Construction de spécifications formelles abstraites dirigée par les buts

Abderrahman Matoussi

UNIVERSITÉ PARIS-EST

ÉCOLE DOCTORALE

MSTIC : Mathématiques et Sciences et Technologies de l'Information et de la Communication

THESE

présentée pour obtenir le grade de :

Docteur de l'Université Paris-Est,

spécialité Informatique

par :

Abderrahman MATOUSSI

Titre de la thèse :

Construction de spécifications formelles abstraites dirigée par les buts

soutenue publiquement le 9 décembre 2011 devant le jury suivant :

Président du jury :		
	Camille Salinesi	Professeur à l'Université Paris 1 Panthéon-Sorbonne
Rapporteurs :		
	Jean-Pierre Giraudin	Professeur à l'Université de Grenoble
	Vincent Poirriez	Professeur à l'Université de Valenciennes
Examinateurs :		
	Hubert Dubois	Chercheur au CEA-LIST de Saclay
Directeurs de thèse :		
	Régine Laleau	Professeur à l'Université Paris-Est Créteil
	Frédéric Gervais	Maître de conférences à l'Université Paris-Est Créteil

Remerciements

Je remercie tout d'abord Jean-Pierre Giraudin et Vincent Poirriez pour avoir accepté l'évaluation de cette thèse en tant que rapporteurs. Je tiens à remercier également Camille Salinesi pour l'honneur qu'il me fait en présidant le jury de cette thèse et Hubert Dubois pour avoir accepté d'en être examinateur.

Je tiens à exprimer toute ma gratitude à Régine Laleau, à la fois directrice du laboratoire LACL et ma directrice de thèse, pour la confiance qu'elle m'a témoignée et pour sa haute qualité de l'encadrement tout au long de ces années. Ses conseils, sa bonne humeur et ses qualités humaines ont largement contribué à l'aboutissement de cette thèse. Je tiens à lui exprimer toute ma reconnaissance et tout mon respect.

Je remercie également Frédéric Gervais, mon co-encadrant de thèse, qui m'a été d'une aide inestimable avec ses conseils avisés. Je tiens à lui exprimer toute ma gratitude.

Je tiens aussi à remercier tous mes collègues du laboratoire LACL et de l'IUT Sénart-Fontainebleau.

Enfin, je rends hommage à toute ma famille et à tous mes amis qui m'ont toujours apporté leur soutien. Cette thèse leur est dédiée.

Abderrahman

Résumé

Avec la plupart des méthodes formelles, un premier modèle peut être raffiné formellement en plusieurs étapes, jusqu'à ce que le raffinement final contienne assez de détails pour une implémentation. Ce premier modèle est généralement construit à partir de la description des besoins obtenue dans la phase d'analyse des exigences. Cette transition de la phase des exigences à la phase de spécification formelle est l'une des étapes les plus délicates dans la chaîne de développement formel. En fait, la construction de ce modèle initial exige un niveau élevé de compétence et beaucoup de pratique, d'autant qu'il n'existe pas de processus bien défini pour aider les concepteurs. Parallèlement à ce problème, il s'avère également que les exigences non-fonctionnelles sont largement marginalisées dans le processus de développement logiciel. Les pratiques industrielles actuelles consistent généralement à spécifier seulement les exigences fonctionnelles durant les premières phases de ce processus et à laisser la prise en compte des exigences non-fonctionnelles au niveau de l'implémentation.

Pour surmonter ces problèmes, la thèse vise à définir un couplage entre un modèle d'exigences exprimé en SysML/KAOS et des spécifications formelles abstraites, tout en garantissant une distinction entre les exigences fonctionnelles et non-fonctionnelles dès la phase d'analyse des exigences. Pour cela, la thèse propose tout d'abord deux approches différentes (l'une dédiée au B classique et l'autre à Event-B) dans lesquelles des modèles formels abstraits sont construits progressivement à partir du modèle de buts fonctionnels SysML/KAOS. La thèse se focalise par la suite sur l'approche dédiée à Event-B afin de la compléter et l'enrichir en se servant de deux autres modèles SysML/KAOS qui décrivent les buts non-fonctionnels et leurs impacts sur les buts fonctionnels. Nous présentons différentes manières permettant d'injecter ces buts non-fonctionnels et leurs impacts dans les modèles abstraits Event-B déjà obtenus. Des liens de correspondance entre les buts non-fonctionnels et les différents éléments Event-B sont également établis afin de faciliter la gestion de l'évolution de ces buts.

Les différentes approches proposées dans cette thèse ont été appliquées pour la spécification du composant de localisation qui est une partie critique d'un système de transport terrestre. L'approche dédiée à Event-B est implémentée dans l'outil SysKAOS2EventB, permettant ainsi de générer une architecture de raffinement Event-B à partir d'un modèle de buts fonctionnels SysML/KAOS. Cette mise en œuvre s'appuie principalement sur les technologies de transformation de modèles à modèles.

MOTS-CLÉS : Ingénierie des exigences, Spécification formelle, SysML/-KAOS, B, Event-B, Raffinement, But non-fonctionnel, Transformation de modèles à modèles.

Abstract

With most of formal methods, an initial formal model can be refined in multiple steps, until the final refinement contains enough details for an implementation. Most of the time, this initial model is built from the description obtained by the requirements analysis. Unfortunately, this transition from the requirements phase to the formal specification phase is one of the most painful steps in the formal development chain. In fact, building this initial model requires a high level of competence and a lot of practice, especially as there is no well-defined process to assist designers. Parallel to this problem, it appears that non-functional requirements are largely marginalized in the software development process. The current industrial practices consist generally in specifying only functional requirements during the first levels of this process and in leaving the consideration of non-functional requirements in the implementation level.

To overcome these problems, this thesis aims to define a coupling between a requirement model expressed in SysML/KAOS and an abstract formal specification, while ensuring a distinction between functional and non-functional requirements from the requirements analysis phase. For that purpose, this thesis proposes firstly two different approaches (one dedicated to the classical B and the other to Event-B) in which abstract formal models are built incrementally from the SysML/KAOS functional goal model. Afterwards, the thesis focuses on the approach dedicated to Event-B in order to complete it and enrich it by using the two other SysML/KAOS models describing the non-functional goals and their impact on functional goals. We present different ways to inject these non-functional goals and their impact into the obtained abstract Event-B models. Links of correspondance between the non-functional goals and the different Event-B elements are also defined in order to improve the management of the evolution of these goals.

The different approaches proposed in this thesis have been applied to the specification of a localization component which is a critical part of a land transportation system. The approach dedicated to Event-B is implemented in the SysKAOS2EventB tool, allowing hence the generation of an Event-B refinement architecture from a SysML/KAOS functional goal model. This implementation is mainly based on the model-to-model transformation technologies.

Keywords : Requirements engineering, Formal specification, SysML/-KAOS, classical B, Event-B, Refinement, Non-functional goal, Model-to-model transformation.

Table des matières

Introduction Générale

Contexte

La transition difficile de la phase d'analyse à la phase de spécification

L'emploi des méthodes formelles pour la spécification de systèmes complexes est de plus en plus fréquent. Ces méthodes ont montré leur capacité à produire des systèmes critiques pour de grands projets industriels tels que : la ligne 14 du métro parisien [Behm *et al.* 1999] ou la navette Roissy VAL [Badeau & Amelot 2005] qui ont utilisé la méthode B [Abrial 1996a]. Dans la plupart des méthodes formelles, un modèle mathématique initial peut être raffiné de manière formelle en plusieurs étapes jusqu'à ce que le raffinement final contienne assez de détails pour une implémentation. Ce modèle, représentation abstraite du futur système, est généralement issu d'une description des besoins obtenue lors de la phase d'analyse des exigences. Les concepteurs se servent beaucoup de leur intuition, leur « sixième sens » et leur expérience pour la construction de ce modèle, d'autant plus qu'il n'y a pas de processus bien défini pour les guider. C'est pourquoi la transition de la phase d'analyse à la phase de spécification est maintenant reconnue comme étant la principale faiblesse dans la chaîne de développement formel [Abrial 2003, Abrial 2006]. Ce maillon faible complique par la suite la tâche de validation des modèles formels dans la mesure où il devient très difficile d'assurer la correspondance entre les exigences et les spécifications formelles initiales. En effet, d'un côté les usagers qui ont exprimé leurs exigences éprouvent des difficultés à comprendre les modèles formels et d'un autre côté les concepteurs des modèles formels se trouvent parfois dans l'impossibilité de faire le lien avec les exigences initiales. Par conséquent, le fossé entre la phase d'analyse des exigences et la phase de spécification est en train de s'élargir et la réconciliation paraît de plus en plus difficile.

Le problème s'aggrave pour un système complexe incluant un grand nombre d'exigences et une documentation souvent trop verbeuse. Dans une telle situation, les concepteurs essaient de faire directement une sorte de pseudo-programmation au lieu de construire un modèle formel initial. Pour faire face à ce problème, l'idée encouragée par de nombreux chercheurs comme JR Abrial [Abrial 2010] est de réécrire complètement les documents d'exigences avant de commencer toute activité de spécification. Dans ce contexte, l'application des méthodes de l'ingénierie des exigences dès le début du processus de conception et avant même d'utiliser les méthodes formelles peut être intéressante car ces méthodes permettent la structuration et la documentation de l'ensemble des exigences. Parmi ces méthodes, l'ingénierie des exigences dirigée par les buts [Nuseibeh & Easterbrook 2000, van Lamsweerde 2009], notée GORE (acronyme de « *Goal-Oriented Requirements Engineering* »), permet la structuration des buts d'un système et la représentation

des frontières du système. L'avantage des méthodes GORE est qu'elles sont très proches de la façon dont les humains pensent et sont aussi faciles à comprendre par toutes les parties prenantes [Brandozzi & Perry 2001]. En effet, les buts jouent un rôle important dans l'ingénierie des exigences en fournissant un pont qui peut lier les besoins des clients à la spécification du système [Robinson & Pawlowski 1998].

La prise en compte des exigences non-fonctionnelles

Dans les méthodes d'ingénierie des exigences, deux types d'exigences peuvent être identifiés [Glinz 2007]. Les exigences fonctionnelles spécifient les fonctions que le système à concevoir doit être en mesure d'effectuer. Les exigences non-fonctionnelles quant à elles représentent les propriétés ou les contraintes sous lesquelles le système à concevoir doit fonctionner telles que les aspects de performance, de qualité ou de sécurité. Les pratiques industrielles actuelles consistent à spécifier les exigences fonctionnelles dès les premières phases du processus de développement logiciel et à laisser la prise en compte des exigences non-fonctionnelles au niveau de l'implémentation [Tonu 2006, Kassab *et al.* 2007]. Ce choix est souvent justifié par l'énorme pression (en termes de temps) pour le déploiement rapide du logiciel [Cysneiros & Kushniruk 2003]. Pourtant, la prise en compte des exigences non-fonctionnelles dès les phases initiales du processus de développement augmente les chances de succès des systèmes logiciels. Selon [Ebert 1997], les conséquences de négliger les exigences non-fonctionnelles sont souvent plus dramatiques que l'omission des exigences fonctionnelles. Dans ce contexte, [Ebert 1997, Neto *et al.* 2000] énumèrent quelques problèmes réels bien connus dûs à l'omission des exigences non-fonctionnelles durant les premières phases du processus de développement. Parmi ces problèmes, nous pouvons citer le fiasco du Service Ambulancier de Londres (LAS) en octobre 1992 [Finkelstein & Dowell 1996]. Leur nouveau logiciel de délégation automatique d'ambulances est tombé en panne après à peine trois jours d'utilisation. En effet, ce logiciel a retardé l'arrivée des ambulances sur les scènes (jusqu'à 11 heures de retard), causant ainsi la mort d'une trentaine de personnes. Le nouveau système a été arrêté et l'ancien remis en service. Parmi les causes de ce drame, nous pouvons trouver notamment une omission de nombreuses exigences non-fonctionnelles au cours du développement du logiciel telles que l'utilisabilité (une mauvaise maîtrise de l'information sur l'écran).

Dans l'étude bibliographique [Matoussi & Laleau 2008] sur la prise en compte des exigences non-fonctionnelles dans le processus de développement logiciel, nous avons identifié un certain nombre de limites et en particulier : (i) la rareté des travaux qui considèrent les exigences non-fonctionnelles depuis la phase d'analyse jusqu'à l'implémentation ; (ii) la majorité des travaux qui ont pris en compte les exigences non-fonctionnelles au niveau analyse et spécification s'appuient seulement sur des techniques informelles et semi-formelles. Ces limites peuvent être justifiées par le fait que l'analyse et la spécification des exigences non-fonctionnelles n'est

pas toujours évidente puisque généralement il est difficile de les exprimer d'une façon objective et mesurable et elles peuvent être en conflit les unes avec les autres. Par ailleurs, les exigences non-fonctionnelles sont plus difficiles à traiter que les fonctionnelles parce que leur impact n'est pas généralement localisé sur une partie du système, mais s'étend sur l'ensemble du système [Aurum & Wohlin 2005]. Les exigences non-fonctionnelles peuvent donc avoir un impact sur plusieurs exigences fonctionnelles.

Objectifs de la thèse

L'objectif de la thèse est de faire face aux problèmes ci-dessus. Les travaux de cette thèse s'insèrent dans le cadre du projet TACOS [1], qui vise à construire une approche par composants pour la spécification de systèmes sûrs, depuis l'expression des exigences jusqu'à une spécification formelle, en utilisant ou adaptant des langages et des outils existants. Le domaine d'application choisi est celui du transport terrestre, et plus précisément, les nouveaux types de véhicules en libre-service qui seront utilisés dans les villes de demain, appelés Cycabs. Les systèmes de ce domaine, à la fois distribués et embarqués, nécessitent l'expression des exigences fonctionnelles et non-fonctionnelles.

La thèse vise à proposer une méthode complète d'analyse et de spécification qui permet de formaliser les exigences fonctionnelles et non-fonctionnelles, en garantissant une distinction entre ces deux types d'exigences dès la phase d'analyse des exigences. La thèse vise également à développer un logiciel libre construit à partir des logiciels libres disponibles. L'objectif est que cet outil puisse être utilisé comme composant d'un outil plus général d'analyse-conception-développement de systèmes. En effet, à notre connaissance, très peu d'outils libres existent dans le domaine de l'analyse des exigences, et encore moins d'outils qui permettent d'exprimer les aspects formels de cette phase.

Apports de la thèse

La thèse étudie les travaux qui se servent de l'approche GORE pour assurer la correspondance entre la phase d'analyse des exigences et la phase de spécification. Cette étude a révélé que ces travaux ne considèrent pas toutes les parties du modèle de buts GORE, mais seulement les buts feuilles de la hiérarchie de buts. Par conséquent, le modèle formel n'inclut aucune information sur les buts abstraits et surtout sur le type de raffinement de buts. De plus, ces travaux ne définissent pas de frontières entre les exigences fonctionnelles et non-fonctionnelles. De là, notre idée est de remonter les méthodes formelles jusqu'à la phase d'analyse des exigences, c'est-à-dire inclure la phase d'analyse des exigences dans le processus de développement formel tout en restant au même niveau d'abstraction. Le choix s'est

1. ANR-06-SETIN-017. http://tacos.loria.fr

porté sur la méthode SysML/KAOS [Laleau *et al.* 2010, Gnaho & Semmak 2011] pour spécifier le modèle de buts, parce qu'elle met l'accent sur la distinction entre les exigences fonctionnelles et non-fonctionnelles en définissant trois modèles. Le premier modèle décrit les buts fonctionnels, le second modèle décrit les buts non-fonctionnels et le dernier présente l'impact des buts non-fonctionnels sur les buts fonctionnels.

La thèse propose tout d'abord deux approches différentes (une dédiée au B classique [Abrial 1996a] et l'autre dédiée à Event-B [Abrial 2010]) qui se servent du premier modèle SysML/KAOS (décrivant les buts fonctionnels) afin de dériver des spécifications formelles abstraites. Il pourra alors être possible de prouver la contrepartie formelle (en B et en Event-B) du modèle de buts SysML/KAOS et d'établir des liens de correspondance entre les buts et les spécifications formelles abstraites. Dans l'approche dédiée à Event-B, la thèse propose également une extension de raffinement Event-B afin de permettre l'expression explicite de la notion d'ordonnancement, d'entrelacement et d'exclusivité entre les événements, sans l'utilisation des variables de contrôle. Cette extension permet de rendre les spécifications Event-B plus lisibles et d'éviter l'enchevêtrement avec les variables de contrôle.

La thèse se focalise par la suite sur l'approche dédiée à Event-B afin de la compléter en se servant de deux autres modèles SysML/KAOS (décrivant les buts non-fonctionnels et leurs impacts sur les buts fonctionnels). L'idée est d'étudier différentes solutions pour la prise en compte des buts non-fonctionnels et de leurs impacts dans les modèles Event-B abstraits, obtenus à partir du modèle de buts fonctionnels. Cette étude a montré que les buts non-fonctionnels et leurs impacts vont se traduire par différents éléments Event-B qui vont compléter et enrichir les modèles Event-B abstraits obtenus. Des liens de correspondance entre les buts non-fonctionnels et les différents éléments Event-B sont également établis afin de faciliter la gestion de l'évolution des exigences non-fonctionnelles.

Enfin, la thèse décrit l'outil développé SysKAOS2EventB, permettant de générer une architecture de raffinement Event-B à partir d'un modèle de buts fonctionnels SysML/KAOS. L'outil est développé sous la forme d'un plug-in sur la plateforme TOPCASED [2] en se basant sur les technologies de transformation de modèles-à-modèles. Le plug-in développé assure la liaison avec la plateforme support d'Event-B, RODIN [3]. Cette dernière permet ainsi de vérifier et valider le modèle Event-B (et par conséquent sa contrepartie semi-formelle SysML/KAOS) en utilisant l'ensemble des plug-in RODIN tels que le prouveur, le model-checker ou l'animateur.

Les différentes approches proposées dans cette thèse ont été appliquées pour la spécification du composant de localisation qui est une partie critique d'un système de transport terrestre.

2. www.topcased.org/
3. http://rodin.cs.ncl.ac.uk/

Structure de la thèse

Outre cette introduction, la thèse est organisée en trois parties.

– La première partie fixe le contexte du travail et décrit l'état de l'art. Le chapitre 1 présente l'ingénierie des exigences, en mettant l'accent sur l'approche GORE. Il présente également les méthodes formelles B et Event-B. Le chapitre 2 décrit l'état de l'art sur le couplage entre la phase d'analyse des exigences et la phase de spécification, en se focalisant sur les travaux qui se servent de l'approche GORE. Ce chapitre dresse également un bilan des différentes insuffisances constatées.

– La deuxième partie présente nos contributions afin de faciliter la transition de la phase d'analyse des exigences à la phase de spécification. Le chapitre 3 présente une première approche, dédiée au B classique, permettant de dériver une architecture d'un modèle B à partir d'un modèle de buts fonctionnels SysML/KAOS. Le chapitre 4 présente une autre approche, dédiée à Event-B, dans laquelle les modèles Event-B abstraits sont construits d'une manière incrémentale à partir des modèles de buts fonctionnels SysML/KAOS. Le chapitre 5 étend la seconde approche (dédiée à Event-B) en présentant les différentes manières de prendre en compte les buts non-fonctionnels et leurs impacts dans les modèles abstraits Event-B déjà obtenus.

– La troisième partie inclut le chapitre 6 qui présente d'une manière détaillée le développement du plug-in SysKAOS2EventB implémentant l'approche proposée dans le chapitre 4.

