et Mariano G.
INRETS
ESTAS
20 Rue Elysée Reclus

59650 Villeneuve d'Ascq

Résumé

La modélisation des systèmes ferroviaires complexes reste une problématique difficile surtout si ceux-ci assurent des fonctions liées à la sécurité. Parmis les différents systèmes ferroviaires, nous avons choisi de nous intéresser aux « passages à niveau ». Dans le cadre de cet article, nous proposons une méthodologie de modélisation basée sur l'utilisation du couple UML/B. La notation UML permet de décrire le système dans son environnement. L'impact des défaillances est formalisé sous forme de use-case et de diagramme de séquence. Le comportement du système est décrit sous forme d'un graphe de classe et pour chaque classe, on caractérise son comportement par un diagramme d'état/transition. Nous proposons un processus de traduction du modèle UML vers le langage B, qui se focalise actuellement sur les diagrammes état/transition. Ces travaux se poursuivent au travers de l'intégration des travaux sur la traduction de contrainte OCL en B dans notre processus.

Summary

During the critical software development process, safety and security requirements must be traced from informal specifications to code generation. So we need to trace them in the different models: informal, semi formal or formal ones. In guided transport, these requirements, particularly in safety, are very strict. We present a new method here to transform a semi formal modelling to a formal specification which enables them to be traced. This method has been applied to a level crossing case study. This study is integrated into a larger one that aims at linking an informal approach (UML notation) to a formal (B method) one.

Introduction

Dans le cadre de cet article, nous présentons les premiers résultats d'une étude menée sur le passage d'une modélisation semi-formelle à une spécification formelle d'un passage à niveau (noté PN). Cette étude s'intègre dans une recherche plus large visant à étudier les liens entre modèles non-formels (notation UML) et modèles formels (méthode B) dans le contexte du domaine ferroviaire. Dans ce domaine, où les exigences en matière de sûreté de fonctionnement sont des plus sévères, il est important de préserver la traçabilité de ce type d'exigences au cours du processus de développement logiciel et donc entre les différents modèles utilisés qu'ils soient non-formels, semi-formels ou formels. Il existe des approches comme par exemple [10] et [9] qui proposent une méthodologie basée sur UML et la méthode B mais notre processus tente de prendre en compte l'aspect sûreté de fonctionnement des systèmes modélisés.

Notre présentation se décompose en 3 parties. Dans la première partie, nous présentons l'étude de cas. La seconde partie introduit la notation UML et la modélisation semi formelle du passage à niveau. La dernière partie commence par une courte présentation de la méthode B et se poursuit par une présentation du modèle B généré à partir du modèle UML. Enfin nous concluons et présentons quelques perspectives.

Etude de cas

Un passage à niveau est un lieu de croisement entre un flux de circulation routier et un flux de circulation ferroviaire. La principale règle de sécurité, pour ce genre de situation, consiste à éviter l'entrée simultanée du traffic routier et du traffic ferroviaire dans la zone définie par le passage à niveau ([6]).

Nous considérons en premier lieu une ligne à voie ferrée unique qui croise une route au même niveau. La zone d'intersection est appelée zone d'annonce. Les passages à niveau peuvent être gardés ou à signalisation automatique lumineuse et sonore (noté SAL). La sécurité des passages à niveau repose généralement sur la fermeture des barrières. Les feux routiers seront composés d'un seul feu bicolore (éteint, orange, rouge). Lorsque ce dernier est éteint, les utilisateurs de la route (voitures, piétons, ...) ont la possibilité de traverser le passage à niveau. Dans le cas contraire, le passage à niveau sera fermé et le traffic ferroviaire sera prioritaire.

Comme leurs noms l'indiquent, les passages à niveaux gardés [14] sont gérés par des gardes. Ces derniers doivent assurer la sécurité de ces passages soit en fermant les barrières dès l'approche d'un train au passage ou en demandant l'arrêt du ou des trains en cas de problème dans le passage en question. Ce type de passage à niveau a tendance à disparaître.

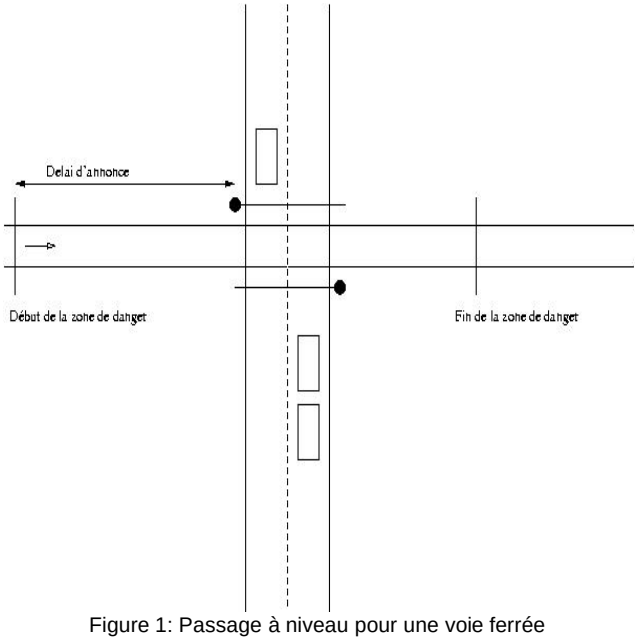


Figure 1: Passage à niveau pour une voie ferrée

Notre étude se focalisera sur le cas des passages à niveau français à signalisation automatique lumineuse et sonore pour une voie de circulation ferroviaire. Le fonctionnement en est le suivant. La détection de l'approche d'une circulation ferroviaire dans un passage à niveau à signalisation automatique lumineuse et sonore se fait par l'intermédiaire de détecteurs (capteurs) placés sur la voie. Cette approche est signalée, aux usagers de la route, silmutanément par l'allumage des feux routiers, le tintement des

sonneries et l'abaissement des barrières. Le signal sonore peut être atténué ou même supprimé en cas de gêne notable pour les riverains. Ces indications seront maintenues jusqu'au dégagement du passage à niveau. Après, elles disparaîtront progressivement, en commençant par relèvement des barrières, puis par l'extinction des feux et enfin, si nécessaire, l'arrêt du signal sonore.

Ce système de contrôle sera activé dès le déclenchement des informations d'annonce. Ce déclenchement est provoqué par les circulations ferroviaire elles-mêmes. Le point à partir duquel l'annonce sera déclenchée s'appelle o*rigine d'annonce*. Un délai est nécessaire entre le déclenchement de l'annonce et l'arrivée du train le plus rapide au passage à niveau. Ce délai, appelé *délai d'annonce*, est fonction de la vitesse maximale des trains au passage à niveau.

Acquisition du problème: Modélisation semi-formelle

Présentation générale de la notation UML

Né de la fusion des méthodes objets dominantes (OMT, Booch et Jacobson), puis normalisé par l'OMG[1] (Object Management Group) en 1997, UML (Unified Modeling Language) est rapidement devenu un standard incontournable. La notation UML ([12]) permet de modéliser une application selon une vision *orientée objet*. Cette modélisation d'un système est réalisée au travers de neuf types de diagrammes différents apportant chacun sa pierre à l'édifice du système.

Chaque diagramme permet de décrire une vue particulière du système:

- les cas d'utilisation (Use Case Diagram),
- les composants (Component Diagram),
- les collaborations (Collaboration Diagram),
- les classes, (Class Diagram),
- le déploiement, (Deployement Diagram),
- l'état, (State Diagram),
- l'activité (Activity Diagram),
- les séquences, (Sequence Diagram)
- les objets, (Object Diagram)

Le lecteur intéressé par plus d'élément sur l'aspect syntaxique et sémantique pourra se reporter au guide de référence "Unified Modeling Language" ([13]).

Bien que la notation UML soit un langage qui permet de représenter des modèles, il ne définit pas le processus d'élaboration de ces modèles. Cependant, dans le cadre de la modélisation (\cite{BoochAl1999}) d'une application informatique, les auteurs de la notation UML préconisent d'utiliser une démarche:

- itérative et incrémentale,
- guidée par les besoins des utilisateurs du système,
- centrée sur l'architecture logicielle.

La notation UML n'est pas à l'origine des concepts objet, mais il en donne une définition plus formelle et apporte la dimension méthodologique qui faisait défaut à l'approche objet. La définition et l'introduction du langage OCL (Object Constraint Language) au sein de la notation UML est un apport conséquent qui permet de définir des contraintes durant la conception qui peuvent être des propriétés de sécurité (Safety) ou de bon fonctionnement (Liveness).

Comme la notation UML n'impose pas de méthode de travail particulière, elle peut être intégrée à n'importe quel processus de développement logiciel et ceci de manière transparente. Par conséquent, elle doit être vue comme une boîte à outils, qui permet d'améliorer progressivement vos méthodes de travail, tout en préservant vos modes de fonctionnement. La popularité de la notation UML est renforcée par un nombre important d'outils, les Ateliers de Génie Logiciel (AGL), qui permettent de mettre en oeuvre la notation, sous forme graphique et qui accompagne le processus de développement en permettant la génération partielle de code, la documentation et la rétro-ingénérie.

En plus d'être soutenue par l'ensemble des acteurs de l'informatique, la notation UML est aujourd'hui utilisée dans tous les domaines et depuis peu elle a été introduite dans le domaine des applications dites critiques, ce qui n'est pas sans soulever certaines questions.

Processus de modélisation

Dans le cadre de ce projet, nous cherchons à définir une méthodologie qui permet d'acquérir les besoins dans le cadre de la conception d'une application ferroviaire. Les applications ferroviaires sont soumises à de fortes contraintes de sécurité. C'est pourquoi nous proposons un processus de modélisation qui est orienté « *prise en compte de la sécurité* ».

Pour modéliser le système, nous proposons donc un processus d'acquisition découpé en quatre phases:

- identification des constituants (acteurs) de l'environnement du système et de leurs interactions (fonctionnement nominal et défaillances) avec celui ci;
- identification des effets des interactions (graphe de séquence) entre l'environnement et le système;
- identification de l'architecture du système (graphe de classe);
- identification du comportement du système et des sous-systèmes(graphe état/transition).

Description du modèle semi-formel

Notre système se décompose en trois sous-systèmes de contrôle. L'un sera embarqué dans le train, le second contrôlera le passage à niveau et le dernier est lié à la supervision du système. La supervision permet à un *poste de contrôle commande* (noté PCC) d'en gérer l'exploitation et la régulation ainsi que ses pannes.

Constituants de l'environnement

De la description de l'étude de cas, nous pouvons extraire une première liste des composants de l'environnement : train, rail, lampe, barrière, signal sonore, conducteur, réseau de communication, opérateur du *PCC*, agent de maintenance.