Nous terminons cette thèse par une conclusion générale et nous discutons des perspectives ouvertes par les travaux réalisés.

Publications issues de ce travail de thèse

Revues Internationales

1. Régine Laleau, Farida Semmak, Abderrhaman Matoussi, Dorian Petit, Ahmed Hammad and Bruno Tatibouet. « A First Attempt to Combine SysML Requirements Diagrams and B ». *Innovations in Systems and Software Engineering*, Volume 6, Numbers 1-2, pages 47-54, Mars 2010. Springer.

Revues Nationales

2. Aderrahman Matoussi et Régine Laleau. « Une approche pour la prise en compte des buts non-fonctionnels SysML/KAOS dans les spécifications abs-

traites Event-B ». *RSTI - Ingénierie des Systèmes d'Information (ISI)*. Article sélectionné pour le numéro spécial INFORSID'11/ISI, Version étendue en cours de soumission.

3. Aderrahman Matoussi et Régine Laleau. « Un outil de construction de spécifications abstraites Event-B dirigée par les buts ». *RSTI - Ingénierie des Systèmes d'Information (ISI)*. Volume 16, numéro 5/2011, pages 143-166. Septembre 2011. Hermes-Lavoisier.

Conférences internationales avec comité de sélection et actes

4. Abderrahman Matoussi, Frédéric Gervais and Régine Laleau. « A Goal-Based Approach to Guide the Design of an Abstract Event-B Specification ». *Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 139-148, Las Vegas, USA, Avril 2011. IEEE Computer Society.
5. Abderrahman Matoussi, Frédéric Gervais and Régine Laleau. « Specification of a Localization Component Driven by a Goal-Based Approach : Some Lessons We Learned ». *Proceedings of the 13th Brazilian Symposium on Formal Methods (SBMF'10)*, vol. 6527 de LNCS, pages 177-193, Natal, Brasil, November 2010. Springer.
6. Atif Mashkoor and Abderrahman Matoussi. « Towards Validation of Requirements Models ». *Proceedings of the 2nd International Conference on Abstract State Machines, Alloy, B and Z (ABZ'10)*, vol. 5977 de LNCS, pages 404, Orford, Québec, Canada, Février 2010. Springer.
7. Abderrahman Matoussi and Dorian Petit. « Specification of a localization component : feedback ». *Proceedings of the 2nd International Conference on Abstract State Machines, Alloy, B and Z (ABZ'10)*, vol. 5977 de LNCS, pages 401-402, Orford, Québec, Canada, Février 2010. Springer.
8. Abderrahman Matoussi, Frédéric Gervais and Régine Laleau. « A First Attempt to Express KAOS Refinement Patterns with Event B ». *Proceedings of the First International Conference on Abstract State Machines, B and Z (ABZ'08)*, vol. 5238 de LNCS, pages 338, London, UK, Septembre 2008. Springer.

Workshops internationaux avec comité de sélection et actes

9. Abderrahman Matoussi. « Expressing KAOS Goal Models with Event-B ». *Proceedings of Doctoral Symposium of 16th International Symposium on Formal Methods (FM'09-DS)*, pages 60-67, Eindhoven, The Netherlands, Novembre 2009. [4]

4. http://alexandria.tue.nl/repository/books/654108.pdf

10. Abderrahman Matoussi, Frédéric Gervais and Régine Laleau. « Expressing KAOS Goal Refinement Patterns with Event-B ». *Proceedings of the Rodin User and Developer Workshop*, Southampton, United Kingdom, Juillet 2009. [5]

Conférences nationales avec comité de sélection et actes

11. Aderrahman Matoussi et Régine Laleau. « Une première approche de traçabilité entre modèles d'exigences non-fonctionnelles et spécifications abstraites Event-B ». *Actes du XXIXème Congrès INFORSID*, pages 301-316, Lille, France, Mai 2011.
12. Abderrahman Matoussi, Frédéric Gervais et Régine Laleau. « De KAOS vers Event-B : Approche dirigée par les buts ». *Actes de l'Atelier Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2009)*, pages 71-86, Toulouse, France, Janvier 2009.

Workshops nationaux avec comité de sélection et actes

13. Abderrahman Matoussi, Frédéric Gervais et Régine Laleau. « Définition d'une sémantique Event-B pour les patrons de raffinement de buts KAOS ». *Actes des journées du GDR CNRS Génie de la programmation et du logiciel*, pages 167-185, Pau, France, Mars 2010.
14. Abderrahman Matoussi, Frédéric Gervais et Régine Laleau. « Une première formalisation du modèle des exigences ». *Actes des journées du GDR CNRS Génie de la programmation et du logiciel*, pages 270-271, Toulouse, France, Janvier 2009.

5. http://deploy-eprints.ecs.soton.ac.uk/137/

Première partie

Contexte

CHAPITRE 1

Les concepts de base

Sommaire

Pour améliorer la compréhension des nombreux concepts utilisés dans cette thèse, ce chapitre est consacré à la présentation de l'ingénierie des exigences, en particulier de l'ingénierie des exigences dirigée par les buts et des méthodes formelles B et Event-B.

1.1 L'ingénierie des exigences

1.1.1 Définition de l'ingénierie des exigences

La mesure principale du succès d'un système est le degré auquel il répond à l'objectif fixé à l'avance [Nuseibeh & Easterbrook 2000]. En ce sens, l'ingénierie des exigences est le processus de découverte de cet objectif en identifiant, analysant et documentant les parties prenantes et leurs besoins. Les exigences sont donc l'expression d'un besoin documenté sur ce qu'un système ou un logiciel devrait être ou faire. Les parties prenantes quant à elles désignent des personnes, des organisations ou des systèmes ayant une interaction directe ou indirecte avec les exigences. L'une des définitions les plus claires de l'ingénierie des exigences est proposée par [Zave 1997] :

« L'ingénierie de exigences est la branche du génie logiciel qui concerne les buts réels, les fonctions et les contraintes d'un système logiciel. Elle concerne aussi la relation que ces facteurs entretiennent avec les spécifications précises du comportement logiciel, et avec leur évolution dans le temps et à travers les familles de logiciels. »

Notons que même si cette définition concerne les logiciels, elle peut être appliquée à des systèmes en général. Cette définition est intéressante pour un certain nombre de raisons soulignées dans [Nuseibeh & Easterbrook 2000]. Premièrement, elle souligne l'importance « des buts réels » qui motivent le développement d'un système. Ces buts représentent le *pourquoi* ainsi que le *quoi* d'un système. Deuxièmement, cette définition se réfère à « des spécifications précises » qui fournissent la base pour analyser les exigences, les valider, définir ce que les concepteurs doivent construire, et vérifier qu'elles sont faites correctement lors de la livraison. Enfin, la définition se réfère à des spécifications qui « évoluent dans le temps et à travers les familles de logiciels », mettant l'accent ainsi sur la réalité d'un monde changeant et la nécessité de la réutilisation de spécifications partielles, comme le font les ingénieurs dans d'autres branches de l'ingénierie.

L'analyse des exigences est par conséquent une phase critique parce que s'il y a des erreurs à ce niveau, il est difficile et coûteux de les corriger ultérieurement dans les autres phases. Par exemple, [Boehm & Papaccio 1988] estime que la correction d'une erreur de spécification pendant la phase de maintenance peut coûter jusqu'à 200 fois plus chère que pendant la phase d'analyse des exigences. Pour cela, il est important de s'intéresser à la façon dont ces exigences sont spécifiées. Il y a deux manières extrêmes pour la spécification des exigences : une spécification complètement formelle et une spécification informelle. Il existe aussi certaines représentations intermédiaires qui ont montré leur utilité pour la communication entre les parties prenantes et les analystes.

1.1.2 Les exigences fonctionnelles vs les exigences non-fonctionnelles

Dans l'ingénierie des exigences, il existe une distinction importante entre les exigences fonctionnelles et les exigences non-fonctionnelles, notées NFRs (*Non-Functional Requirements* en anglais) [Glinz 2007]. Les exigences fonctionnelles se réfèrent aux services que le futur système devrait fournir tandis que les NFRs se limitent à la manière dont ces services doivent être fournis ; i.e la qualité de service. Les NFRs touchent beaucoup d'aspects tels que la performance, les contraintes de conception, les attributs de qualité, etc. Pour mieux comprendre le concept des NFRs, [Glinz 2007] constitue une excellente référence qui recense et discute la plupart des définitions des 20 dernières années.

Rappelons que l'analyse des NFRs n'est pas toujours évidente puisque généralement il est difficile de les exprimer d'une façon objective et mesurable, comme le souligne [Nuseibeh & Easterbrook 2000]. Par ailleurs, les NFRs sont connues pour leur caractère conflictuel. Par exemple, l'utilisation d'un double mot de passe augmente la sécurité des données, mais elle a des impacts négatifs sur les performances du système (ralentir le système) et l'utilisabilité.

Généralement, les méthodes existantes d'ingénierie des exigences construisent les modèles où les exigences fonctionnelles et non-fonctionnelles sont étroitement liées. Dans ce contexte, Axel van Lamsweerde [van Lamsweerde 2009] pense par exemple que la distinction entre les exigences fonctionnelles et non-fonctionnelles ne doit pas être prise dans un sens strict et clair, acceptant ainsi des chevauchements entre ces deux types d'exigences. Or, il s'avère qu'en pratique ces deux types d'exigences n'évoluent pas au fil du temps de la même manière et que les exigences fonctionnelles sont beaucoup plus stables que les non-fonctionnelles. C'est pourquoi nous avons adopté dans cette thèse un point de vue différent, en mettant l'accent sur la séparation entre ces deux types d'exigences.

1.1.3 Activités de l'ingénierie des exigences

[van Lamsweerde 2009, Nuseibeh & Easterbrook 2000] décrivent les activités suivantes qui caractérisent une démarche d'ingénierie des exigences.

1. *La compréhension du domaine :* Cette activité consiste en l'identification des parties prenantes à l'aide d'interviews en étudiant aussi l'environnement du système. Les problèmes du système actuel sont ainsi découverts et les possibilités d'amélioration sont analysées.
2. *L'élicitation* [1] *des exigences :* Cette activité consiste à découvrir les exigences et les hypothèses candidates pour le futur système en se basant sur les faiblesses du système actuel, extraites dans l'activité de la compréhension du

1. Le mot éliciter vient du latin **elicere** qui, d'après le dictionnaire latin-français *Le Grand Gaffiot* signifie « tirer de, faire sortir, attirer ».

domaine. Plusieurs techniques peuvent être employées pour avoir les informations appropriées telles que les interviews avec les parties prenantes, les scénarios, les questionnaires, etc.

3. *L'évaluation des exigences :* Le but de cette activité est d'évaluer les exigences afin de choisir les meilleures alternatives, en se basant par exemple sur l'analyse des risques et des conflits entre les exigences.
4. *La spécification et la documentation des exigences :* Cette activité consiste à détailler, structurer, et documenter les caractéristiques acceptées du futur système telles qu'elles ressortent de l'activité d'évaluation. Le produit obtenu est une première version du cahier des charges qui est écrit en langage naturel, semi-formel (des diagrammes) ou formel.
5. *L'assurance de la qualité des exigences :* Le cahier des charges obtenu à l'issue de l'activité précédente doit être soigneusement analysé et validé avec les parties prenantes afin de repérer les insuffisances (incohérences, omissions, contradictions ou ambiguïtés) par rapport aux besoins réels. Diverses techniques peuvent être utilisées telles que les reviews, l'animation, etc. Le résultat final de cette activité est un cahier des charges consolidé.
6. *La gestion de l'évolution des exigences :* Elle consiste à se préparer aux changements que le futur système peut subir. Cette anticipation d'évolution doit prendre en compte tous les aspects (les exigences, les hypothèses sur l'environnement, etc.) en utilisant diverses techniques telles que la gestion de la traçabilité, les changements à la volée, etc.

Même s'il y a des dépendances de données entre ces différentes phases dans la mesure où chaque phase a besoin des données de la phase précédente, ces phases sont généralement étroitement liées et elles peuvent se chevaucher. Ainsi, le processus de l'ingénierie des exigences peut être considéré comme une itération sur des incréments successifs selon le modèle en spirale [Boehm 1988].

1.1.4 Les approches de l'ingénierie des exigences

La littérature propose plusieurs approches d'ingénierie des exigences. Chacune d'entre elles se concentrent sur des activités spécifiques du processus d'ingénierie des exigences. Dans ce qui suit, nous présentons brièvement quelques approches :

- L'approche à base de scénarios [Sutcliffe 2003, Rolland *et al.* 1998] : Dans ce type d'approche, les exigences sont décrites à l'aide de scénarios. Ces derniers sont des descriptions du monde réel qui sont exprimées à l'aide du langage naturel, de diagrammes, etc.

- L'approche GORE [van Lamsweerde 2009] : Elle se base sur la définition des exigences comme étant des buts qui peuvent être divisés et raffinés.

Cette approche est détaillée dans la suite.

– L'approche orientée-aspects [Rashid & Chitchyan 2008] : Cette approche reconnaît explicitement l'importance de bien identifier, de traiter et de séparer des préoccupations transversales qui ont été par ailleurs réparties dans des artefacts d'autres exigences (des cas d'utilisation, des modèles de buts, etc).

– L'approche « Problem Frames » (schéma de problème) [Jackson 2001] : C'est une approche qui permet de structurer l'analyse des exigences d'un logiciel et de concevoir une solution logicielle. Elle aide à comprendre d'une part le contexte dans lequel réside le problème et d'autre part les aspects pertinents pour la conception d'une solution. Cette approche s'intéresse surtout aux exigences fonctionnelles.

Dans la dernière décennie, la popularité de l'approche GORE a augmenté due au fait que son processus d'élaboration se termine là où la plupart des autres approches d'ingénierie des exigences débutent [van Lamsweerde & Letier 2002]. GORE se concentre beaucoup sur les activités qui précèdent la formulation des exigences [Nuseibeh & Easterbrook 2000]. En effet, la plupart des autres approches d'ingénierie des exigences sont concentrées sur ce que le logiciel doit faire et comment il doit le faire en traitant les exigences seulement en terme de processus et de données et en ne s'intéressant donc pas à la justification de ces exigences. Par conséquent, il sera difficile par la suite de comprendre les exigences et de juger si elles capturent vraiment tous les besoins des parties prenantes. Aussi, [Castro *et al.* 2002] affirme que la tendance de la plupart des autres approches est d'abstraire les constructions de programmation (de bas niveau) au niveau des exigences plutôt que de pousser les abstractions des exigences jusqu'au niveau de la conception. Ainsi, on va se focaliser dans ce qui suit sur l'approche GORE en la détaillant et en présentant quelques méthodes qui s'y réfèrent.

1.2 L'approche GORE

La naissance de l'approche GORE remonte à plus de deux décennies avec précisément le travail de Yue [Yue 1987] qui était le premier à affirmer que la modélisation explicite des buts dans les modèles d'exigences peut fournir un critère pour la complétude des exigences. [Dardenne *et al.* 1993, van Lamsweerde 2009] définissent un but comme un objectif que le système et son environnement doivent réaliser grâce à la coopération de différents agents (matériel, logiciel ou humain). Un but placé sous la responsabilité d'un agent du système est appelé une exigence (*requirement*), tandis qu'un but placé sous la responsabilité d'un agent de l'environnement du système est appelé une attente (*expectation*). Les principales activités suivantes sont normalement présentes dans la plupart des méthodes GORE :

l'élicitation des buts, le raffinement de buts, divers types d'analyse des buts et l'attribution de la responsabilité d'un but à un agent. Dans ce qui suit, nous présentons une panoplie de méthodes adoptant le paradigme GORE.

1.2.1 Le NFR framework

Le NFR framework [Chung *et al.* 2000] se concentre sur la modélisation et l'analyse des exigences non-fonctionnelles afin de les mettre dès le début dans l'esprit de l'analyste avant même les exigences fonctionnelles. L'idée principale de la méthode consiste à systématiquement modéliser et raffiner les NFRs et à étudier les interdépendances, les influences et les priorités entre ces NFRs. Pour cela, le NFR framework se base sur les « softgoals »[2] qui sont divisés en trois types : (i) *les softgoals NFR* représentent les NFRs à prendre en considération ; (ii) *les softgoals d'opérationnalisation* modélisent les techniques permettant de satisfaire les softgoals NFR ; (iii) *les claim softgoals* permettent à l'analyste de justifier ses choix pour le raffinement de softgoals, les priorités entre softgoals, etc.

Les softgoals peuvent être raffinés en utilisant le raffinement ET/OU. Les interdépendances entre les softgoals peuvent généralement être capturées avec des contributions positives (+) ou négatives (-). Tous ces concepts sont représentés graphiquement grâce au graphe d'interdépendance des softgoals (SIG). Le NFR framework offre une démarche qui s'appuie sur les activités suivantes : (i) la capture des softgoals NFR ; (ii) le raffinement des softgoals NFR ; (iii) l'identification des softgoals d'opérationnalisation pour satisfaire les softgoals NFR ; (iv) l'étude des ambiguïtés, des compromis, des priorités et des interdépendances entre les NFRs ; (v) l'analyse des différentes alternatives d'opérationnalisation selon une procédure bien définie qui détermine l'impact sur les NFRs de haut niveau. L'analyste sélectionne enfin l'alternative qui répond le mieux aux NFRs de haut niveau. L'ensemble des softgoals d'opérationnalisation pour l'alternative sélectionnée peut être alors implémenté dans le futur logiciel.

Afin de faciliter la tâche de l'analyste tout au long de cette démarche, le NFR framework fournit un ensemble de catalogues stockant les connaissances de conception. Ces catalogues sont de trois types :

- *Les catalogues des types NFR* incluent des concepts sur des types particuliers des NFR, tels que la performance, etc.
- *Les catalogues des méthodes* encodent les connaissances qui aident dans le raffinement des softgoals et dans l'opérationnalisation.
- *Les catalogues des règles de corrélation* stockent les connaissances qui aident à détecter les interdépendances implicites entre les softgoals. Par exemple, le catalogue pourrait inclure le fait que l'indexation contribue positivement au temps de réponse.

2. contrairement au hardgoal, un softgoal est un but qu'on ne peut pas définir formellement.

1.2.2 Le framework i*

Le framework iSTAR ou i*, acronyme de « *Intentional STratégic Actor Relationships* » [Yu 1995, Yu 1997], part du principe que les acteurs (les parties prenantes ou les entités actives) d'un système sont non seulement reliés entre eux par leurs actions ou l'information qu'ils échangent, mais aussi par les attributs intentionnels (les buts, les capacités, les croyances et les engagements) qui les caractérisent. Le framework i* comporte deux modèles principaux : le modèle de dépendance stratégique et le modèle de raisonnement stratégique. Dans ces modèles i*, les acteurs sont décrits dans leur contexte organisationnel et dépendent les uns des autres pour réaliser leurs buts, exécuter leurs tâches, ou disposer de ressources.

i* peut être utilisé tout au long des phases du processus d'ingénierie des exigences. Pendant les premières phases, le framework i* est utilisé pour faciliter l'analyse du domaine par des diagrammes qui permettent de représenter les acteurs du système, leurs buts et leurs relations. Les modèles i* développés à ce stade aident à comprendre pourquoi le futur système est nécessaire. Durant les dernières phases de l'ingénierie des exigences, les modèles i* sont utilisés pour proposer de nouvelles configurations du système qui seront évaluées en fonction de la façon dont elles répondent aux besoins fonctionnels et non-fonctionnels des utilisateurs. Pour cela, les softgoals (comme dans le NFR framework) sont utilisés comme critères de sélection pour choisir l'alternative du processus de configuration qui répond le mieux aux NFRs du système. En i*, il y a les liens de type « moyen-finalité » (*means-ends* en anglais) entre les buts pour spécifier les différentes alternatives pour réaliser un but. Les liens de décomposition quant à eux connectent un but/tâche avec ses composants (sous-tâches, softgoals, etc.). Un softgoal, un but ou une tâche peuvent être également liés à d'autres softgoals avec des liens de contribution, comme dans le NFR framework. Ces liens de contribution spécifient deux niveaux de contributions positifs (+ et ++) et négatifs (- et –) pour la satisfaction d'un softgoal, la réalisation d'un but, ou l'exécution d'une tâche.