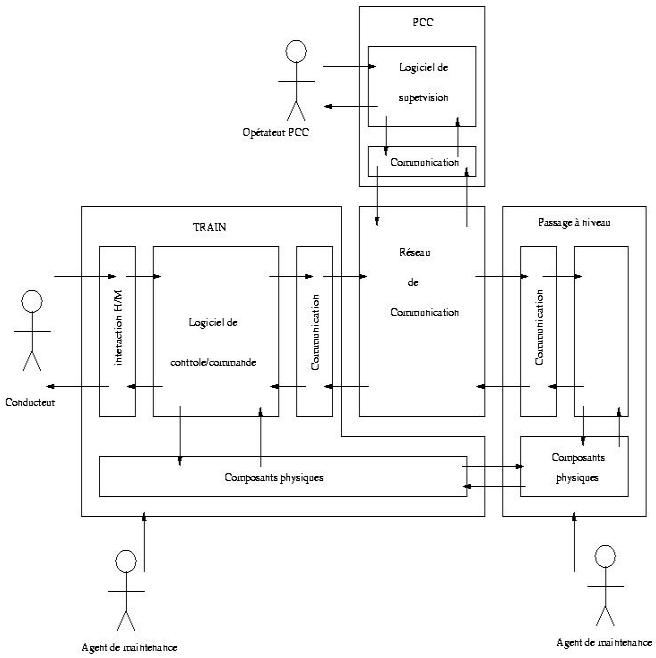


Figure 2 : Passage à niveau pour une voie ferrée

La figure précédente présente une première architecture du système et son environnement. Les sous-systèmes nommés *PCC* et *réseau de communication* n'intervenant pas dans la sécurité du système, nous avons choisi de ne pas les modéliser dans la suite de cette étude de cas.

[1] http://www.omg.org

Effet des interactions

Dans cette phase, nous cherchons à décrire les interactions fonctionnelles entre les composants de l'environnement et le système étudié.

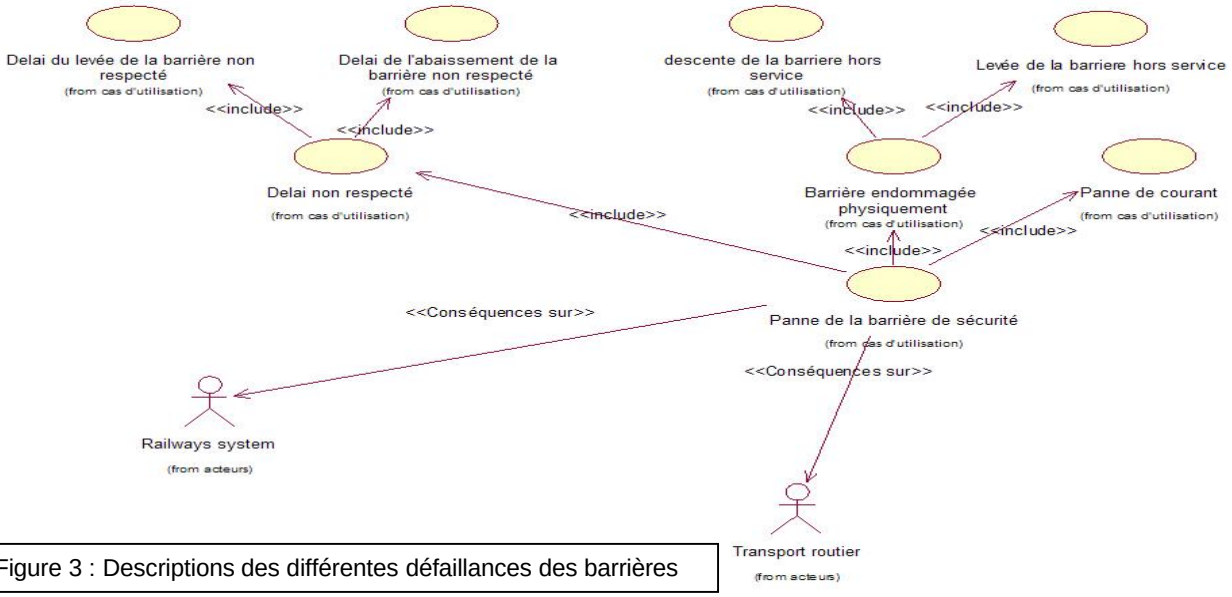


Figure 3 : Descriptions des différentes défaillances des barrières

Dans les applications dites *critiques de fonctionnement* comme notre étude de cas, il faut tenir compte, dans la conception du système, de l'interaction avec l'environnement. En effet, une source très répandue d'échec de fonctionnement est le dysfonctionnement des capteurs. Des échecs peuvent également être engendrés par une défaillance de la communication entre les deux systèmes mais aussi par une dégradation des installations (barrières qui ne bougent plus, des feux en panne,...etc).

La figure 3 introduit un cas d'utilisation qui présente les différents types de défaillance des barrières, les trois événements initiateurs sont l'absence de courant, la panne franche (barrière hors service) et un délai d'abaissement trop long.

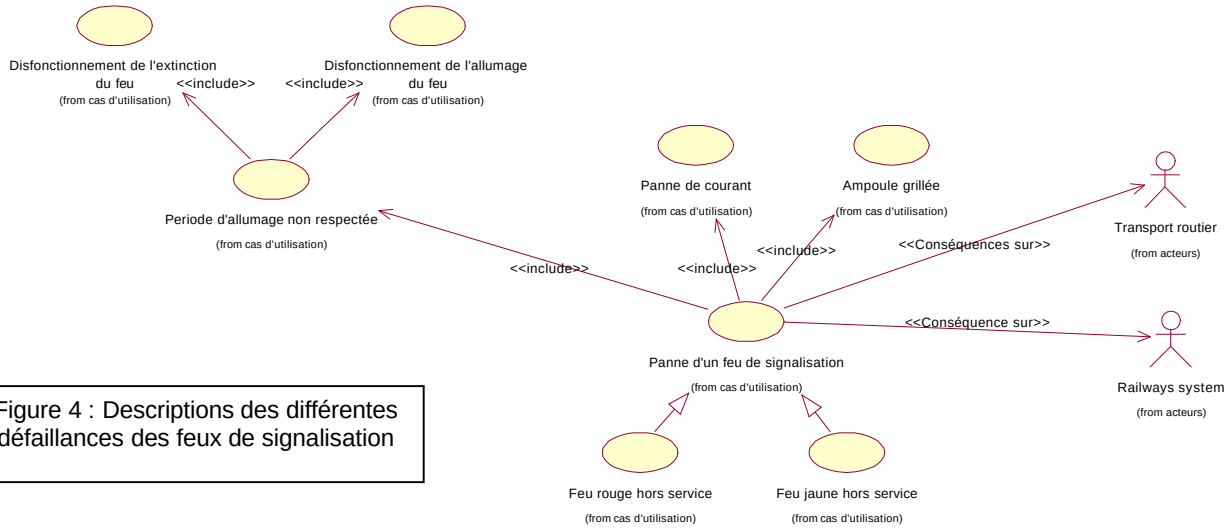


Figure 4 : Descriptions des différentes défaillances des feux de signalisation

On peut noter que pour les barrières, l'unique manière de détecter une défaillance est le dépassement de délai de fermeture ou d'ouverture. Ces défaillances peuvent apparaître à n'importe quel moment, mais la mise en place d'une réparation ne peut être lancée que si le passage à niveau est dégagé (pas de train dans le passage à niveau). Les interactions entre l'environnement et le système seront donc modélisées au travers de cas d'utilisation décrivant les besoins fonctionnels et de graphe de séquence décrivant les impacts de l'environnement sur le système.

Pour décrire les besoins fonctionnels, nous utilisons les cas d'utilisations qui permettent de décrire les actions pouvant être réalisées par les acteurs sur le système. Il faut rappeler que les acteurs (signaux, feux, barrières, sous-système de communication, ...) peuvent agir (ou ré-agir) au travers de bons et de mauvais comportements (panne, défaillance).

Le cahier des charges initial ([6]) insiste très fortement sur l'aspect défaillance comme le montre les exigences suivantes :
In the case study only a limited number of failures are regarded:

1. *failures of the yellow traffic light*
2. *failures of the red traffic light*
3. *failures of the barriers,*
4. *failures of the vehicle sensor and*
5. *the delay or loss of telegrams on the radio network.*

La figure 4 présente les défaillances liées aux feux régulant le traffic routier.

Chaque cas d'utilisation est caractérisé par un scénario textuel ou par un graphe de séquence.

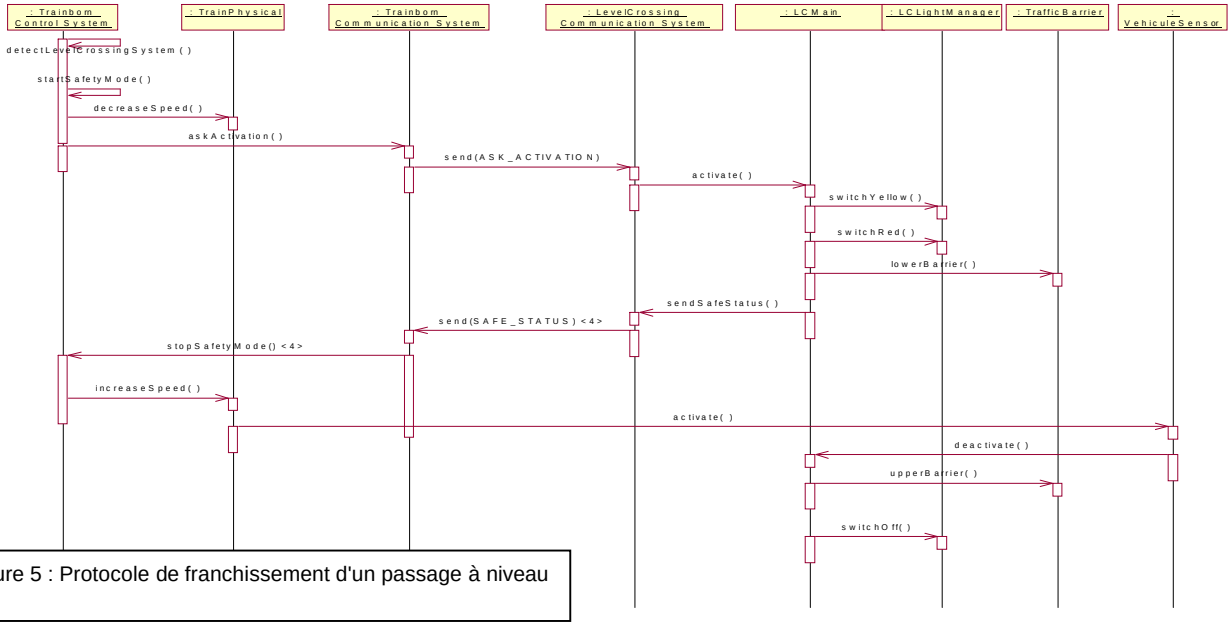


Figure 5 : Protocole de franchissement d'un passage à niveau

Nous utilisons les graphes de séquence pour décrire les impacts de l'environnement sur le système. Le graphe de séquence de la figure 5 décrit le fonctionnement de base du protocole de franchissement du passage à niveau par un train.