Un langage, basé sur i* et nommé *Goal-oriented Requirements Language* (GRL), définit tous les concepts de i* (but, tâche, ressource, liens de contribution, etc.). GRL [Liu & Yu 2001] est une partie de la notation des exigences utilisateur (URN) qui est une nouvelle recommandation de l'union internationale des télécommunications (ITU). URN fournit le premier standard des langages orientés-buts et il intègre aussi la notation UCM (Use Case Maps).

TROPOS [3] [Castro *et al.* 2001] est une méthode d'ingénierie logicielle orientée-agent qui couvre tout le processus de développement logiciel. TROPOS se base sur i* afin de couvrir les toutes premières phases de l'analyse des exigences, permettant ainsi une meilleure compréhension de : (i) l'environnement dans lequel le logiciel doit fonctionner ; (ii) types d'interactions qui devraient se produire entre les logiciels et les agents humains. TROPOS adopte le framework i* afin d'exprimer les concepts d'acteur, de buts et de leurs interdépendances.

3. TROPOS désigne en grec la façon de faire les choses.

1.2.3 GBRAM

La méthode GBRAM, acronyme de « *Goal-Based Requirements Analysis Method* » [Anton 1996, Anton 1997], sert à l'identification et l'abstraction de buts à partir de diverses sources d'information en utilisant un ensemble de questions standards. GBRAM comprend deux activités : l'analyse des buts et le raffinement des buts. L'analyse des buts consiste à explorer les sources d'information pour identifier les buts, les organiser et les classifier. L'intérêt de GBRAM est qu'il fait la distinction entre les buts de réalisation (les buts fonctionnels) et les buts de maintenance (les buts non-fonctionnels). L'activité de raffinement de buts quant à elle concerne l'évolution des buts à partir du moment où ils sont identifiés jusqu'au moment où ils sont traduits vers des exigences opérationnelles. Tout au long de l'activité de raffinement de buts, GBRAM définit la relation de *précédence* qui consiste à trouver les buts qui précèdent les autres. Tous les concepts de GBRAM (buts, agents, parties prenantes...) sont spécifiés seulement sous une forme textuelle dans des schémas de buts, sans fournir aucune notation graphique.

1.2.4 AGORA

La méthode AGORA, acronyme de « Attributed Goal-Oriented Requirements Analysis Method » [Kaiya *et al.* 2002], se caractérise par l'attachement des valeurs d'attributs (de préférence et de contribution) au graphe de buts fournissant ainsi des techniques d'analyse quantitative sur les buts. Tout au long du processus de raffinement et de décomposition de buts, un analyste attribue des valeurs de contribution et de préférence pour les arêtes et les nœuds d'un graphe de buts, respectivement. La valeur de contribution d'une arête représente le degré de la contribution du sous-but à la réalisation de son but parent, tandis que la matrice de préférence d'un nœud (un but) représente le degré de préférence ou de satisfiabilité de ce but pour chaque partie prenante. Ces valeurs quantitatives peuvent aider un analyste à choisir un but parmi les alternatives de buts, à reconnaître les conflits entre les buts et à analyser l'impact des changements des exigences. En outre, les valeurs sur un graphe de buts et ses caractéristiques structurelles permettent à l'analyste d'estimer la qualité des spécifications des exigences résultantes, telles que l'exactitude, la non-ambiguïté, la complétude, etc. Les valeurs de qualité estimées peuvent lui suggérer quels buts devraient être améliorés et/ou raffinés.

1.2.5 Crews L'Ecritoire

L'approche CREWS-L'Ecritoire [Rolland *et al.* 1998] consiste à découvrir les exigences à travers un couplage bi-directionnel entre les buts et les scénarios. Ce couplage permet ainsi le mouvement de buts vers des scénarios (concrétiser un but par un scénario) et vice-versa. Le processus de CREWS-L'Ecritoire est défini alors comme suit : (i) un scénario est écrit pour chaque but identifié ; (ii) une fois le

scénario écrit, il est analysé afin de produire d'autres buts qui représentent des cas fonctionnels alternatifs ou des cas exceptionnels de non-satisfaction du but initial.

Dans ce processus, la découverte des buts et l'écriture des scénarios sont deux étapes complémentaires. Les buts/scénarios sont progressivement découverts en répétant le cycle : analyse des buts, écriture des scénarios et identification des buts par l'analyse de scénario. Chacune de ces trois étapes du cycle est supportée par des mécanismes qui guident l'exécution de l'étape. CREWS-L'Ecritoire repose sur le concept du fragment d'exigence (*Requirement Chunk* en anglais) qui est une paire but-scénario et qui constitue un moyen possible d'atteindre un but. Il existe trois types de relations entre les fragments d'exigence : la relation de composition, la relation d'alternative, et la relation de raffinement. Les deux premières relations mènent à une structure horizontale ET/OU entre les fragments d'exigence. Ce sont des extensions de relations classiques ET/OU entre les buts. La relation de raffinement quant à elle relie les fragments d'exigence à différents niveaux d'abstraction, établissant ainsi un lien vertical entre les fragments.

1.2.6 La méthode KAOS

KAOS, acronyme de « *Keep All Objectives Satisfied* » [Dardenne *et al.* 1993, van Lamsweerde 2009], est une méthode d'ingénierie des exigences qui résulte des travaux de recherche menés à l'Université de Louvain, en collaboration avec l'Université d'Oregon. Cette méthode permet aux analystes de construire des modèles d'exigences et de produire des documents à partir de ces modèles. KAOS a distingué les buts des propriétés du domaine qui sont des déclarations descriptives sur l'environnement telles que des lois physiques ou des normes organisationnelles. Le modèle des exigences KAOS se compose de cinq sous-modèles fortement liés par des règles de cohérence :

- Le modèle central est *le modèle de buts* qui décrit les buts du système et de son environnement. Ce modèle est organisé dans une hiérarchie obtenue grâce au raffinement de buts de plus haut niveau (les buts stratégiques) vers des buts de bas niveau (les exigences).
- *Le modèle objet* : il permet de décrire le vocabulaire du domaine. Il est représenté par un diagramme de classes UML.
- *Le modèle des responsabilités* : il permet d'assigner les exigences (les buts feuilles) aux différents agents. Ces agents appartiennent au système à construire (agents internes) ou à son environnement (agents externes).
- *Le modèle des opérations* : il représente les opérations du système en termes de leurs caractéristiques individuelles et leurs liaisons avec les modèles de buts (liens d'opérationnalisation des exigences), objet (liens entrée-sortie) et responsabilités (liens d'exécution).
- *Le modèle des comportements* : il résume tous les comportements que les agents doivent accomplir pour satisfaire les exigences. Ces comportements

sont exprimés sous la forme d'opérations exécutées par les agents responsables.

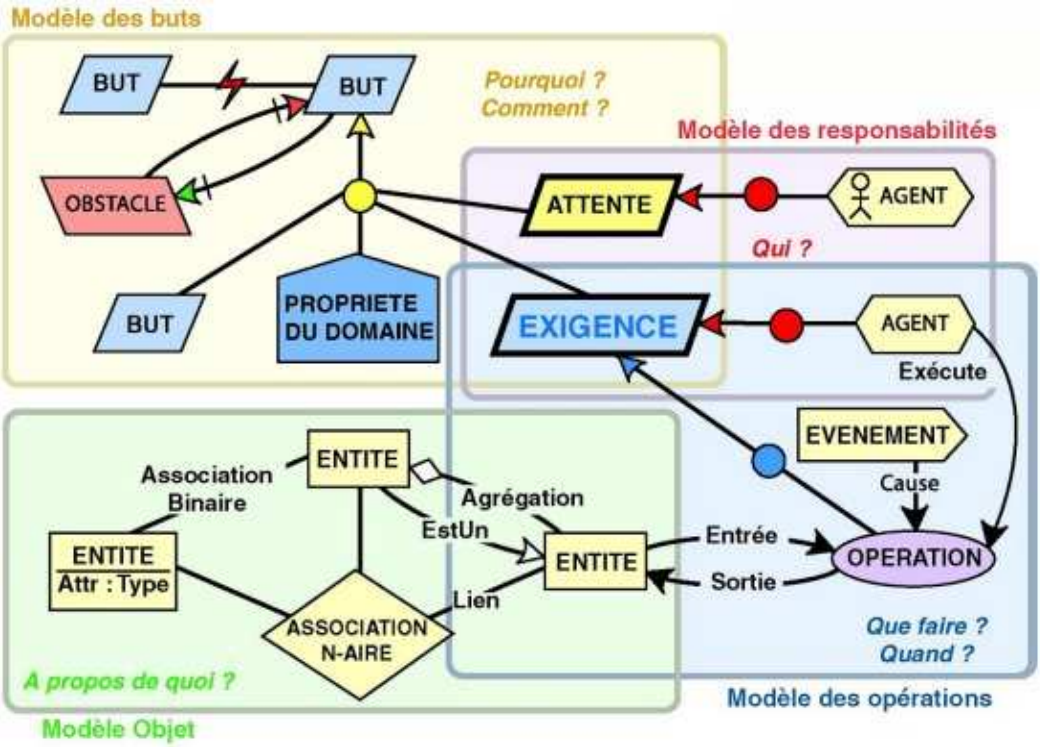


FIGURE 1.1: Un aperçu du modèle des exigences KAOS

La Figure 1.1 donne un aperçu des quatre premiers sous-modèles KAOS ainsi que des principaux éléments qui les constituent [4]. Le modèle des exigences KAOS peut être vu donc comme une sorte de puzzle visant à assembler les différentes pièces identifiées au fur et à mesure que les sources d'information sont exploitées. Le modèle de buts joue un rôle central dans la méthode KAOS. Par exemple, il est possible de dériver certains modèles KAOS (le modèle des opérations, le modèle objet, etc.) à partir du modèle de buts d'une façon systématique [van Lamsweerde 2009]. De plus, le modèle de buts permet de supporter très tôt différentes formes d'analyse des exigences telles que l'analyse de risques, l'analyse de conflits, ou l'évaluation des alternatives.

Comme dans la plupart des méthodes GORE, les buts dans KAOS peuvent être raffinés selon des raffinements ET ou OU. Un raffinement ET implique que la conjonction des sous-buts est une condition suffisante pour réaliser le but parent. Un raffinement OU associe un but à des sous-buts alternatifs pour lesquels l'accomplissement du but de plus haut niveau exige l'accomplissement d'au moins un de ses sous-buts. KAOS offre un ensemble de patrons de raffinement [Darimont & van Lamsweerde 1996, van Lamsweerde 2009] qui décom-

4. www.respect-it.be/

posent les buts. Ces patrons ne peuvent être utilisés que dans le contexte de différentes tactiques telles que : (i) *le raffinement par jalons* qui consiste à identifier les sous-buts comme des étapes successives dans le temps permettant de satisfaire le but de plus haut niveau ; (ii) *le raffinement par cas* qui consiste à identifier différents cas possibles pour satisfaire le but où l'ensemble des cas forme le but principal.

Les sous-buts $G_1, G_2, ..., G_n (n \geq 1)$ raffinent le but G sous *Dom* (les propriétés du domaine et les hypothèses) selon le raffinement ET si et seulement si les conditions suivantes sont vérifiées [5] :

1. $\{G_1, G_2, \ldots, G_n, Dom\} \models G$ (conséquence logique)
2. $\{G_1, ..., G_{j-1}, G_{j+1}, ..., G_n, Dom\} \not\models G$ (minimalité)
3. $\{G_1, G_2, \ldots, G_n, Dom\} \not\models faux$ (cohérence)

La première condition exprime que la satisfaction de tous les sous-buts, en considérant aussi *Dom*, est suffisante pour la satisfaction du but parent G. La seconde condition affirme que si l'un des sous-buts dans le raffinement est manquant (par exemple G_j), la satisfaction du but parent n'est plus garantie. La troisième condition exprime que les différents sous-buts, les propriétés du domaine et les hypothèses ne peuvent pas se contredire les uns les autres.

Les patrons de raffinement KAOS sont regroupés selon le comportement de différents types de buts. Ces derniers peuvent être *Achieve, Maintain, Cease* ou *Avoid*. Un but *Achieve* décrit des comportements attendus du système tels qu'une certaine condition cible doit être atteinte un jour dans le futur. Un but *Maintain* décrit des comportements attendus du système tels qu'une certaine « bonne » condition doit être toujours vraie quelque soit l'état du système. Les buts *Cease* et *Avoid* sont quant à eux les variantes duales des buts *Achieve* et *Maintain*, respectivement. Un principe de la méthode KAOS est d'arrêter le raffinement lorsqu'il est possible d'assigner chaque exigence à un agent particulier. C'est cet agent qui doit effectuer toutes les opérations permettant d'opérationnaliser cette exigence. Notons que KAOS est supportée par un outil propriétaire, appelé Objectiver [6].

1.2.7 SysML/KAOS

Bien que la plupart des approches GORE permettent d'exprimer de manière structurée et précise les exigences auxquelles un système doit répondre, elles ne considèrent que la phase d'analyse des exigences. En effet, ces approches ne proposent pas des liens entre les exigences identifiées et les spécifications élaborées dans les étapes suivantes du développement logiciel. Cela rend difficile la validation des spécifications par rapport aux exigences. L'OMG tente depuis peu de

5. Notons que S$\models$A signifie : « A est toujours satisfaite dans toute circonstance où S est satisfaite ». On dit alors que S implique sémantiquement A. Par contre, S$\not\models$A signifie que S n'implique pas sémantiquement A.

6. http://www.objectiver.com/

répondre à ce problème en proposant SysML [Friedenthal *et al.* 2008], langage de modélisation pour l'analyse et la spécification de systèmes complexes. SysML est un profil UML qui utilise certains diagrammes UML et propose aussi des extensions comme la prise en compte des exigences. Un des aspects intéressants est que SysML permet tout au long du cycle de développement de relier les exigences avec d'autres types d'éléments, assurant ainsi une continuité depuis l'analyse des besoins jusqu'à l'implémentation. Malgré ses apports, l'ensemble des concepts proposés dans SysML n'est pas aussi riche que dans les autres méthodes d'ingénierie des exigences : (i) les exigences sont spécifiées de manière informelle sous forme de texte ; (ii) la sémantique des relations entre exigences (comme par exemple la relation de contenance) n'est pas définie de manière précise, ce qui entraîne des ambiguïtés [Goknil *et al.* 2011].

Afin de pouvoir bénéficier des apports de SysML, comme par exemple les relations de traçabilité, tout en garantissant une définition précise des exigences et des relations entre les exigences, [Laleau *et al.* 2010, Gnaho & Semmak 2011] proposent d'étendre SysML avec les concepts les plus pertinents utilisés dans les approches GORE. Le résulat de ces extentions est une nouvelle méthode : la méthode SysML/KAOS. C'est un profil SysML qui s'inspire de : (i) la méthode KAOS pour représenter les buts fonctionnels ; (ii) la méthode NFR framework et le framework i* pour représenter les buts non-fonctionnels et leurs impacts sur les buts fonctionnels. Dans ce qui suit, nous présentons ces différentes extensions. Les boîtes grises des Figures 1.2 et 1.3 représentent un extrait du méta-modèle du diagramme des exigences de SysML [7]. Les concepts intégrés dans SysML/KAOS sont représentés par les boîtes blanches.

1.2.7.1 SysML/KAOS : Les buts fonctionnels

Comme KAOS permet de concevoir des modèles de buts fonctionnels très expressifs et de représenter ces modèles à différents niveaux d'abstraction, [Laleau *et al.* 2010] propose tout d'abord d'étendre le méta-modèle des exigences de SysML avec les concepts des buts fonctionnels KAOS. Comme le montre la Figure 1.2, un méta-modèle étendu a été proposé (le méta-modèle SysML/-KAOS) pour construire des hiérarchies d'exigences fonctionnelles sous forme de buts. Une exigence (méta-classe **Requirement**) possède un identifiant **IdReq**, un nom **NameReq** ainsi qu'un texte **TextReq** qui la décrit. Les associations **Derives**, **Contains** et **Copy** permettent de représenter les relations entre exigences. Rappelons que les boîtes blanches de la Figure 1.2 représentent l'extension de ce méta-modèle pour prendre en compte le modèle de buts KAOS. La méta-classe **Goal**, qui est spécifiée comme une sous-classe de la méta-classe **Requirement** de SysML, peut être spécialisée en un but fonctionnel (méta-classe **Functional Goal**) et un but non-fonctionnel (méta-classe **Non Functional Goal**, détailléé dans la Figure 1.3).

Un but fonctionnel est caractérisé par son **TypeGoal** qui précise le type d'un

7. www.sysml.org/specs/

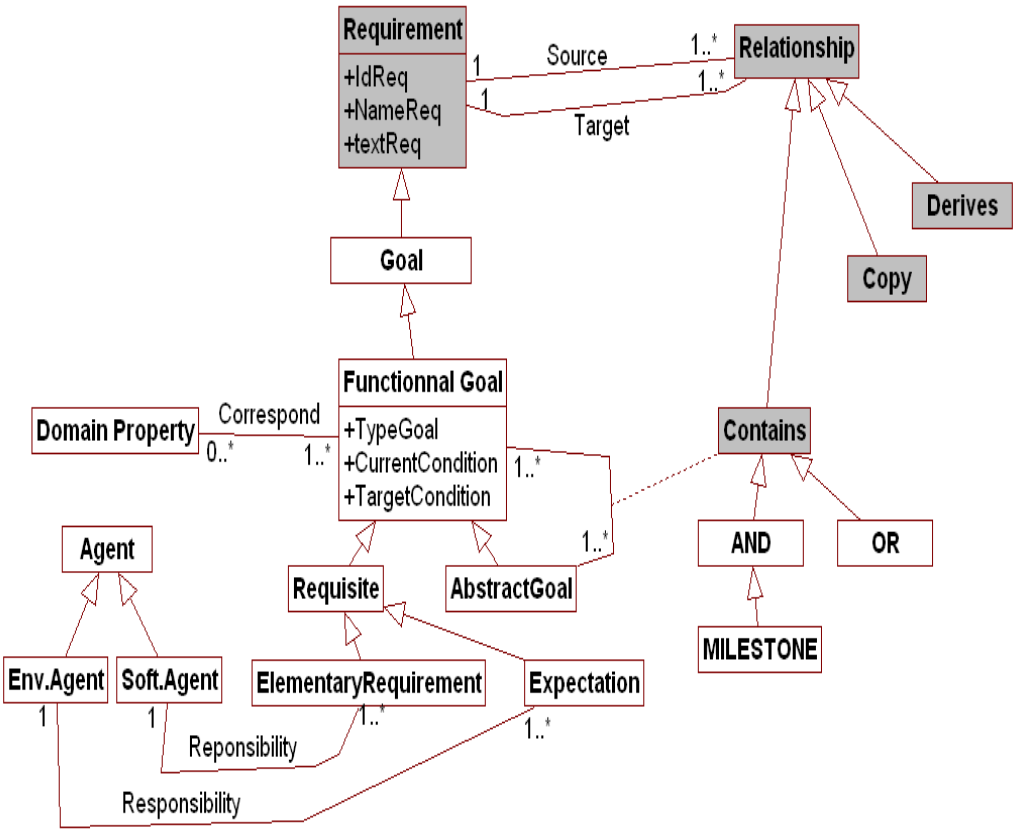


FIGURE 1.2: Une vue du méta-modèle des buts fonctionnels SysML/KAOS

but (« Achieve » ou « Cease »). Un but « Achieve » prescrit des comportements attendus où une certaine condition cible (**TargetCondition**) doit être tôt ou tard établie quand une autre condition (**CurrentCondition**) est vérifiée dans l'état actuel du système. Un but fonctionnel peut être soit un but abstrait (méta-classe **Abstract Goal**), soit un besoin (méta-classe **Requisite**). Un but abstrait est raffiné en sous-buts. Les hiérarchies de buts sont modélisées par l'intermédiaire de l'association **Contains**. Cette dernière devient une classe association entre un but abstrait et ses sous-buts qui peut être spécialisée pour représenter le raffinement de buts (AND, OR, MILESTONE). Un but qui n'est plus raffiné est un besoin qui peut être soit : (i) une exigence élémentaire (méta-classe **Elementary Requirement**) qui est placée sous la responsabilité d'un agent du système (méta-classe **Soft. Agent**) ; (ii) une attente (méta-classe **Expectation**) qui est placée sous la responsabilité d'un agent de l'environnement du système (méta-classe **Env. Agent**).

1.2.7.2 SysML/KAOS : Les buts non-fonctionnels et leurs impacts

Le travail de [Gnaho & Semmak 2011] s'inspire par la suite du NFR framework et du framework i* pour compléter le méta-modèle SysML/KAOS obtenu par les concepts liés aux buts non-fonctionnels et l'impact de ces buts sur les

buts fonctionnels. Comme le montre la Figure 1.3, la méta-classe **Non Functional Goal** représente les buts non-fonctionnels. Comme pour les buts fonctionnels, cette méta-classe est spécifiée comme une sous-classe de la méta-classe **Goal**, qui elle-même est une sous-classe de la méta-classe **Requirement** de SysML. Un but non-fonctionnel permet de représenter une qualité que doit avoir le futur système pour satisfaire une exigence fonctionnelle. L'attribut **NFGType** spécifie le type d'exigence non-fonctionnelle tandis que l'attribut **Topic**(sujet) représente l'élément du domaine concerné par ce type d'exigence. L'écriture d'un but non-fonctionnel respecte donc la syntaxe suivante : *NFGType [Topic]*.

Un but non-fonctionnel peut être soit un but abstrait (méta-classe **Abstract NFG**), soit un but élémentaire (méta-classe **Elementary NFG**). Un but non-fonctionnel abstrait est raffiné en sous-buts. Les hiérarchies de buts sont modélisées par l'intermédiaire de l'association **Refinement**. Cette dernière devient une classe association entre but abstrait et ses sous-buts et représente comme dans les buts fonctionnels, un raffinement AND/OR. Un but qui ne peut plus être raffiné est un but élémentaire.

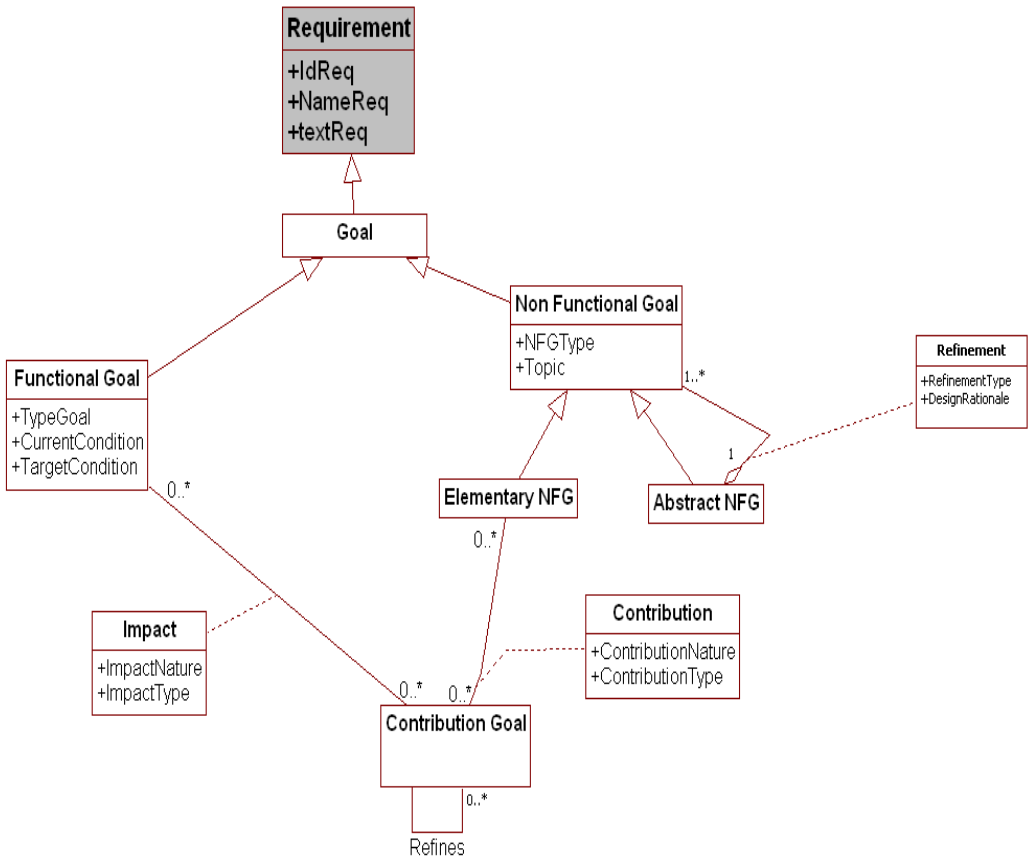


FIGURE 1.3: Une vue du méta-modèle des buts non-fonctionnels et leurs impacts SysML/-KAOS

Un but élémentaire peut être associé à plusieurs buts de contribution (méta-classe **Contribution Goal**) où chaque but de contribution représente une façon dont le but élémentaire peut être satisfait. La classe association **Contribution** décrit les caractéristiques de la contribution à la satisfaction du but élémentaire. Elle a deux propriétés : **ContributionNature** et **ContributionType**. La propriété **ContributionNature** permet de spécifier si la contribution est positive ou négative. Une contribution positive (ou négative) permet de décrire le fait qu'un but de contribution participe à satisfaire le but élémentaire de façon positive (ou négative). On rajoute un + ou un - selon que cette contribution est positive ou négative, sur le lien entre le but élémentaire et le but de contribution. L'attribut **ContributionType** permet de spécifier si la contribution est directe ou indirecte. Une contribution directe, représentée graphiquement par un trait plein permet de spécifier une contribution explicite au but élémentaire non-fonctionnel. Une contribution indirecte, représentée graphiquement par des tirets, permet de spécifier une contribution implicite au but élémentaire non-fonctionnel. Un but de contribution peut faire l'objet d'un raffinement AND/OR, ce qui est représenté par l'association réflexive dans le méta-modèle de la Figure 1.3.

Le concept **Impact** permet de relier les buts non-fonctionnels aux buts fonctionnels. Il représente le fait qu'un but de contribution a un effet sur un but fonctionnel. Un but fonctionnel exprime un besoin à satisfaire par le futur système. Par ailleurs, un but de contribution exprime une façon de réaliser un but non fonctionnel qui lui-même exprime une qualité attendue du système en termes de performance, de coût, de sécurité, etc. La question est la suivante comment une qualité recherchée peut-elle être prise en compte dans le futur système ? Comme indiqué dans le méta-modèle (voir la Figure 1.3), le concept **Impact** est représenté sous la forme d'une classe association entre un but de contribution et un but fonctionnel, montrant ainsi la façon dont un but non-fonctionnel peut impacter un but fonctionnel. Cet impact peut être comme dans le cas des contributions, de nature positive ou négative et de type direct ou indirect.

1.2.7.3 SysML/KAOS : Outillage

Un outil SysML/KAOS [Gnaho & Semmak 2011] a été développé dans l'environnement TOPCASED pour supporter ces extensions en utilisant l'éditeur Ecore et la technologie EMF [Steinberg *et al.* 2009]. Pour résoudre le problème des classes associations qui ne sont pas supportées en Ecore, l'outil reprend la technique de MERKS [8] : la classe association **Contains** est considérée comme une classe qui contient les données de la relation d'association plus une référence à chacune des deux classes associées (**Functional Goal** et **Abstract Goal**). Les classes associées quant à elles définissent une relation bidirectionnelle entre elles, plus une référence dans chacune de ces classes vers la classe association **Contains**.

8. www.ed-merks.blogspot.com/2008/01/modeling-associations-with-ecore.html

1.3 La méthode formelle Event-B

Les méthodes formelles sont des techniques qui peuvent être utilisées lors de chacune des phases du cycle de développement logiciel pour aider à structurer le raisonnement et apporter des garanties sur le développement. Pour cela, ces méthodes reposent sur un raisonnement mathématique rigoureux fondé sur un langage formel. Parmi les méthodes formelles, on peut citer la méthode Event-B [Abrial 2010]. C'est une évolution de la méthode B classique [Abrial 1996a]. Ces deux méthodes partagent d'ailleurs les mêmes principes fondamentaux (la logique du premier ordre et la théorie des ensembles) que les langages formels Z [Spivey 1988] et VDM [Bjørner & Jones 1978].

1.3.1 Un aperçu de la méthode B classique

La méthode B[9] [Abrial 1996a] est basée sur la notion de machine abstraite et l'utilisation de raffinements formellement prouvés. Elle a été utilisée dans le projet METEOR [Behm *et al.* 1999], dans le développement ferroviaire [Dehbonei & Mejia 1994] et dans le développement des systèmes non-critiques [Tatibouët *et al.* 2003]. Une spécification B est structurée en un ensemble de machines. La partie statique d'une machine B est la définition d'espace des états qui apparaît dans les clauses VARIABLES et INVARIANTS. La première clause énumère les composants d'état et la seconde définit des restrictions sur les valeurs possibles que ces composants peuvent prendre. L'état de la machine ne peut être modifié que par des opérations. Le langage permettant d'exprimer cette partie dynamique est un langage de substitutions généralisées. Une substitution[10] généralisée est un transformateur de prédicats[11] permettant d'écrire les opérations qui font évoluer l'état du système modélisé. Les principales substitutions du langage B sont présentées dans le tableau 1.1. Une opération peut posséder également des paramètres d'entrée et de sortie. Les opérations B peuvent être appelées par des composants extérieurs.

Le mécanisme de raffinement en B classique consiste à reformuler (par des étapes successives) les variables et les opérations de la machine abstraite, afin d'arriver finalement à un module qui constitue le programme à implémenter. Les étapes intermédiaires de reformulation sont appelées les raffinements et le dernier niveau de raffinement est appelé l'implémentation. Lors des phases initiales de spécification, les instructions des opérations utilisent des pré-conditions et de l'indéterminisme. Le raffinement fait que ces pré-conditions deviennent de plus en plus larges (faibles) et les instructions de plus en plus déterministes. Les machines B sont étiquetées selon leur niveau d'abstraction : MACHINE, REFINEMENT ou

9. Le choix de la lettre B est un hommage à Nicolas Bourbaki, un mathématicien imaginaire, dont le nom a été utilisé par un groupe de mathématiciens français qui entreprit depuis 1939 une refonte des mathématiques.
10. Une substitution désigne le remplacement d'une variable par une valeur qu'elle peut prendre.
11. Cette notion est analogue à celle de plus faible pré-condition de Hoare [Hoare 1969].

Notation B	Conditions de définition	Signification
x := Exp	x est une variable Exp est une expression	Substituer Exp à x
skip		Ne rien modifier
PRE P **THEN** S **END**	P est un prédicat S est une substitution	S'assurer de la pré-condition P et exécuter S
ASSERT P **THEN** S **END**	P est un prédicat S est une substitution	Exécuter S sous l'assertion que P est vrai
SELECT P **THEN** S **END**	P est un prédicat S est une substitution	Exécuter S si P est vrai
IF P **THEN** S **END**	P est un prédicat S est une substitution	Exécuter S si P est vrai (sinon skip)
VAR Z **IN** S **END**	Z une liste de variables S est une substitution	Les variables locales Z sont utilisables dans la substitution S
ANY X **WHERE** P **THEN** S **END**	X est une liste de variables P est un prédicat S est une substitution	Sélectionner une valeur de X qui vérifie le prédicat P et exécuter S
S ; T	S et T sont des substitutions	Exécuter S puis T
S || T	S et T sont des substitutions	Exécuter S et T en même temps
CHOICE T **OR** S **END**	S et T sont des substitutions	Exécuter S ou T

TABLE 1.1: Quelques substitutions du langage B.

IMPLEMENTATION, du niveau le plus abstrait jusqu'au niveau le plus concret. Ce mécanisme de raffinement permet de préserver des propriétés du système déjà prouvées dans des modèles de plus haut niveau. De là, le raffinement d'une opération est correctement prouvé s'il établit le même résultat que son abstraction.

Notons que B dispose également d'opérateurs (INCLUDES, SEES, USES, IMPORTS) permettant de composer des machines en utilisant d'autres machines dans le but d'architecturer le projet B. Par exemple, le mécanisme d'inclusion permet à l'état du système abstrait d'être divisé en plusieurs parties indépendantes où chaque partie est encapsulée dans une machine B « incluse » séparée. Ces parties peuvent alors être combinées dans une seule machine B « incluante » en utilisant la clause INCLUDES. Cette dernière permet de rendre visible, dans la machine « incluante » les données et les opérations des machines B « incluses ».

L'Atelier B [12] constitue l'outil référence supportant toutes les phases du processus de développement B et permettant de gérer des projets en langage B. l'Atelier B dispose aussi d'un outil de preuve interactif, appelé le prouveur B. Il permet de générer et prouver (automatiquement ou interactivement) les obligations de preuve qui garantissent la justesse et l'efficacité du développement de système.

12. www.atelierb.eu/

1.3.2 De la méthode B classique vers Event-B

La méthode Event-B [Abrial 2010] est une évolution de la méthode B, dont l'objectif est de modéliser des systèmes fermés. Un système fermé est un système modélisé avec l'ensemble de toutes les interactions avec son environnement. Il n'y a donc plus besoin de modéliser les entrées ou sorties pour communiquer avec l'environnement. Pour cela, les opérations B sont remplacées par des événements en Event-B. Contrairement aux opérations B qui sont appelées par des composants, les événements Event-B se déclenchent spontanément si une condition (appelée *garde*) devient vraie. Contrairement au B classique, Event-B offre également la possibilité d'exprimer certaines contraintes dynamiques telles que des contraintes de vivacité (*liveness* en anglais). Ces points font que le B classique est mieux approprié pour le développement de logiciels (spécification par contrat) et Event-B pour le développement de systèmes (spécification par observation), comme le confirme [Bendisposto *et al.* 2008].

1.3.3 Structure d'un modèle Event-B

Un modèle Event-B est décomposé en deux parties : *le contexte* et *la machine*. Cette séparation permet d'indiquer à une machine donnée les contextes qu'elle « voit ». Un modèle Event-B peut contenir des contextes seulement, des machines seulement ou les deux. Dans le premier cas, le modèle représente une structure mathématique pure. Le deuxième cas représente quant à elle un modèle non paramétré. Le dernier cas représente un modèle paramétré par les contextes. La figure 1.4 présente les deux composants avec leurs différentes clauses telles qu'elles sont déclarées dans la plateforme RODIN.

CONTEXT
  <identifiant_contexte>
EXTENDS
  <liste_identifiant_contextes>
SETS
  <liste_identifiant_ensembles>
CONSTANTS
  <liste_identifiant_constantes>
AXIOMS
  <label>: <prédicat>
  ...
THEOREMS
  <label>: <prédicat>
  ...
END
	MACHINE
	  <identifiant_machine>
	REFINES
	  <identifiant_machine>
	SEES
	  <liste_identifiant_contextes>
	VARIABLES
	  <liste_identifiant_variables>
	INVARIANTS
	  <label>: <prédicat>
	  ...
	THEOREMS
	  <label>: <prédicat>
	  ...
	VARIANT
	  <variant>
	EVENTS
	  <liste_événements>
	END

FIGURE 1.4: La structure d'un modèle Event-B

1.3.3.1 Le contexte

Le contexte contient la partie statique du modèle et il est constitué de plusieurs clauses :

- La clause CONTEXT représente le nom du composant qui devrait être unique dans un modèle.
- La clause EXTENDS déclare la liste des contextes qu'étend le contexte décrit. Un contexte peut étendre un autre contexte en rajoutant de nouvelles constantes et de nouvelles propriétés.
- La clause SETS définit les ensembles porteurs du modèle. Ces ensembles non vides servent à typer le reste des entités du modèle. Notons qu'il est possible de déclarer ces ensembles en indiquant la liste des éléments les constituant (les éléments listés sont distincts les uns des autres).
- La clause CONSTANTS contient la liste des constantes utilisées par le modèle.
- La clause AXIOMS définit les propriétés liées aux constantes et notamment leurs types.
- La clause THEOREMS exprime des propriétés qui peuvent être déduites à partir des propriétés présentées dans la clause AXIOMS.

1.3.3.2 La machine

La machine contient la partie dynamique du modèle et elle est constituée de plusieurs clauses :

- La clause MACHINE représente le nom du composant qui devrait être unique dans un modèle.
- La clause REFINES déclare le nom de la machine raffinée par la machine décrite.
- La clause SEES spécifie la liste des contextes « vus » par la machine. Dans ce cas, la machine peut utiliser les constantes et les propriétés figurant dans les contextes.
- La clause VARIABLES contient la liste des variables du modèle.
- La clause INVARIANTS définit les propriétés d'invariance du modèle telles que des informations sur les types des variables et des propriétés de sûreté.
- La clause THEOREMS exprime des propriétés qui peuvent être déduites des propriétés d'invariance de la machine et des propriétés présentes dans les clause AXIOMS et THEOREMS du contexte vu. En outre, cette clause peut contenir des propriétés que l'on souhaite prouver afin de les employer dans la preuve des invariants du modèle.
- La clause VARIANT définit l'expression du variant du modèle.
- La clause EVENTS contient la liste des événements qui opèrent une ou plusieurs substitutions sur la valeur des variables. Parmi ces événements, l'événement INITIALISATION donne une valeur initiale aux variables.

1.3.4 La notion d'évènement

Un événement Event-B correspond à un changement d'état dénotant une transition dans le système modélisé. Un événement est essentiellement composé d'une garde (la clause WHEN) qui définit les conditions nécessaires au déclenchement de l'évènement et d'une action (la clause THEN) qui définit l'évolution des variables d'état. Notons que plusieurs gardes d'événements peuvent être vraies en même temps. Néanmoins, un seul événement peut se déclencher et le choix de cet événement est non déterministe. Un évènement peut posséder des paramètres (des variables locales) définis dans la clause ANY. En Event-B, on peut distinguer trois formes d'événements :

– *La forme simple* inclut seulement une action. Cela équivaut donc à une garde qui est toujours vraie.