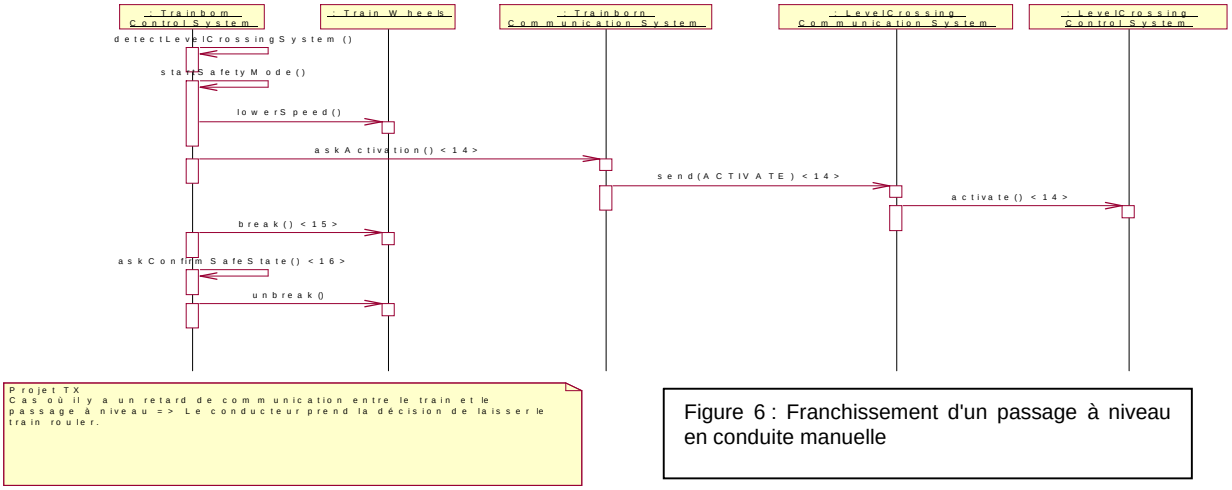


Figure 6 : Franchissement d'un passage à niveau en conduite manuelle

Tout d'abord, les feux s'allument afin d'arrêter le flux routier. Dans le cas où la zone ne serait pas sensible au bruit, des sonneries tinteront au moment de l'allumage des feux routiers. Après un délai de préavis, les barrières s'abaissent. Si l'abaissement des barrières se passe sans encombres (c'est-à-dire que l'abaissement s'est fait en respectant les limites temporelles maximales d'abaissement), le système est dit en « safe-mode » et le train aura le droit de passer le passage à niveau.

La fin de passage du train impliquera le relèvement des barrières et l'extinction des feux routiers et éventuellement l'arrêt des sonneries.

Il est important de noter que le système de contrôle peut passer à tout moment en mode manuel. Ce mode correspond à la prise en main humaine (gardien du passage à niveau) de la gestion de la sûreté du passage à niveau, et peut être déclenché pour cause de dysfonctionnement matériel (des barrières qui ne s'abaissent pas, des feux qui ne s'allument pas,...etc).

Si le train ne reçoit pas de confirmation du passage à niveau qu'il est dans un état *sûr*, deux comportements sont possibles:

1. Le train freine et s'arrête ;
2. Le conducteur décide de prendre la main (voir figure 6 et d'effectuer une *conduite à vue*.