Event *nom* $\widehat{=}$
begin
act : $x :\mid P(x, x')$
end

– *La forme gardée* inclut une garde et une action qui ne dépendent que des variables d'états du modèle.

Event *nom* $\widehat{=}$
when
grd : $G(x)$
then
act : $x :\mid P(x, x')$
end

– *La forme complète* inclut des paramètres, une garde et une action.

Event *nom* $\widehat{=}$
any
prm : t
where
grd : $G(x, t)$
then
act : $x :\mid P(x, x', t)$
end

Il existe aussi un évènement avec une action et une garde vide : l'évènement *skip*. Rappelons qu'il existe un évènement obligatoire, nommé INITIALISATION, qui est toujours de la forme simple. A la différence d'un véritable évènement, cet évènement permet d'initialiser le système en spécifiant les valeurs initiales possibles (qui doivent bien sûr respecter les invariants).

Rappelons également que l'action d'un évènement décrit les changements opérés sur les variables. Comme dans le B classique, ces changements sont exprimés à l'aide du langage de substitutions généralisées. Les substitutions en Event-B ont trois formes possibles :

- *La substitution généralisée* de la forme $x :| P(x,x')$: c'est la forme la plus générale des substitutions où x' représente la valeur de la variable x après la substitution et P est un prédicat. Cette forme exprime que la variable x a une nouvelle valeur x' qui est telle que le prédicat $P(x,x')$ est vrai. L'opérateur utilisé par cette forme est l'opérateur : $|$ (se lit *devient_tel_que*).
- *La substitution ensembliste* de la forme $x :\in E(x)$: cette forme de substitution indique qu'une variable x est modifiée de manière à ce qu'elle devienne un élément d'un ensemble $E(x)$. L'opérateur utilisé par cette forme est l'opérateur $:\in$ (se lit *devient_appartenant_à*). Cette forme de substitution peut être exprimée sous la forme généralisée $x :| (x' \in E(x))$.
- *La substitution simple* de la forme $x := Exp(x)$: cette forme de substitution est représentée par l'opérateur d'affectation où la variable x prend la valeur de l'expression $Exp(x)$. Elle peut être aussi exprimée sous la forme généralisée $x :| (x' = Exp(x))$.

En Event-B, il est possible aussi d'avoir plusieurs *substitutions en parallèle* de la forme $(x :| P(x,x')) \,||\, (y :| Q(y,y'))$. Ces substitutions sont exécutées en même temps mais ne peuvent pas concerner les mêmes variables. La forme généralisée de ces substitutions est $x,y :| (P(x,x') \wedge Q(y,y'))$.

En théorie, un évènement peut se définir comme un prédicat d'une forme particulière appelé prédicat « avant-après » traditionnellement noté prédicat BA (*Before-After Predicate* en anglais). Ce prédicat définit la relation entre les valeurs des variables d'état avant et après l'exécution de l'évènement. Les valeurs *avant* sont dénotées par le symbole de la variable tandis que les valeurs *après* sont dénotées par le symbole des variables suivi du caractère prime. Le tableau 1.2 résume les prédicats BA et les gardes [13] selon la forme des événements.

Forme de l'événement	Le prédicat BA	La garde
Simple	$P(x,x')$	TRUE
Gardée	$G(x) \wedge P(x,x')$	$G(x)$
Complète	$\exists t.((G(t,x) \wedge P(x,x',t))$	$\exists t.G(t,x)$

TABLE 1.2: Les prédicats BA et les gardes des événements.

13. Pour simplifier la notation, les ensembles et les constantes ne sont pas pris en compte dans la présentation des prédicats BA et des gardes.

1.3.5 Le raffinement en Event-B

Le raffinement en Event-B consiste à développer le système de manière incrémentale en partant d'un modèle abstrait qui constitue une spécification du système. A chaque étape de raffinement, des détails du système sont rajoutés graduellement dans un modèle concret qui doit préserver la fonctionnalité et les propriétés des modèles plus abstraits. Ces détails rajoutés apparaissent dans l'état du système en ajoutant des variables et dans le comportement en détaillant les événements de l'abstraction ou en ajoutant de nouveaux événements. Notons que les nouveaux événements raffinent un événement particulier de l'abstraction qui est l'évènement vide (appelé *skip*). Un lien entre les variables d'abstraction et les variables du raffinement est explicitement défini par des *invariants de collage*. Des obligations de preuve sont générées à chaque étape de raffinement afin d'assurer la correction du raffinement.

1.3.6 Les obligations de preuve en Event-B

Un des points les plus importants de la méthode Event-B est la preuve de la correction des modèles. Pour assurer cette correction, un ensemble d'obligations de preuve (notées OP) doivent être déchargées. Ces obligations de preuve concernent différents aspects du modèle tels que la vérification des propriétés d'invariance d'une machine Event-B ou la preuve de la correction d'un raffinement. Dans ce qui suit, nous détaillons les principales obligations de preuve [14].

1.3.6.1 La préservation de l'invariant

L'invariant $I(x)$ est une propriété que le système doit préserver quelle que soit son évolution. Pour cela, chaque déclenchement de tout événement *e* qui a comme prédicat « avant-après » $BA(x,x')$, doit préserver cette propriété en démontrant l'obligation de preuve suivante :

$$\boxed{I(x) \wedge BA(x,x') \Rightarrow I(x')}$$

Pour l'événement INITIALISATION qui a une forme particulière puisque son prédicat BA porte seulement sur des nouvelles valeurs, il faut prouver que sa substitution $x :| init(x')$ implique la correction de l'invariant. L'obligation de preuve correspondante est décrite comme suit :

$$\boxed{init(x') \Rightarrow I(x')}$$

14. Pour des raisons de simplification, les ensembles, les constantes, les axiomes, les théorèmes et les paramètres (les variables locales) ne sont pas pris en compte dans la présentation des différentes obligations de preuve. Néanmoins, ces éléments doivent être considérés dans la pratique.

1.3.6.2 La faisabilité d'une action indéterministe d'un événement

Dans une machine Event-B, chaque événement doit être toujours faisable i.e. la faisabilité impose qu'une nouvelle valeur x' existe effectivement pour chaque événement e ayant une substitution sous la forme généralisée $P(x,x')$ (une action indéterministe). Soit x la valeur de la variable avant le déclenchement de l'événement e qui : (i) satisfait la garde de l'événement $G(x)$; (ii) vérifie l'invariant du système $I(x)$. L'obligation de preuve de faisabilité est décrite comme suit :

$$\boxed{I(x) \wedge G(x) \Rightarrow \exists x'.P(x,x')}$$

1.3.6.3 Les théorèmes

Rappelons que les théorèmes dans les modèles Event-B sont des formules qui peuvent être utiles pour réécrire l'invariant sous une forme spécifique ou démontrer des lemmes. L'obligation de preuve consiste à prouver que ces formules sont déduites à partir de l'invariant de la machine ainsi que l'ensemble des prédicats dans les clauses AXIOMS et THEOREMS du contexte.

1.3.6.4 Le non-blocage

Quand les événements sont gardés, les évènements peuvent ne plus se produire dans certain états, ce qui peut bloquer le système. Il faut vérifier donc que, dans tous les états, il existe au moins un évènement qui peut se déclencher (sa garde est vraie). Cette vérification est faite par un théorème montrant la disjonction des gardes des évènements. Cette obligation de preuve est décrite comme suit :

$$\boxed{I(x) \Rightarrow (G_1(x) \vee ... \vee G_m(x))}$$

Il est important de noter que l'application de cette obligation de preuve n'est pas toujours obligatoire puisque il y a parfois des systèmes où le blocage peut être un comportement désiré par les concepteurs. De plus, Il faut noter que le non-blocage n'est pas conservé par le raffinement, puisqu'on peut renforcer les gardes. Il est ainsi conseillé de le vérifier pour chaque raffinement. Dans ce cas, il suffit de montrer par un théorème que la disjonction de toutes les gardes abstraites $G_1(x),...,G_m(x)$ implique la disjonction de toutes les gardes concrètes $H_1(y),...,H_n(y)$. Cette obligation de preuve est décrite comme suit, sachant que J est l'invariant de collage :

$$\boxed{(I(x) \wedge J(x,y) \wedge (G_1(x) \vee ... \vee G_m(x))) \Rightarrow (H_1(y) \vee ... \vee H_n(y))}$$

1.3.6.5 OP relative au raffinement de l'événement d'initialisation

Comme il n'y a pas d'état avant l'initialisation, la substitution de l'initialisation concrète, notée $y :| init(y')$, attribue une valeur initiale aux nouvelles variables introduites et aux variables conservées du modèle abstrait. L'obligation de preuve consiste à montrer, qu'étant donnée une valeur initiale concrète y', il existe une valeur initiale abstraite x' avec laquelle l'invariant de collage J est vérifié. Cette obligation de preuve est notée comme suit :

$$\boxed{init(y') \Rightarrow \exists x'.(init(x') \wedge J(x',y'))}$$

1.3.6.6 OP relative au raffinement des événements abstraits

Cette obligation de preuve garantit que l'événement abstrait est correctement raffiné par l'événement concret. Supposons que le prédicat BA de l'événement abstrait est noté $BA_A(x,x')$ et que celui de l'événement concret est noté $BA_C(y,y')$. I désigne quant à lui l'invariant abstrait tandis que J désigne l'invariant de collage entre les variables abstraites et concrètes. L'obligation de preuve consiste à montrer qu'à chaque transition concrète, il existe un transition abstraite qui va vers un état valide pour l'invariant de collage. Pour cela, il faut montrer qu'il existe une valeur x' abstraite compatible avec l'évènement abstrait et qui vérifie l'invariant de collage. L'obligation de preuve est décrite donc comme suit :

$$\boxed{I(x) \wedge J(x,y) \wedge BA_C(y,y') \Rightarrow \exists x'.(BA_A(x,x') \wedge J(x',y'))}$$

En pratique, cette obligation de preuve n'est pas présentée comme cela. Comme le prédicat $BA_A(x,x')$ se compose de deux parties (la garde et la substitution), ces deux parties seront séparées pour former deux obligations de preuve distinctes :

- *Le renforcement de la garde* assure que la garde concrète est plus forte que la garde abstraite. Autrement dit, il n'est pas possible d'exécuter la version concrète tandis que la version abstraite ne l'est pas. Le terme « plus fort » signifie que la garde concrète implique la garde abstraite.
- *La simulation d'action* assure que l'événement concret transforme les variables concrètes d'une manière qui ne contredit pas l'événement abstrait. Autrement dit, les actions de l'événement abstrait sont simulées par les actions de l'événement concret.

1.3.6.7 OP relative à l'introduction de nouveaux événements

Si l'événement concret est un nouvel événement raffinant l'événement abstrait *skip*, alors le prédicat BA de l'événement abstrait est tel que $BA_A(x,x') = (x = x')$. Pour cela, l'obligation de preuve précédente (liée au raffinement des événements abstraits) devient comme suit :

$$\boxed{I(x) \wedge J(x,y) \wedge BA_C(y,y') \Rightarrow J(x,y'))}$$

D'un autre côté, l'introduction de nouveaux événements nécessite d'utiliser *un variant* afin de les empêcher de s'exécuter indéfiniment au détriment des événements abstraits. Un variant est une expression ayant pour valeur un entier naturel qui décroit à chaque fois qu'un des nouveaux évènements se déclenche. Cela permet d'assurer que les nouveaux évènements finiront par redonner la main aux événements abstraits. Afin d'assurer que le variant est un entier positif, on doit prouver cette première obligation de preuve :

$$\boxed{I(x) \wedge J(x,y) \Rightarrow V(y) \in \mathbb{N}}$$

La seconde obligation de preuve doit assurer que le variant décroît à chaque exécution d'un nouvel événement :

$$\boxed{I(x) \wedge J(x,y) \wedge BA_C(y,y') \Rightarrow V(y') < V(y)}$$

1.3.7 RODIN : l'outil support d'Event-B

Event-B fournit actuellement un logiciel libre sous la forme d'une plate-forme (dérivée de « Eclipse ») de spécification et de preuve, appelé RODIN. La plateforme RODIN a été initialement développée dans le cadre d'un projet européen, du même nom. La plateforme RODIN stocke les modèles dans une base de données et s'articule sur un ensemble de plug-in permettant par exemple de : (i) prouver les modèles en utilisant les prouveurs de l'Atelier B mais aussi le propre prouveur de RODIN, appelé NewPP ; (ii) vérifier les modèles par les techniques du model-checking en utilisant ProB [Leuschel & Butler 2003] : (iii) animer les modèles en utilisant par exemple BRAMA [Servat 2007] ou ProB. L'intérêt de RODIN est que c'est une plateforme extensible. D'ailleurs, elle est maintenant en cours d'amélioration et d'extension par d'autres plug-in dans le cadre d'un autre projet européen, appelé DEPLOY [15].

15. www.deploy-project.eu/

CHAPITRE 2

Etat de l'art sur le couplage entre les exigences et les méthodes formelles

Sommaire

Rappelons que la transition de la phase des exigences à la phase de spécification formelle est l'une des étapes les plus délicates dans la chaîne de développement formel. Ce chapitre présente différentes approches formelles de la littérature permettant de faciliter cette transition en mettant l'accent sur les approches qui se servent du paradigme GORE.

2.1 Un aperçu de quelques pratiques actuelles non-basées sur GORE

Abrial propose dans le premier chapitre de son livre [Abrial 2010] sa propre méthode qui consiste en la reformulation et la structuration des exigences avant de commencer la spécification formelle. Pour cela, il propose que le cahier des charges distingue le texte explicatif de sa référence. Pour Abrial, le texte explicatif sert initialement pour comprendre le futur système dans la phase d'analyse. C'est la référence au texte qui doit être utilisée dans la suite du cycle de développement logiciel. Abrial propose alors que chaque référence soit exprimée par une courte déclaration écrite en langage naturel, avec une étiquette associée à un nombre spécifique. Les étiquettes servent pour qualifier la nature de l'exigence. Par exemple, FUN désigne les exigences fonctionnelles, ENV pour les exigences d'environnement et SAF pour les exigences de sécurité. S'inspirant de cette notation proposée par Abrial, [Jastram *et al.* 2009, Jastram *et al.* 2010] présentent une approche assurant la traçabilité entre les exigences et les modèles Event-B. Les exigences liées à l'environnement (ENV) sont prises en compte dans Event-B comme étant des structures de données (des ensembles, des constantes et des variables). Les exigences de sécurité (SAF) quant à elles sont considérées comme des invariants Event-B tandis que les exigences de vivacité sont exprimées comme des expressions en logique temporelle linéaire LTL [Pnueli 1977] vérifiables par le model-checker ProB. L'approche utilise aussi les verbes figurant dans les exigences afin de dériver les noms des événements Event-B. Les invariants et les formules LTL guident alors la création des gardes et des actions de ces événements.

[de Sousa & Russo 2010] propose une nouvelle approche montrant comment les cas d'utilisation et les cas de test (exprimées en langage naturel) peuvent guider l'écriture des premières spécifications formelles B. Les cas d'utilisation sont utilisés comme des lignes directrices pour la définition des opérations B, qui incluent les pré- et post-conditions. Les cas de test quant à eux peuvent aider à définir les invariants B. Etant donné qu'il est difficile de considérer l'ensemble du langage naturel, les phrases décrivant les scénarios des cas d'utilisations sont écrites en utilisant un langage naturel contraint, c'est-à dire un sous-ensemble du langage naturel. Dans ce même esprit, [Cabral & Sampaio 2008] présente une approche qui traduit automatiquement les cas d'utilisation (exprimées en langage naturel contraint) vers des spécifications formelles en algèbre de processus CSP [Roscoe *et al.* 1997]. La particularité de l'approche [Cabral & Sampaio 2008] est qu'elle considère différentes vues de cas d'utilisation qui peuvent refléter différents niveaux d'abstraction. C'est pourquoi une notion de raffinement en CSP est définie entre ces différents vues. Ce raffinement est vérifié en utilisant le model-checker de CSP.

Malgré l'apport de ces différents approches et pratiques industrielles, surtout en terme de traçabilité, l'absence d'une méthode d'ingénierie des exigences structurant préalablement les exigences pose divers problèmes. En effet, il est très difficile de choisir à quel niveau de raffinement formel il faut introduire une exigence et

quels liens existent entre les différentes exigences. C'est la raison pour laquelle de nouvelles approches sont proposées ces dernières années qui font appel aux diverses méthodes d'ingénierie des exigences afin d'assurer un passage *plus rigoureux* vers les spécifications formelles. Dans ce chapitre, nous présentons un aperçu de ces travaux en mettant l'accent sur les approches qui utilisent le paradigme GORE pour structurer les buts, y compris les exigences qui sont les buts feuilles dans un modèle de buts GORE. Notons aussi que la plupart des approches présentées se basent sur la méthode KAOS [van Lamsweerde 2009] comme méthode GORE car dans KAOS l'accent est mis sur le raisonnement semi-formel et formel de buts comportementaux (buts fonctionnels) pour la dérivation de raffinements de buts, des opérationnalisations de buts, etc, tandis que dans i* [Yu 1995, Yu 1997] par exemple, l'accent est mis plus sur le raisonnement qualitatif des softgoals.

2.2 L'expression formelle des modèles GORE

Dans ce qui suit, nous présentons deux approches qui ont proposé d'enrichir les modèles GORE avec une étape de formalisation dans le but de prouver et vérifier ces modèles et de faciliter la transition vers la phase de spécification formelle.

2.2.1 La sémantique formelle de KAOS

[Dardenne *et al.* 1993, Darimont & van Lamsweerde 1996, Letier 2001] proposent un ensemble de techniques formelles qui formalisent les heuristiques et les techniques informelles utilisées pour la construction des modèles KAOS. La plupart des concepts KAOS tels que les buts, les opérations, les propriétés du domaine et les hypothèses sont spécifiés formellement en utilisant la logique RT-LTL (la logique LTL avec des constructions temps réel). Les opérateurs temps réel de cette logique sont inspirés de [Koymans 1992], pour la spécification des propriétés impliquant des délais en temps réel.

Cette sémantique formelle de KAOS permet par exemple de vérifier formellement les différentes conditions de raffinement de buts afin de s'assurer que le raffinement est correct et complet (pas de buts qui manquent). Plusieurs techniques sont utilisées pour vérifier l'exactitude d'un raffinement de buts : (i) les prouveurs de théorèmes dédiés pour LTL tels que STeP [Bjørner *et al.* 1996] dans lesquels les formules RT-LTL sont encodées ; (ii) le model-checking borné en utilisant des techniques de SAT-solveur pour déterminer la satisfaisabilité d'une assertion logique ; (iii) l'utilisation d'un catalogue des patrons formels de raffinement de buts [Darimont & van Lamsweerde 1996] qui formalisent en RT-LTL les patrons informels de raffinement de buts. Cette dernière technique a été largement adoptée puisqu'elle permet de guider le raffinement de buts, tout en cachant les détails formels. En fait, l'intérêt de cette technique est qu'un patron est prouvé une fois pour toute en utilisant par exemple le prouveur STeP. Ce patron peut être ensuite réutilisé dans des situations de correspondance à travers l'instanciation de

ses méta-variables, qui sont liées aux conditions de l'application du patron. De même, la preuve d'un raffinement instancié n'est que l'instanciation de la preuve générique correspondante. Etant donné le succès des patrons formels de raffinement de buts, [Letier 2001] propose d'étendre la sémantique formelle de KAOS en présentant un catalogue de patrons formels d'opérationnalisation qui permettent de vérifier que les opérationnalisations des buts feuilles (les exigences) sont correctes.