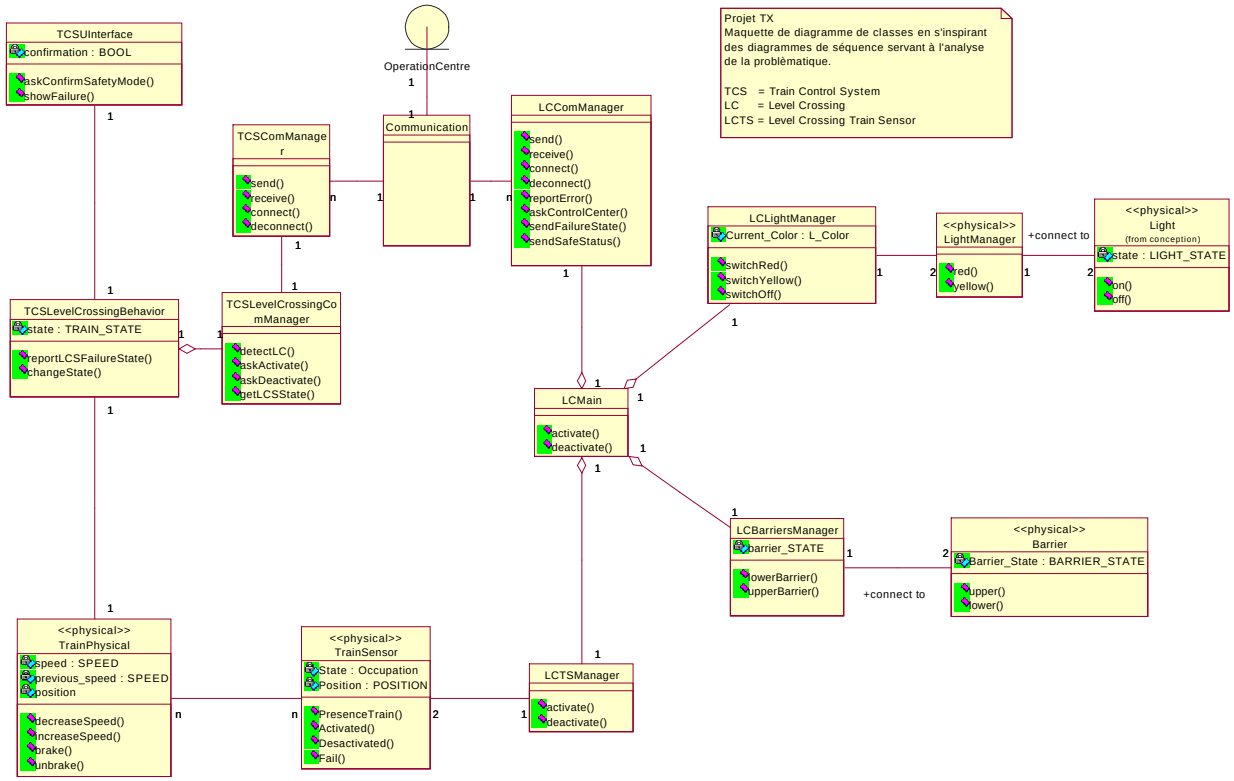
Architecture

La figure 7 présente une architecture du système et son environnement. Cette architecture est construite à partir de l'analyse des interactions entre les composants du système.

Le graphe de classe fait apparaître les acteurs physiques (TrainPhysical, TrainSensor, Light, Barrier) et les interactions entre eux. À titre d'exemple, on voit apparaître une relation entre la classe TrainPhysical et la classe TrainSensor. Cette relation signifie que des éléments physiques sont en interaction. Nous remarquerons que la classe TrainSensor propose une méthode fail qui introduit le fait que le capteur peut tomber en panne.

Comportement

Nous allons maintenant décrire, à l'aide de diagramme d'état, le système de contrôle et le système embarqué.



Les barrières, les feux routiers et le signal sonore seront gérés par le système de contrôle du passage à niveau. Le sous-système de contrôle du passage à niveau sera activé dès le déclenchement des informations d'annonce. À l'activation du sous-système de contrôle, une séquence d'action ordonnée (Figure 8) sera effectuée par ce système afin de vider le passage à niveau à temps et de garantir sa fermeture à tout traffic routier.

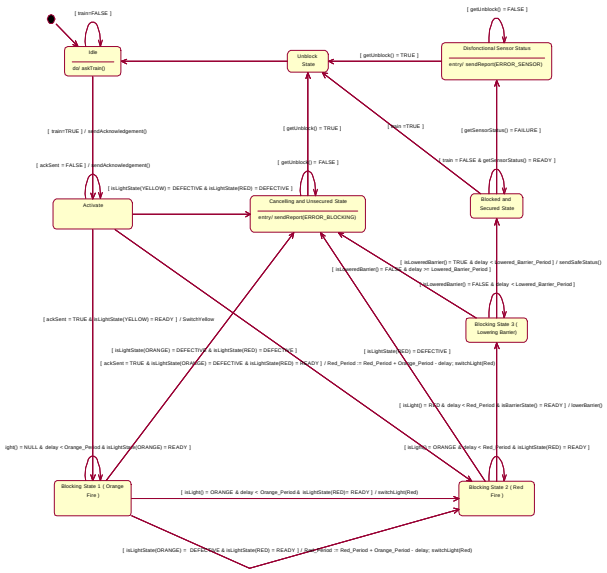


Figure 8 : Comportement du logiciel de contrôle commande.

Le système embarqué exécute une série d'actions à l'approche d'un passage à niveau (figure 9). Lorsque le train passe par l'origine d'annonce du passage à niveau, il demande au système de contrôle du passage à niveau un acquittement pour savoir s'il a commencé à abaisser les barrières ou pas. Après réception de l'acquittement (ack), le système embarqué dans le train se met en attente en commençant le freinage afin de laisser du temps pour la fermeture des barrières. Après ce délai d'attente, le système de contrôle du passage à niveau envoie son état au système embarqué. Si le passage à niveau est en « safe-mode », le contrôle embarqué annule le freinage et reprend sa vitesse initiale. Un capteur de « fin-passage » détecte la sortie du train du passage à niveau, et déclenche le relèvement des barrières et l'extinction des feux.

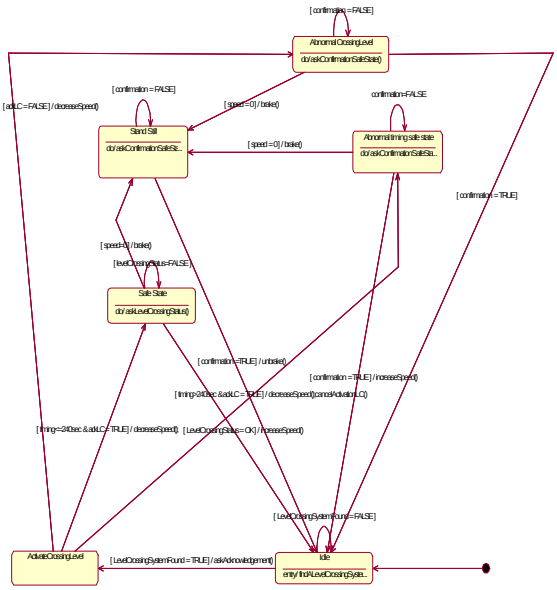


Figure 9 : Comportement du logiciel embarqué

Nous avons donc maintenant une modélisation de notre étude de cas que nous allons pouvoir traduire en B.

Génération des applications: Modélisation formelle

Présentation de la méthode B

En ce qui concerne les origines théoriques, la méthode B ([1]) est née de la rencontre dans les années 1980 et au sein du *Programming Research Group*, des créateurs respectifs des méthodes *Z* et *VDM*. En conséquence, les racines théoriques de ces méthodes sont semblables, cependant Z et VDM restent limitées à la spécification (description) du logiciel alors que la méthode B s'efforce d'unifier les aspects description et réalisation dans un même formalisme.

La méthode B permet de générer un code exécutable sécurisé et prouvé à partir d'expressions mathématiques formelles. Ces expressions mathématiques sont rassemblées dans un composant B appelé « machine ». Elles sont ensuite raffinées par un ou plusieurs composants B appelés raffinements. Après tous les raffinements successifs, nous obtenons le dernier composant dit « implementation ». L'ensemble de ces composants est appelé « module ». Un module peut dépendre de plusieurs autres modules mais nous ne présenterons pas les principes et les différents types de dépendance dans cet article.

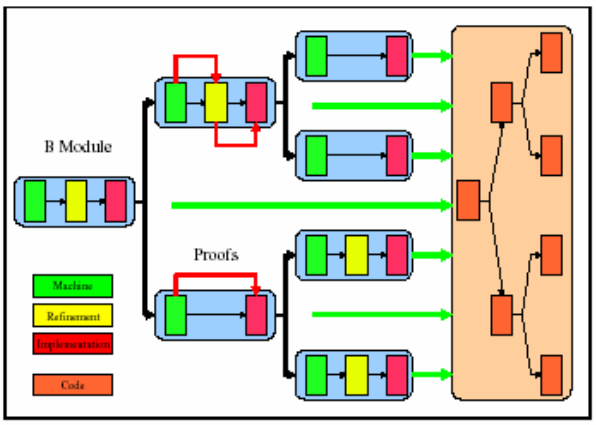


Figure 10 : Développement B

Un code généré par la méthode B est à priori sécurisé et prouvé à l'aide d'un outil de preuve résolvant actuellement 75% des obligations de preuves générales. Pour conclure cette courte présentation, on peut affirmer que la méthode B fait partie des méthodes formelles qui semblent les mieux acceptées dans le cadre industriel du développement des logiciels critiques appliqués au ferroviaire (voir par exemple [2], [5]). En effet, la méthode B permet de construire effectivement un pont entre la modélisation mathématique et sa réalisation informatique.

Processus de génération des modèles B

Il existe plusieurs travaux sur la génération de code B à partir de modèle UML, on peut citer les travaux ([8] et [7]) de l'équipe dédale[2] et ceux ([11]) de l'équipe sial/arlog[3]. Les travaux menés dans le cadre du projet **B-RAIL** ne cherche pas à décrire un nouveau processus de génération de code, nous voulons proposer une méthodologie globale qui cherche à acquérir au mieux le besoin et qui s'appuie sur des techniques de génération de modèle B. Le problème actuel est d'arriver à fédérer les différents travaux menés sur ce sujet. Nous avons proposé un premier processus ([3]) de génération de code B à partir de modèle UML qui se base sur le graphe de classe et sur les graphes états/transitions. Cependant, le processus de génération de code B à partir de modèle UML n'est pour l'instant ni outillé ni finalisé.

[2] Pour plus d'information sur l'équipe DEDALE: http://www.loria.fr/equipes/dedale/home.html

[3] Pour plus d'information sur l'équipe ARLOG: http://www.prism.uvsq.fr/recherche/themes/sial/arlog

Description des modèles formels

Cette section a pour but de présenter le modèle B obtenu à partir de la traduction du modèle UML. La machine abstraite de la figure 11, intitulée *TCS_LevelCrossingBehavior_0*, définit tous les états dans un ensemble nommé STATE et introduit une opération nommée *change_state* qui décrit les changements d'état du sous-système gérant le passage à niveau. Ce composant n'est pas déterministe car il utilise la substitution *list_var:(predicate)* pour décrire l'opération *change_state*. Cette substitution indique que l'ensemble des variables devient tel que le prédicat soit vrai.

```
MACHINE
        TCS_LevelCrossingBehavior_0
SETS
STATES ={Idle,SafeState,ActivateCrossingLevel,StandStill,
         AbnormalTimingSafeState, AbnormalCrossingLevel}
CONCRETE_VARIABLES
        current_state
INVARIANT
        current_state : STATES
INITIALISATION
        current_state := Idle

OPERATIONS
change_state =
 BEGIN
  current_state :(
     current_state :STATES
  &((current_state$0=Idle)
      =>current_state:{ActivateCrossingLevel,Idle})
 ...
  &((current_state$0= StandStill)
    => current_state/:{SafeState, AbnormalTimingSafeState})
  )
 END
END
```