[Ponsard *et al.* 2007] présente l'outil FAUST qui permet d'automatiser la vérification des raffinement de buts basée sur des techniques de model-checking. FAUST offre aussi la possibilité aux utilisateurs du futur système de valider leurs exigences à travers l'animation du modèle des exigences KAOS.

2.2.2 Formal Tropos

En plus de sa représentation graphique, Tropos fournit un langage de spécification formelle, dit Formal Tropos [Fuxman *et al.* 2001]. Ce langage utilise une notation textuelle pour les modèles i* et permet la description de contraintes dynamiques entre les différents éléments de la spécification en logique temporelle LTL. Formal Tropos possède ainsi une sémantique permettant une analyse formelle. Un outil, appelé T-tool [Fuxman *et al.* 2001], est développé afin de supporter cette analyse. Cet outil est basé sur le model-checker symbolique NuSMV [Cimatti *et al.* 2000]. L'outil traduit automatiquement une spécification Formal Tropos dans une spécification de langage intermédiaire qui fait le lien entre Formal Tropos et des différents moteurs de vérification. La représentation en langage intermédiaire est par la suite automatiquement traduite en NuSMV, qui peut alors effectuer différents types d'analyse formelle, tels que la vérification des propriétés du système et l'animation de la spécification.

2.2.3 Discussion

L'étape de formalisation des modèles GORE est importante car elle permet de prouver ces modèles en détectant par exemple des buts manquants, des buts inutiles, etc. Néanmoins, ces approches se basent sur la technique de model-checking qui souffre du problème de l'explosion du nombre d'états explorés. D'un autre côté, cette formalisation ne peut pas combler le fossé entre la phase d'analyse des exigences et les autres phases de développement dans la mesure où elle se base sur l'utilisation de la logique temporelle LTL. LTL n'est pas une méthode formelle concrète et les concepteurs sont donc obligés d'utiliser une autre méthode formelle plus concrète (la méthode B, Event-B, etc.) pour développer leurs systèmes. Cette non-utilisation de la même méthode formelle dans les différentes phases de développement ne va pas faciliter par la suite la tâche de validation des spécifications formelles par rapport aux exigences initiales bien qu'elles aient été déjà formalisées avec LTL.

2.3 Dérivation des spécifications formelles à partir du modèle des opérations KAOS

Dans cette section, nous présentons trois approches qui proposent de dériver des spécifications formelles en se basant sur le modèle des opérations KAOS.

2.3.1 De KAOS vers B

[Ponsard & Dieul 2006] présente une approche orientée buts qui définit le lien entre le modèle des exigences KAOS et la phase de spécification. Ce lien consiste à dériver des machines B abstraites à partir des différents modèles KAOS. Pour cela, les auteurs associent une machine B à chaque agent KAOS puisque les agents sont les entités actives qui permettent de réaliser les opérations. Ainsi, toutes les opérations KAOS (avec leurs pré-conditions et post-conditions) pour un agent donné sont représentées en B comme des opérations de la machine correspondante. Les attributs de l'agent et les arguments des opérations sont représentés en B dans les clauses SETS, VARIABLES et CONSTRAINT. L'information qui sert à l'initialisation de la machine B est extraite à partir du graphe de raffinement de buts KAOS. De plus, tous les buts « Maintain » (à assurer toujours) qui sont sous la responsabilité d'un agent sont vus comme des invariants pour la machine correspondant à cet agent. Les auteurs soulignent enfin que les machines B obtenues peuvent être par la suite raffinées en B.

2.3.2 L'approche FADES

[Hassan 2009] présente FADES, acronyme de « *Formal Analysis and Design for Engineering Security* », qui dérive des machines B abstraites à partir d'un ensemble d'exigences de sécurité modélisées à l'aide d'une extension de KAOS dédiée pour la sécurité [Hassan 2009]. Ces machines B abstraites sont ensuite raffinées en utilisant le mécanisme de raffinement B. Le but final de FADES est de produire des implémentations conformes aux exigences initiales grâce à l'utilisation de techniques de test. Un aperçu général de l'approche FADES décrit dans la Figure 2.1.

L'étape cruciale dans FADES est la transformation des exigences KAOS vers une spécification abstraite B. En fonction de la taille du système, le modèle des exigences KAOS est traduit vers une ou plusieurs machines abstraites B. Pour cela, [Hassan 2009] associe une machine B à chaque objet KAOS concerné par les buts opérationnels. De cette façon, la relation en KAOS entre les différents objets est traduite en B en utilisant les clauses de liaison entre les machines tel que INCLUDES, USES ou SEES. Les attributs de l'objet sont considérés comme des variables et leurs types comme des invariants de typage dans la machine B correspondante. L'invariant du domaine lié à l'objet KAOS (exprimé en logique du premier ordre) est vu comme comme un invariant dans la machine B. De plus,

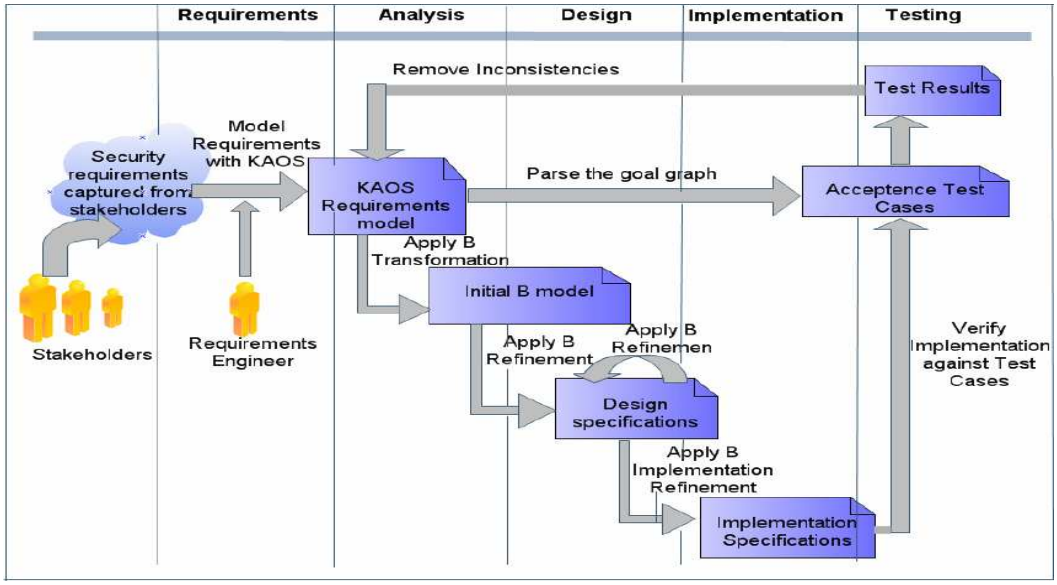


FIGURE 2.1: Un aperçu général de l'approche FADES [Hassan 2009]

chaque machine B représentant un objet comprend aussi un ensemble représentant toutes ses instances. Ainsi, les attributs des objets sont représentées comme des relations entre l'ensemble des instances et les types de l'attribut.

Chaque opération KAOS manipulant l'objet est représentée comme une opération B et utilise les objets KAOS soit comme paramètres ou comme des valeurs de retour. La pré-condition de cette opération KAOS (exprimée en logique du premier ordre) est considérée comme une pré-condition B de l'opération B correspondante. Il est ensuite de la responsabilité du concepteur de remplir le corps de chaque opération tout en prenant en considération la post-condition de cette opération KAOS. Les propriétés du domaine KAOS sont exprimées en B comme des invariants pour les machines B modélisant les objets KAOS et comme des pré-conditions pour les opérations modélisant les opérations KAOS.

Une fois la transformation vers B achevée, les machines B abstraites sont raffinées jusqu'à l'implémentation. Pour assurer la conformité des implémentations obtenues vis-à-vis des exigences initiales, FADES dérive un ensemble de cas de test d'acceptation à partir du modèle de buts. Ces cas de test décrivent les différents scénarios de comportement de sécurité qui doivent être respectés par les spécifications B raffinées. Cette étape de vérification supplémentaire fournie par FADES permet de détecter des incohérences qui pourraient exister dans le modèle des exigences ou dans le processus de raffinement B. Les auteurs affirment que les deux principales sources de problèmes rencontrés dans l'implémentation sont des incohérences soit au niveau du modèle des exigences KAOS ou dans les décisions de conception réalisées au cours des étapes de raffinement B.

L'approche [Hassan 2009] a été outillée en exportant tout d'abord en format XML le modèle des exigences KAOS créé à l'aide de l'outil propriétaire Objectiver, comme le montre la Figure 2.2. Puis, un analyseur du graphe de buts a été

implémenté en Java afin d'analyser le code XML du modèle KAOS et d'extraire les informations nécessaires pour construire le modèle initial B et les cas de test d'acceptation. L'analyseur produit ainsi deux artefacts : (i) le modèle initial B représentant le modèle des exigences ; (ii) les cas de test d'acceptation. Il est ensuite de la responsabilité du concepteur de remplir manuellement le corps de chaque opération tout en prenant en considération la post-condition de cette opération KAOS. L'outil développé fait appel enfin à l'outil B-Toolkit afin de raffiner les machines B abstraites jusqu'à arriver à l'implémentation.

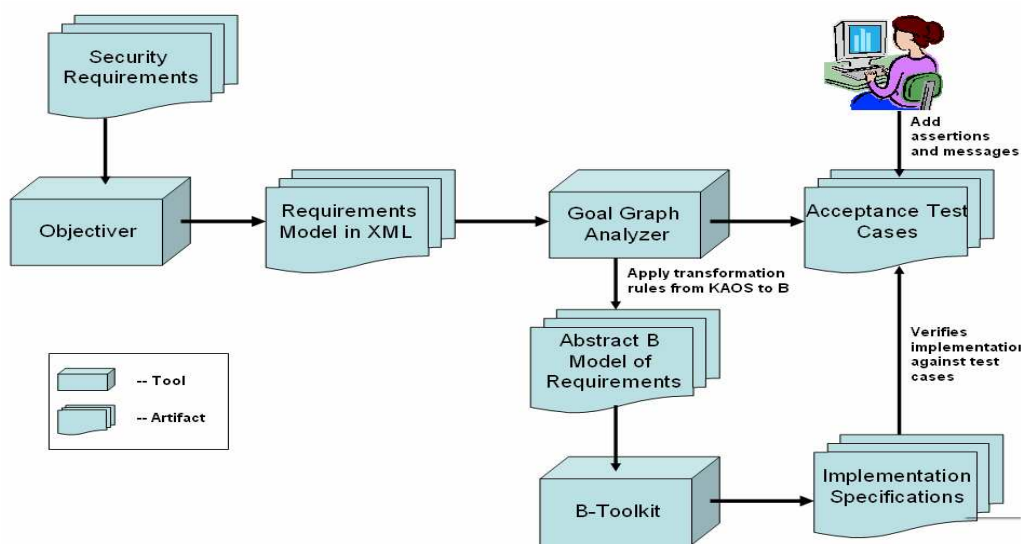


FIGURE 2.2: Un aperçu général de l'outil implémentant FADES [Hassan 2009]

2.3.3 De KAOS vers VDM++

[Nakagawa *et al.* 2007] propose une approche pour transformer le modèle des exigences KAOS en spécifications formelles VDM++ [Fitzgerald *et al.* 2005]. Cette approche connecte les opérations KAOS (qui opérationnalisent les exigences KAOS) à celles de VDM++ et les entités KAOS aux objets ou types VDM++. Un aperçu de l'approche [Nakagawa *et al.* 2007] est illustré dans la Figure 2.3. L'approche incite les concepteurs à travers des commentaires spécifiques à compléter manuellement les spécifications VDM++ au cas où certaines informations manquent au niveau du modèle des exigences KAOS. De plus, ces spécifications formelles générées contiennent juste des opérations implicites constituées d'entrées/sorties, les pré-conditions et les post-conditions. Afin de créer les opérations explicites, les concepteurs doivent ajouter manuellement le corps des opérations c'est-à-dire les algorithmes décrivant le fonctionnement des opérations. Les développeurs préparent aussi des cas de test afin de vérifier les spécifications formelles à l'aide des outils VDM.

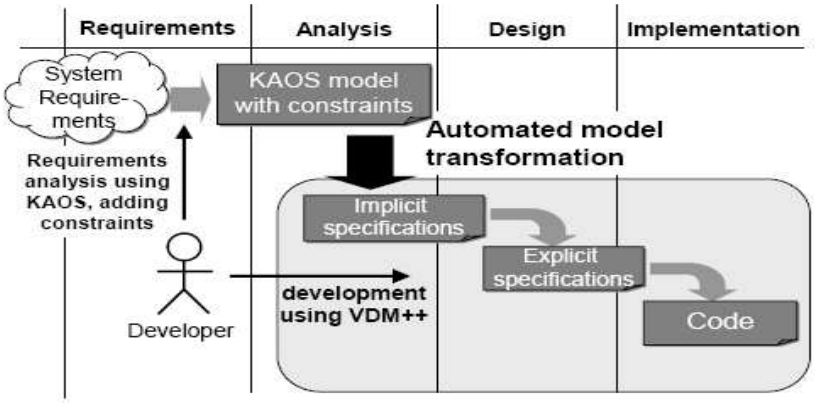


FIGURE 2.3: Un aperçu de l'approche générant des spécifications VDM++ [Nakagawa *et al.* 2007]

L'approche [Nakagawa *et al.* 2007] est mise en œuvre en tant qu'un générateur automatique sous la forme d'un plug-in Eclipse. Comme le montre la Figure 2.4, le générateur accepte en entrée un code XML (représentant le modèle des exigences KAOS) généré par l'outil Objectiver. Il produit en sortie trois types de fichiers Classes qui sont les blocs de construction de modèles VDM++ : (i) *la classe de définition commune* qui se compose des types utilisés dans tous les fichiers de spécification sachant que les types correspondent à des entités n'ayant pas d'attributs dans le modèle KAOS ; (ii) *la classe objet* qui représente un objet passif sachant que ces objets sont des entités qui possèdent des attributs dans le modèle KAOS ; (iii) *la classe système* qui est composée des variables contrôlées par un seul système (un agent dans le modèle KAOS) et par ses opérations.

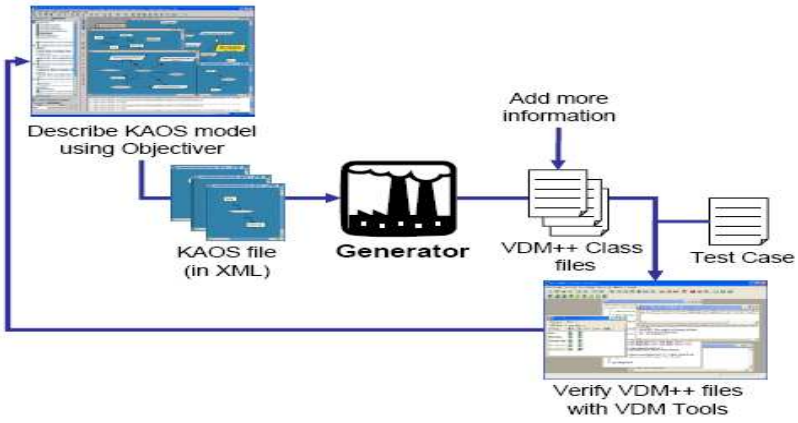


FIGURE 2.4: Un aperçu de l'outil implémentant l'approche [Nakagawa *et al.* 2007]

2.3.4 Discussion

Nous avons présenté trois approches qui proposent d'assurer un passage vers les spécifications formelles directement à partir des opérations KAOS. Ces travaux ne tiennent pas compte du modèle de buts et se sont plutôt concentrés sur les opérations qui opérationnalisent les buts. Avec de telles approches, on dérive directement des modèles formels concrets qui répondent à la question « comment peut-on réaliser le futur logiciel ? » c'est-à-dire que les modèles formels décrivent les solutions. Or, on sait que les langages formels basés sur le raffinement répondent aussi à la question « que fait le système ? ». D'un autre côté, ces travaux partent de l'hypothèse que le modèle des opérations KAOS est déjà prouvé correct et complet. Pourtant, l'approche KAOS est à la base informelle et il est fort probable qu'il y ait des erreurs telles que des opérations incorrectes ou manquantes. Pour résoudre ce problème, nous pensons qu'une activité de preuve sur l'ensemble du modèle KAOS est nécessaire avant de commencer toute dérivation des modèles formels. Pour cela, ces travaux doivent utiliser : (i) l'outil FAUST [Ponsard *et al.* 2007] afin de vérifier les raffinement des buts KAOS ; (ii) les catalogues de patrons formels d'opérationnalisation [Letier 2001] qui permettent de vérifier que les opérationnalisations des buts feuilles sont correctes. Sans l'utilisation conjointe de ces deux techniques, il semble difficile de détecter par la suite dans le modèle formel des oublis d'exigences par exemple. De plus, une étape d'animation au niveau de la spécification est nécessaire pour détecter très tôt des incohérences dans les spécifications formelles vis-à-vis des exigences initiales.

2.4 Une spécification formelle guidée par les buts

Pour surmonter les limites des approches guidées par les opérations, d'autres approches sont proposées par la littérature. Ces approches se servent des modèles de buts pour dériver les modèles formels.

2.4.1 La méthode GOPCSD

[El-Maddah & Maibaum 2003] propose la méthode outillée GOPCSD, acronyme de « *Goal-Oriented Process Control System Design* » dont l'une des fonctionnalités est la génération de spécifications formelles B à partir d'un modèle de buts. Bien que cette méthode soit basée sur KAOS, quelques légères adaptations de la méthode KAOS ont été faites afin de prendre en compte la nature des systèmes de contrôle de processus. Comme le montre la Figure 2.5, l'outil GOPCSD commence d'abord par la construction d'un modèle de buts complet en rassemblant les différents sous-modèles de buts et en utilisant aussi différentes entités (composants, agents, variables et leurs types). Puis, l'outil reprend la démarche KAOS afin de prouver la cohérence et la complétude du modèle construit en résolvant aussi les conflits et les obstacles et en ajoutant de nouvelles exigences manquantes. Enfin,

GOPCSD traduit le modèle de buts obtenu vers des spécifications B en utilisant les diagrammes d'états-transitions (STD) comme représentation intermédiaire. En fait, les auteurs de [El-Maddah & Maibaum 2003] considèrent que les buts sont équivalents aux transitions STD et que les états STD sont équivalents aux pré-conditions et post-conditions relatives aux buts. Dans GOPCSD, les buts opérationnels sont représentés par des transitions directes STD tandis que les buts abstraits sont représentés par des transitions indirectes. En utilisant l'approche définie par Sekerinski [Sekerinski 1998], les diagrammes STD obtenus sont ensuite transformés dans des machines B définissant des opérations B. De plus, d'autres informations provenant directement du modèle de buts construit sont utilisées pour générer des machines B décrivant les variables et leurs typages. Ces spécifications B obtenues peuvent être ensuite raffinées et traduites en code exécutable à l'aide des outils B.

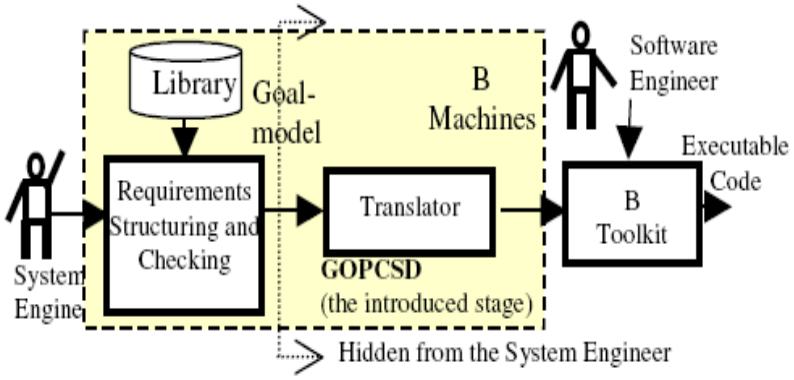


FIGURE 2.5: Un aperçu de la méthode GOPCSD [El-Maddah & Maibaum 2003]

2.4.2 De i* vers Z

[Krishna *et al.* 2004] présente une méthode pour une co-évolution entre les modèles i* et les schémas Z. La première étape de la méthode consiste à définir un mapping entre les diagrammes i* et les spécifications Z. Cette première étape permet de formaliser en Z les éléments des diagrammes i* tels que les buts et les agents, sans ajouter ou supprimer d'informations. La deuxième étape de la méthode est le raffinement des modèles abstraits Z en considérant d'autres informations cachées (obtenues en analysant les diagrammes i*) telles que les propriétés d'invariance. La spécification Z obtenue après ces raffinements est appelée spécification Z étendue. La méthode fournit aussi des lignes directrices pour ajouter toutes les modifications faites sur les diagrammes i* dans le modèle abstrait Z. En fait, les modifications faites sur les diagrammes i* seront propagées dans la spécification Z étendue pour obtenir une nouvelle spécification Z étendue.

2.4.3 De KAOS vers Event-B [Aziz *et al.* 2009]

Récemment, [Aziz *et al.* 2009] a présenté une approche basée sur la vérification constructive qui consiste à établir des liens entre les exigences formalisées (exprimées comme des formules en logique RT-LTL) et une spécification de système, exprimée comme une machine Event-B étendue avec la notion d'obligations [Bicarregui *et al.* 2008]. C'est une approche d'opérationnalisation qui consiste à dériver une machine abstraite Event-B à partir des exigences KAOS en suivant ces étapes : (i) transformer l'état des objets KAOS dans l'état Event-B grâce à l'utilisation de UML-B [Snook & Butler 2008] ; (ii) dériver les événements Event-B à partir des formules RT-LTL associées aux exigences KAOS ; (iii) compléter la machine Event-B avec la partie initialisation et d'autres événements ; (iv) vérifier enfin que la machine Event-B est une opérationnalisation correcte des exigences KAOS.

Les auteurs de [Aziz *et al.* 2009] présentent trois patrons d'opérationnalisation en Event-B afin d'aider le concepteur à dériver une spécification abstraite Event-B à partir des buts feuilles *Achieve* exprimées en RT-LTL. Pour cela, un travail préalable réalisé par les mêmes auteurs dans [Bicarregui *et al.* 2008] a été nécessaire afin d'étendre Event-B avec la notion d'obligation. Cette nécessité est due au fait que l'expression formelle des buts KAOS en RT-LTL définit généralement non seulement un ensemble maximal de comportements autorisés, mais aussi un ensemble minimal de comportements obligés. En fait, quand la garde d'un événement Event-B est vraie, il n'y a pas d'obligation d'exécuter l'événement et son exécution peut être retardée en raison, par exemple, d'un entrelacement avec d'autres événements autorisés. Le choix de l'ordonnancement de ces événements autorisés est non-déterministe. Dans ce contexte, [Bicarregui *et al.* 2008] propose de modéliser les obligations en Event-B comme étant des événements avec des déclencheurs (*Triggers* en anglais) exprimés en logique du premier ordre. Les déclencheurs sont les dualités des gardes puisque lorsqu'une garde est fausse, l'événement ne peut pas se produire tandis que lorsqu'un déclencheur est vrai, l'événement doit se produire.

En utilisant cette notion de déclencheur, les trois patrons d'opérationnalisation des exigences en Event-B [Aziz *et al.* 2009] sont présentés ci-dessous. C et S désignent les formules logiques du premier ordre définies sur l'espace des objets KAOS. Dans la machine Event-B, ces objets sont transformés dans des variables : (i) $\overline{C}$ est la formule correspondante à C sur l'état Event-B ; (ii) $\overline{S}$ est la substitution généralisée provenant du prédicat S, où S est considérée comme la post-condition de la substitution.

– Le patron d'opérationnalisation *Bounded Generation :* Si une exigence KAOS a la définition formelle suivante : $C \Rightarrow \Diamond_{\leq n} S$
Alors, l'événement Event-B correspondant prend cette forme :

$$\boxed{\text{EVENT } e \text{ WHEN } \overline{C} \text{ WITHIN } n \text{ NEXT } \overline{S} \text{ END}}$$

Cet événement exprime que lorsque la condition de déclenchement $\overline{C}$ devient vraie, l'événement e devra être exécuté dans les n prochaines étapes

(une étape correspond à l'exécution d'un événement). Cela nécessite que la condition de déclenchement reste toujours vraie jusqu'à l'exécution de l'événement. Si la condition de déclenchement devient faux au cours de ces étapes (avant l'exécution de l'événement), l'obligation d'exécuter l'événement est annulée.

– Le patron d'opérationnalisation *Immediate Generation :* Si une exigence KAOS est sous la forme : $C \Rightarrow \circ S$
Alors, l'événement Event-B correspondant est sous cette forme :

$$\boxed{\text{EVENT } e \text{ WHEN } \overline{C} \text{ NEXT } \overline{S} \text{ END}}$$

Cette notation, qui est un cas particulier des événements WITHIN (dans le cas où n = 0), exprime qu'à chaque fois que $\overline{C}$ devient vraie, alors l'événement e sera le prochain à être exécuté.

– Le patron d'opérationnalisation *Eventually Generation* Si une exigence KAOS est exprimée comme suit : $C \Rightarrow \Diamond S$
Alors, l'événement Event-B correspondant est notée sous cette forme :

$$\boxed{\text{EVENT } e \text{ WHEN } \overline{C} \text{ EVENTUALLY } \overline{S} \text{ END}}$$

Cette notation est un cas particulier des événements WITHIN dans le cas où le choix du nombre n est non-borné et non-déterministe. Cette notation exprime que lorsque $\overline{C}$ devient vraie, alors l'événement e sera un jour exécuté.

Il est important de noter que ces différents événements déclenchés n'ont pas de gardes. En fait, les auteurs de [Bicarregui *et al.* 2008] considèrent que la garde est la même que l'élément déclencheur puisque l'événement est déclenché lorsqu'il est permis. L'application de ces différents patrons d'opérationnalisation permet d'obtenir une machine Event-B qui est une spécification abstraite d'un système répondant aux exigences KAOS. Une fois complétée par l'initialisation et d'autres événements, cette machine peut être raffinée jusque à l'implémentation selon le raffinement classique d'Event-B. Les formules temporelles associées aux exigences KAOS sont par la suite incluses comme des assertions de vérification qui peuvent être vérifiées par des outils comme ProB [Leuschel & Butler 2003] dans le but de montrer que la spécification proposée respecte bien toutes les exigences.

2.4.4 De KAOS vers Event-B [Ponsard & Devroey 2011]

[Devroey 2010, Ponsard & Devroey 2011] proposent une nouvelle approche pour assurer la traçabilité entre les exigences et la spécification formelle en s'appuyant sur la correspondance entre les agents KAOS et les machines Event-B et en utilisant UML-B [Snook & Butler 2006] et la technique de la décomposition en Event-B [Abrial 2010]. La Figure 2.6 présente un aperçu de l'approche.

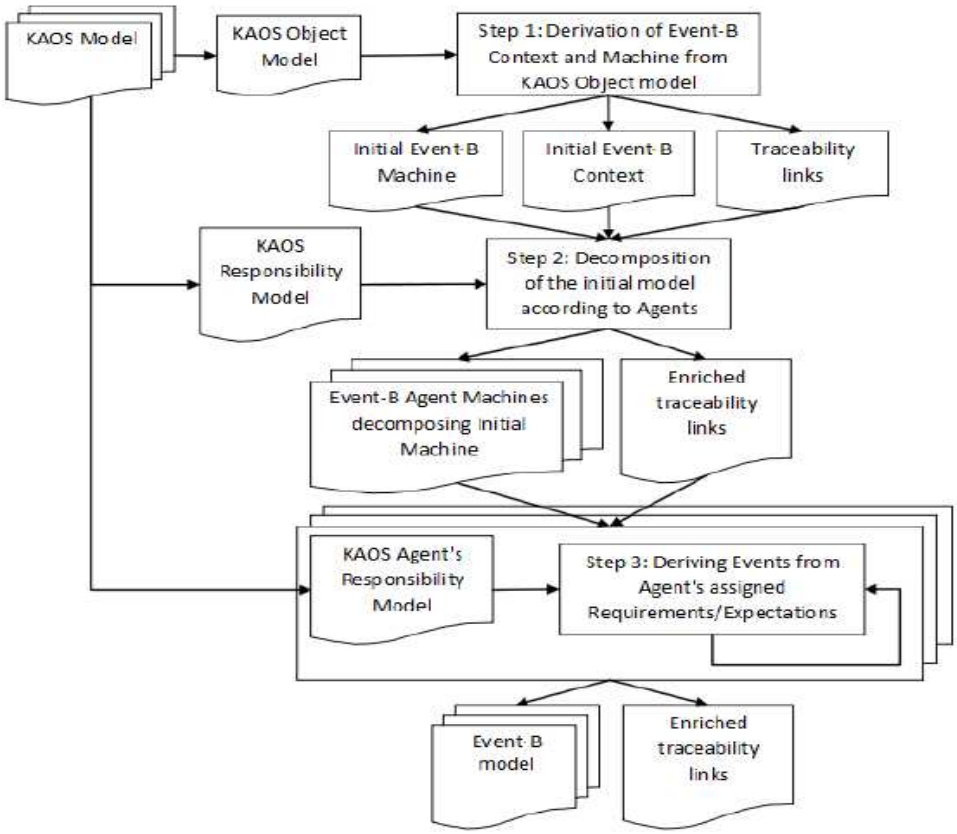


FIGURE 2.6: Un aperçu de l'approche [Devroey 2010]

Comme le modèle objet KAOS définit tous les concepts utilisés pour la définition des buts dans le modèle de buts, l'approche proposée par [Devroey 2010] consiste à créer dans une première étape une machine et un contexte Event-B initiaux afin de représenter l'ensemble du modèle objet KAOS. Puisque le modèle objet KAOS correspond à un diagramme de classes UML simplifié, les auteurs reprennent les règles de transformation présentées en UML-B [Snook & Butler 2006] pour transformer en B tous les concepts d'un diagramme de classes UML (objets, attributs, associations). En plus des variables et des invariants créés pour représenter les différents éléments provenant du modèle objet KAOS, la première étape de l'approche consiste à ajouter un ensemble d'événements abstraits dit *événements update* dans la machine Event-B initiale afin de représenter le fait que les éléments KAOS peuvent évoluer dans le temps. Ainsi, chaque élément du modèle objet KAOS sera traduit en Event-B par une ou plusieurs variables, un ou plusieurs invariants et un seul événement *update* qui représente la mise à jour de l'élément ; c'est-à-dire que l'action de l'événement *update* met à jour les variables Event-B correspondant à l'élément KAOS. Par conséquent, la machine Event-B créée est l'équivalent du système entier, capable de contrôler toutes les données.

Dans la deuxième étape, la machine Event-B initiale obtenue à l'issue de la première étape est décomposée en plusieurs machines « agent » en utilisant la

décomposition basée sur les états [Abrial 2010] ; connue aussi sous le nom de A-Style [1]. L'auteur propose un algorithme associant une machine « agent » à chaque agent KAOS en se basant sur sa capacité de contrôler certaines données. Chaque machine « agent » contient un ensemble de variables, des invariants et *des événements update* provenant de la machine Event-B initiale. Ces variables, invariants et événements correspondent aux éléments du modèle objet KAOS que l'agent KAOS contrôle ou manipule. Les événements placés dans chaque machine « agent » peuvent être partitionnés en deux types :

- Si l'agent KAOS correspondant à la machine « agent » contrôle l'élément du modèle objet KAOS, *l'événement update* correspondant à cet élément est marqué comme *un événement interne* puisque c'est cet agent en question qui est responsable de l'implémentation concrète de cet événement.
- Si l'agent KAOS manipule juste l'élément sans le contrôler, *l'événement update* correspondant à cet élément est marqué comme *un événement externe*. En fait, l'agent KAOS en question dans ce cas n'est pas responsable de l'implémentation concrète de cet événement.

Ainsi, chaque *événement update* dans la machine initiale est répliqué comme suit : (i) *un événement interne* dans une et une seule machine « agent » ; (ii) *un événement externe* dans zéro, une ou plusieurs machine « agent ». Chaque machine Event-B « agent » a maintenant une liste de variables partagées avec des invariants liés à ces variables et une liste des *événements update* qui représente l'évolution dans temps de ces variables.

La dernière étape de l'approche consiste à créer un raffinement Event-B pour chaque machine « agent » où seuls *les événements internes* sont raffinés à chaque fois. Les événements externes, qui seront raffinés dans d'autres machines « agent », assurent avec les variables partagées la communication entre les différents machines « agent ». Le raffinement de chaque machine « agent » est fait en considérant à chaque fois la traduction en Event-B des comportements déclarées dans les exigences et les attentes placées sous la responsabilité de l'agent KAOS en question. Chaque exigence et attente sous la responsabilité de l'agent KAOS est traduite dans la machine « agent » de raffinement par un ou plusieurs événements, avec éventuellement de nouvelles variables et de nouveaux invariants.

Les deux premières étapes de l'approche ont été implémentées dans un outil qui utilise les technologies de transformation de modèles telles que EMF [Steinberg *et al.* 2009] et ATL [Bézivin *et al.* 2003]. Le méta-modèle source utilisé est le méta-modèle KAOS défini dans l'outil propriétaire Objectiver. Cependant, la dernière étape de l'approche n'est pas automatique et s'appuiera sur les compétences du concepteur qui assure le passage de KAOS à Event-B.

1. La lettre A fait référence à Abrial.

2.4.5 Discussion

Cette catégorie d'approches propose d'assurer la correspondance avec les modèles formels en se basant sur les buts. Néanmoins, ces travaux restent partiels parce qu'ils ne considèrent pas toutes les parties du modèle de buts, mais seulement les buts opérationnels, c'est-à-dire les buts feuilles (les exigences). Par conséquent, le modèle formel n'inclut aucune information sur les buts non-feuilles (les buts abstraits) et surtout sur le type de raffinement de buts. Pourtant, les buts abstraits et les différents raffinements donnent une justification aux exigences et constituent d'ailleurs les principaux apports de l'approche GORE. De plus, le type de raffinement de but permet de spécifier des liens d'ordre entre les différentes exigences. Ainsi, le fait de ne pas inclure ces informations importantes dans le modèle formel peut compliquer la tâche du concepteur par la suite. Comme pour les approches guidées par les opérations, nous pensons qu'une activité de preuve sur le modèle de buts en utilisant par exemple FAUST [Ponsard *et al.* 2007] est nécessaire avant de commencer toute dérivation des modèles formels. En fait, il se peut que le modèle de buts ne soit pas correct et complet ; c'est-à-dire il y a des exigences qui manquent ou qui sont inutiles par exemple. Notons aussi que la majorité de ces travaux qui utilise Event-B, ne tirent pas profit de la possibilité qu'offre Event-B de décrire un système à un niveau d'abstraction élevé et d'introduire petit à petit des détails grâce au raffinement.

2.5 Conclusion

Dans ce chapitre, nous avons présenté des travaux de couplage entre la phase d'analyse des exigences et la phase de spécification et nous avons mis l'accent sur les travaux qui s'appuient sur le paradigme GORE. Le tableau 2.1 présente un récapitulatif des différents points communs et spécificités des principales approches de dérivation de spécifications formelles à partir des modèles GORE. La première remarque est que la plupart des approches ne considèrent pas les buts abstraits et le type de raffinement de buts au moment de la spécification formelle. En fait, la majorité des travaux considèrent que le modèle des buts a déjà été validé, d'une manière ou d'une autre. Une autre remarque importante concerne la prise en compte des exigences non-fonctionnelles. Hormis [Hassan 2009], les autres approches n'ont pas considéré explicitement les NFRs au moment de la dérivation des modèles formels. Cela paraît évident dans la mesure où toutes les approches se basent sur KAOS, qui ne sépare pas les exigences fonctionnelles et non-fonctionnelles. Ces approches se sont plutôt focalisées sur les buts fonctionnels bien que quelques approches aient pris en compte les buts KAOS « Maintain », qui peuvent être toutefois considérés comme des NFRs. Ces quelques approches construisent ainsi des modèles formels où les exigences fonctionnelles et non-fonctionnelles sont étroitement liées. Le tableau 2.1 montre aussi que la moitié des approches ne définissent pas de règles de traçabilité explicites entre les exigences et les modèles formels. D'un point de vue

outillage, nous remarquons qu'il y a quelques approches théoriques qui ne sont pas outillées et même les outils qui existent se basent sur l'outil propriétaire Objectiver.

	Langage cible	Buts abstraits	Buts feuilles	Opérations	NFRs	Traçabilité explicite	Outillage
[Ponsard & Dieul 2006]	B			✓			
[Hassan 2009]	B			✓	✓	✓	✓
[Nakagawa *et al.* 2007]	VDM++			✓		✓	✓
[El-Maddah & Maibaum 2003]	B	✓	✓				✓
[Aziz *et al.* 2009]	Event-B		✓				
[Ponsard & Devroey 2011]	Event-B		✓			✓	✓

TABLE 2.1: Points communs et divergences entre les différentes approches de dérivation de spécifications formelles.

En se basant sur ces remarques et ces limites, les travaux de notre thèse visent à proposer une approche complémentaire aux approches existantes. Tandis que les approches existantes dérivent les modèles formels en restant au niveau de la spécification, notre objectif est de remonter le processus de développement formel jusqu'à la phase d'analyse des exigences. Cela va aider à dériver les premières spécifications formelles ; c'est-à-dire les modèles formels qui sont plus abstraits que ceux dérivés par les approches existantes. Pour cela, l'idée consiste à explorer le modèle de buts SysML/KAOS en entier, tout en mettant l'accent sur la séparation entre les buts fonctionnels et non-fonctionnels. Nous proposons tout d'abord deux approches différentes qui se servent des buts fonctionnels pour dériver les spécifications formelles abstraites : une approche dédiée au B classique et une autre dédiée à Event-B. Le choix s'est porté sur ces deux méthodes formelles (B et Event-B) pour les raisons suivantes : (i) elles partagent avec SysML/KAOS une caractéristique importante : la notion de raffinement ; (ii) la plupart des approches existantes proposées par la littérature (voir le tableau 2.1) reposent sur ces deux méthodes formelles, ce qui permet d'assurer une éventuelle complémentarité avec ces approches. La thèse se focalise par la suite sur l'approche dédiée à Event-B afin de la compléter et l'enrichir par la prise en compte des buts non-fonctionnels. Des liens de correspondance sont également définis afin de faciliter surtout la gestion de l'évolution des NFRs. La thèse propose enfin un logiciel libre, nommée SysKAOS2EventB, construit à partir de logiciels libres disponibles (RODIN et TOPCASED).