Figure 11: Machine : TCS_LevelCrossingBehavior_0

La figure suivante, est un extrait de l'implantation *TCS_LevelCrossingBehavior_n*. Cette implantation est fidèle au graphe état/transition de la figure 8.

```
IMPLEMENTATION
        TCS_LevelCrossingBehavior_n
REFINES
        TCS_LevelCrossingBehavior_0
INVARIANT
        (current_state = StandStill) => (brake = TRUE)
 &      (current_state /= StandStill) => (brake = FALSE)
INITIALISATION
        current_state := Idle

OPERATIONS
change_state =
CASE current_state OF
        EITHER  Idle THEN
        VAR bb IN
            bb <-- detectLevelCrossingSystem;
            IF     ( bb = TRUE )
            THEN
            BEGIN
                current_state := ActivateCrossingLevel
            ;       AskAcknowledgement
            END
            ELSIF   ( bb = FALSE)
            THEN current_state := Idle
            END
        END
....
END

END
```

Figure 12: Implémentation : TCS_LevelCrossingBehavior_n

Conclusion

Le projet B-RAIL tente de définir une méthodologie pour le développement de systèmes ferroviaires. Ces systèmes sont soumis à des contraintes de sûreté de fonctionnement très forte comme le montre la norme EN 50128 qui régit le développement des logiciels ferroviaires. La méthodologie proposée s'appuie sur l'utilisation de la notation UML et de la méthode B. La notation UML est utilisée pour acquérir le besoin et décrire le système. Cette notation UML est essentiellement graphique ce qui facilite la compréhension des modèles et la discussion entre les acteurs du projet (client, développeur, valideur, ..). La méthode B est un langage formel qui permet l'expression de propriété et qui au travers d'une activité de preuve permet de détecter les incohérences.
Nous avons pu montrer sur cet exemple la faisabilité et l'intérêt de cette approche. Il nous reste à affiner le processus de génération de modèle B à partir de modèle UML. Cet affinement doit passer par une analyse fine des différents processus de génération qui ont déjà été proposés.

Un axe de travail en cours de réalisation est lié à l'expression de contrainte OCL au sein de notre modèle UML. Actuellement, il existe des contraintes textuelles sous forme de *texte libre*.
L'introduction de contraintes OCL permettra d'utiliser les travaux de R. Marcano ([11]).

Dans le cadre du passage de la notation UML vers un modèle B, deux objectifs différents mais complémentaires peuvent être atteints au travers de deux types de modèles:

1. Un modèle B permettant de générer le code de l'application;
2. Un modèle B permettant de vérifier et de valider le comportement de l'application face au exigences de sécurité exprimée dans le cadre du modèle UML.

Ces deux modèles seront distincts mais complémentaires.

Références

[1] Abrial (Jean-Raymond). ”The B Book - Assigning Programs to Meanings”. Cambridge University Press, août 1996.
[2] Behm (Patrick), Benoit (Paul), Faivre (Alain) et Meynadier (Jean-Marc). “METEOR : A successful application of B in a large project”. Proceedings of FM’99: World Congress on Formal Methods, pp. 369–387.
[3] Bon (Philippe), Boulanger (Jean-Louis) et Mariano (Georges). “Semi formal modelling and formal specification: UML & B in simple railway application”. ICSSEA 2003,December 2-4 2003.
[4] Booch (Grady), Rumbaugh (James) et Jacobson (Ivar). “The Unified Software Development Process”. Addison-Wesley, 1999.
[5] Dehbonei (Babak) et Mejia (Fernando). “Formal development of software in railways safety critical systems”. Railway Operations, éd. par T.K.S. Murthy (B.), Mellitt (C.A.), Brebbia (G.) et Sciutto (S.). COMPRAIL94, pp. 213–219. – Computational Mechanics Publications, 1994.
[6] Jansen (L.) et Schneider (Eckehard). “Traffic Control Systems Case Study : Problem Description and a Note on Domain-Based Software Specification”. Rapport technique, Institute of Control and Automation Engineering, Technical UNIVERSITY of Braunschweig, 2000.
[7] Ledang (Hung) et Souquières (Jeanine). « Contributions for modelling UML statecharts in B”. n IFM 2002: Third International Conference on Integrated Formal Methods.
[8] Ledang (Hung). « Des cas d’utilisation à une spécification B ». In AFADL’2001.
[9] Ledang (Hung). “Automatic translation from UML specifications to B”. In RCS’02 : International Workshop on Refinement of Critical Systems : Methods, Tools and Experience.
[10] Marcano (Rafael), Meyer (Eric), Levy (Nicole) et Souquières (Jeanine). « Utilisation de patterns dans la construction de specifications en UML et B ». In AFADL’2000.
[11] Marcano (Rafael) et Levy. (Nicole). « Transformation d’annotations OCL en expressions B ». In Journées AFADL’2001.
[12] Muller (Pierre-Alain). « Modélisation objet avec UML. » Editions Eyrolles, 2001.
[13] « Unified Modelling Language version 1.4 ». Rapport technique, OMG, 2001.
[14] Rétiveau (Roger). « La signalisation ferroviaire ». Presses de l’école Nationale des Ponts et Chaussées, 1987