Deuxième partie

Contribution

CHAPITRE 3

Une approche pour la dérivation d'une architecture B à partir d'un modèle de buts fonctionnels SysML/KAOS

Sommaire

Nous proposons dans ce chapitre une première approche permettant de dériver l'architecture de la spécification formelle B à partir d'un modèle de buts fonctionnels SysML/KAOS en entier. L'architecture obtenue permet d'établir des liens de correspondance entre les buts fonctionnels SysML/KAOS et les opérations B qui décrivent ces buts. Notons que l'approche présentée dans ce chapitre expose les travaux déjà publiés dans nos articles [Matoussi & Petit 2010, Laleau *et al.* 2010].

3.1 Les règles de correspondance entre des modèles de buts et des spécifications B

L'idée principale de notre approche est de construire l'architecture de la spécification B à partir d'un modèle de buts fonctionnels SysML/KAOS en entier. Cela consiste à définir ce qu'une machine B contient et les liens entre les différentes machines B, comme le montrent les règles de traduction décrites dans les sous-sections suivantes. La combinaison de toutes ces règles permet d'obtenir cette architecture.

3.1.1 Traduction d'un but fonctionnel SysML/KAOS en B

Rappelons (voir la section 1.2.7.1) qu'un but fonctionnel « Achieve » prescrit des comportements attendus où une certaine condition cible doit être tôt ou tard établie quand une autre condition est vérifiée dans l'état actuel du système. Un but « Achieve » dans SysML/KAOS est dénoté comme suit : *Achieve[**TargetCondition** From **CurrentCondition**]*, sachant que le préfixe ***CurrentCondition*** est optionnel.

Nous proposons d'associer une machine B M_G à chaque but G. Cette machine M_G contient une opération appelée OP_G qui décrit le but G, en termes de substitutions généralisées : (i) la clause optionnelle **CurrentCondition** est traduite par une pré-condition (notée P) dans l'opération OP_G et éventuellement des paramètres d'entrée ; (ii) la clause obligatoire **TargetCondition** est traduite par une substitution (notée S) dans l'opération OP_G et éventuellement des paramètres de sortie. Ces derniers doivent être typés en leur donnant une valeur dans S. Les paramètres d'entrée quant à eux doivent être typés dans P. Rappelons (voir le tableau 1.1) qu'une substitution S en langage B peut prendre plusieurs formes.

3.1.2 Traduction d'un raffinement SysML/KAOS en B

Comme le montre la Figure 3.1, chaque raffinement du but G est représenté par une machine raffinement B $Ref M_G$ qui raffine la machine M_G ; c'est-à-dire que l'opération abstraite OP_G est raffinée par une opération concrète appelée $Ref OP_G$ [1].

Le problème maintenant est de structurer la spécification B. Pour cela, nous proposons de faire appel au mécanisme d'inclusion qui est le mécanisme principal en B pour structurer des grandes spécifications (voir section 1.3.1). Ainsi, notre idée est que les machines B M_{G1} et M_{G2} (relatives aux sous-buts du but parent G) sont liées à la machine B $Ref M_G$ à travers la relation d'inclusion INCLUDES, comme le montre la Figure 3.1. L'opération $Ref OP_G$ qui change l'état de la machine d'inclusion $Ref M_G$ est construite en appelant (grâce à la substitution *appel d'opération* qui existe en B) et combinant les opérations OP_{G1} et OP_{G2} des machines incluses M_{G1} et M_{G2}. La nature de cette combinaison dépend du patron

1. Dans le B classique, l'opération raffinée a le même nom que l'opération abstraite. Pour faciliter la compréhension, nous utilisons ici deux noms distincts.

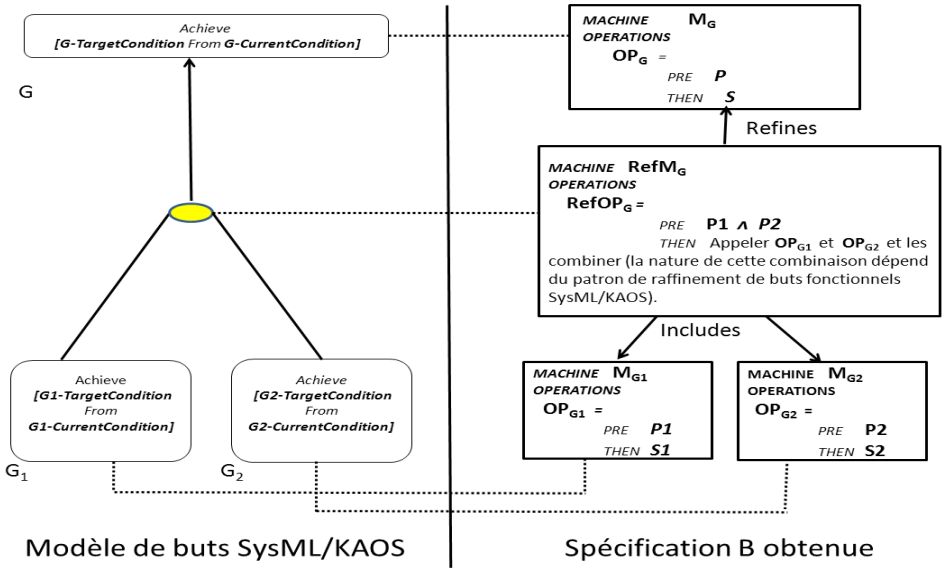


FIGURE 3.1: La traduction des buts fonctionnels et du raffinement SysML/KAOS en B

de raffinement de buts fonctionnels SysML/KAOS, ce que nous présentons dans les sections suivantes. Par ailleurs, la Figure 3.1 montre que la pré-condition de l'opération concrète $RefOP_G$ est construite à partir de la conjonction de toutes les pré-conditions des opérations invoquées.

Remarque concernant le raffinement des pré-conditions B :
Rappelons que la sémantique du raffinement B exige que les pré-conditions soient affaiblies en passant de l'abstraction au raffinement. Or, il s'avère que la traduction proposée de la clause optionnelle **CurrentCondition** de SysML/KAOS vers des pré-conditions B peut ne pas respecter cette contrainte, c'est-à-dire la pré-condition de l'opération concrète $RefOP_G$ peut être plus forte que celle de l'opération abstraite OP_G.

Cela nous a incités à étudier d'autres alternatives pour traduire la **CurrentCondition** SysML/KAOS vers d'autres clauses B. Par exemple, la **CurrentCondition** peut être traduite par un prédicat dans la clause SELECT (une garde) ou encore dans la clause IF (une variante de SELECT). Le problème de ces deux alternatives est que si le prédicat n'est pas vérifié, l'opération B soit : (i) elle « se bloque » pour le cas SELECT ; (ii) elle ne fait rien (skip) pour le cas IF. Dans les deux cas, l'opération s'exécute quand même ce qui n'est pas conforme à la sémantique informelle d'un but SysML/KAOS.

Nous continuons avec l'alternative des pré-conditions B dans la mesure où elle est l'alternative la plus proche de la sémantique informelle d'un but SysML/KAOS. En ce qui concerne l'affaiblissement des pré-conditions, la solution que nous proposons est de réaliser un traitement itératif sur toute l'architecture B afin de remonter à chaque fois la pré-condition de l'opération concrète $RefOP_G$ à l'opération abstraite OP_G, comme le montrent les pré-conditions soulignées dans les Figures 3.2, 3.3 et 3.4.

3.1.3 Traduction du patron de raffinement de buts MILESTONE

Le raffinement MILESTONE en SysML/KAOS prescrit que des sous-buts jalons $G_1, ..., G_n$ sont obligatoires pour atteindre un but parent G, comme le montre la Figure 3.2 (avec juste deux sous-buts).

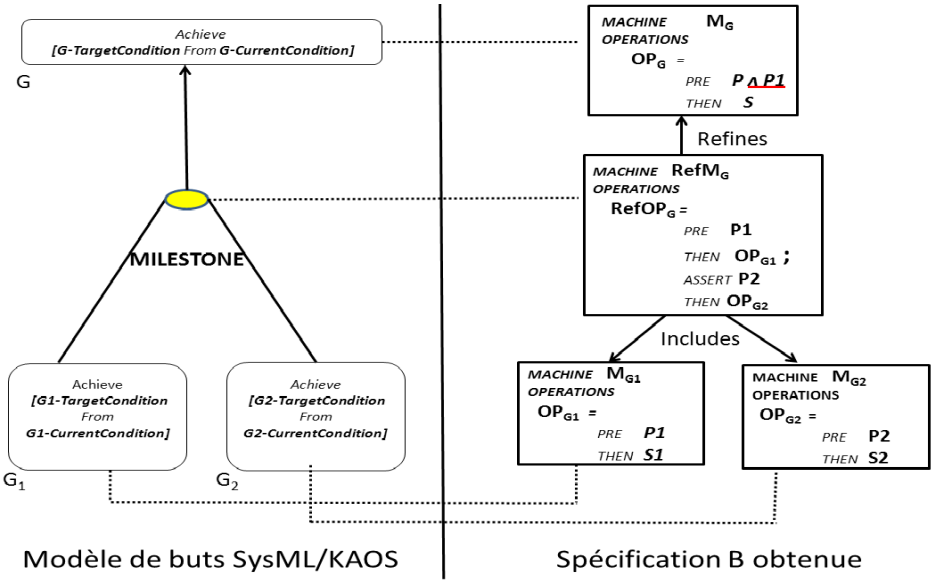


FIGURE 3.2: La correspondance B associée au raffinement MILESTONE

la Figure 3.2 montre que la traduction de ce patron de raffinement en B repose sur le fait de considérer que toutes les opérations invoquées (ici OP_{G1} et OP_{G2}) forment une séquence d'opérations dans l'opération $RefOP_G$, séparées par l'opérateur B de séquentialité sur les substitutions (l'opérateur $(;)$). Outre la possibilité de considérer la conjonction de toutes les pré-conditions des opérations invoquées comme étant la pré-condition de l'opération $RefOP_G$, la construction de cette dernière peut se faire en considérant seulement la pré-condition de la première opération invoquée dans la séquence (ici *P1*). Les pré-conditions des autres opérations invoquées (ici *P2*) sont vérifiées quant à elles par des assertions, c'est-à-dire dans la clause ASSERT qui précède chaque opération invoquée.

3.1.4 Traduction du patron de raffinement de buts AND

Le raffinement AND en SysML/KAOS déclare que le but parent G est satisfait si tous les sous-buts $G_1, ..., G_n$ sont satisfaits, comme le montre la Figure 3.3.

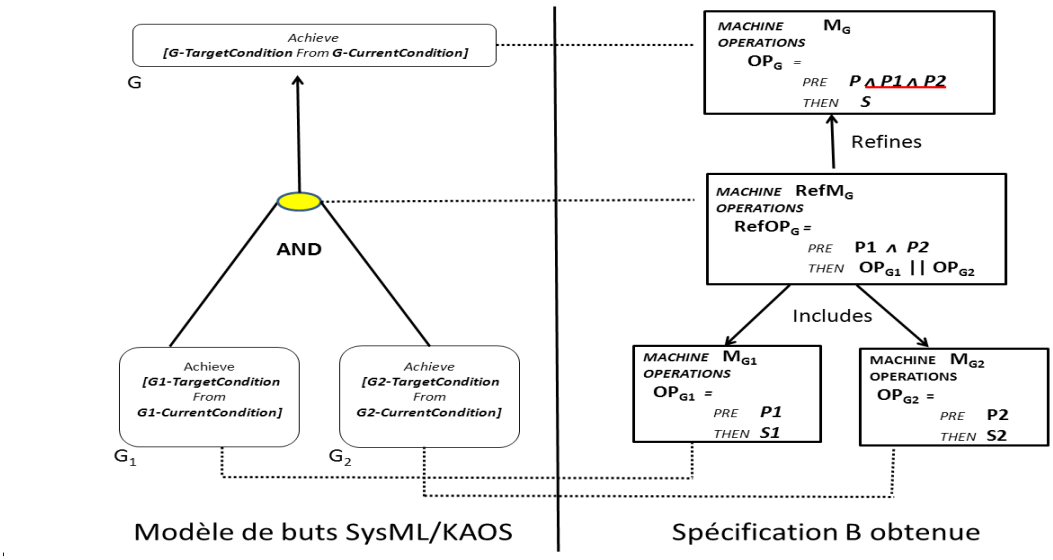


FIGURE 3.3: La correspondance B associée au raffinement AND

Nous proposons que la traduction de ce patron de raffinement en B consiste à considérer que l'opération $RefOP_G$ invoque les opérations (ici OP_{G1} et OP_{G2}) et les exécute en même temps. Pour cela, nous considérons que toutes ces opérations sont composées à travers l'opérateur B de parallélisme sur les substitutions (l'opérateur $(||)$), comme le montre la Figure 3.3. Notons que l'utilisation de cet opérateur exige que les ensembles de variables modifiées par les deux opérations soient disjoints.

3.1.5 Traduction du patron de raffinement de buts OR

Ce raffinement SysML/KAOS capture les alternatives possibles (les sous-buts $G_1, ..., G_n$) pour satisfaire le but parent G, comme le montre la Figure 3.4. Notons que le but parent G est satisfait si exactement un et un seul sous-but est satisfait. Néanmoins, ce patron de raffinement ne précise pas quel sous-but sera effectivement choisi et implanté. Il introduit donc une certaine forme de non-déterminisme borné qui sera résolu plus tard dans la phase d'implémentation.

De là, nous considérons que les opérations invoquées (ici OP_{G1} et OP_{G2}) sont exécutées d'une façon non-déterministe dans $RefOP_G$, séparées par l'opérateur B $(CHOICE)$ comme le montre la Figure 3.4. En fait, cet opérateur B permet de représenter ce non-déterminisme borné. Il permet ainsi de définir un nombre fini de comportements possibles sans préciser lequel sera effectivement implanté.

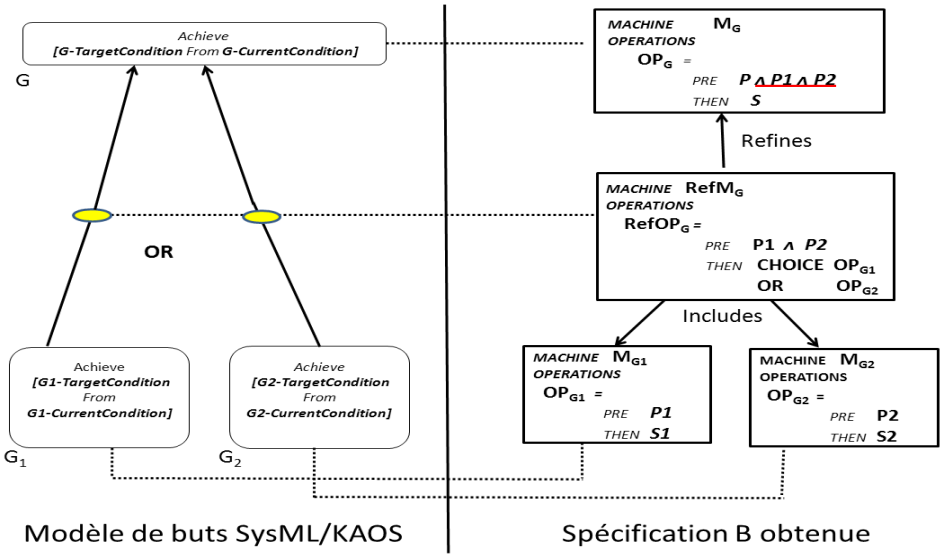


FIGURE 3.4: La correspondance B associée au raffinement OR

3.2 Étude de cas : le composant de localisation

Un système de localisation est une partie critique d'un système de transport terrestre. Plusieurs systèmes de localisation ont été proposés au cours des dernières années. Les systèmes de localisation peuvent être conçus en utilisant des technologies diverses comme GPS, WIFI et les capteurs. Dans le cadre du projet TACOS, l'utilisation de plusieurs technologies est nécessaire dans un système comme un Cycab (le véhicule). En effet, il n'existe pas de technologie efficace qui puisse être utilisé seul. Par exemple, une localisation basée sur les informations GPS ne peut pas être utilisée dans certains contextes, et le composant de localisation doit répondre, même si aucune des données satellitaires ne peut être capturée. Différentes propriétés [Petit *et al.* 2008] doivent être ainsi prises en compte dans le composant de localisation, telles que le format de l'information de localisation, le contexte, etc.

L'objectif de cette section est de donner un retour d'expérience sur la spécification B d'un composant de localisation. La section 3.2.1 donne un aperçu de la première architecture B obtenue « intuitivement » par [Petit *et al.* 2008] sans être guidé par le modèle de buts fonctionnels SysML/KAOS. Ce dernier est ensuite décrit dans la section 3.2.2. Enfin, la section 3.2.3 présente la nouvelle architecture B suite à l'application de notre approche proposée dans la section 3.1.

3.2.1 La première architecture des machines B

Une fois les principales propriétés décrites, [Petit *et al.* 2008] ont commencé à élaborer intuitivement la spécification abstraite B. La Figure 3.5 présente la première architecture B obtenue, qui adopte une approche par composant. Les diffé-

rents composants atomiques (GPS, WIFI, etc.) sont modélisés par des machines abstraites. Ces composants atomiques sont assemblés dans le composant de décision qui fusionne les données. Ce dernier est modélisé en B à travers : (i) la machine abstraite *Location* qui retourne l'information de localisation ; (ii) les différents raffinements (*Location_r1*, *Location_r2*, etc.) où chacun d'entre eux implémente une stratégie différente de fusion de données. L'idée était donc de raffiner la machine *Location* afin d'être de plus en plus précis dans la manière de calculer la localisation. Néanmoins, le principal problème était de déterminer comment prendre en compte toutes les propriétés déjà définies d'une manière graduelle, c'est-à-dire comment trouver par exemple le bon niveau de raffinement pour introduire une propriété donnée. De plus, les obligations de preuve étaient nombreuses et beaucoup d'entre elles étaient complexes à décharger interactivement.

FIGURE 3.5: La première architecture des machines B [Petit *et al.* 2008]

C'est pourquoi nous pensons qu'un modèle semi-formel pourrait être très utile afin d'avoir des lignes directrices sur la façon de structurer les propriétés sous forme de buts. Dans ce qui suit, nous nous concentrons sur l'apport d'un modèle de buts fonctionnels SysML/KAOS pour la construction d'une architecture de spécification B qui assure la correspondance avec toutes les exigences et propriétés fonctionnelles définies.

3.2.2 Le modèle de buts SysML/KAOS du composant de localisation

Lors de l'élaboration d'un modèle de buts, les principales difficultés sont d'abord l'identification des buts, puis la spécification des liens entre ces buts. Souvent, les analystes ont besoin de rechercher à travers les documents préliminaires afin d'extraire les buts en utilisant un certain nombre d'heuristiques [van Lamsweerde 2009], en posant par exemple des questions de type COMMENT et POURQUOI. La Figure 3.6 montre le modèle de buts fonctionnels SysML/KAOS du composant de localisation grâce à l'utilisation de ces heuristiques. Par exemple, une question de type COMMENT sur le but *G* permet de

trouver les buts G_1, G_2 et G_3. Ce modèle de buts contient : (i) des buts de haut niveau, (ii) des liens de raffinement notés par une bulle reliant le but parent avec une flèche, et les sous-buts avec des lignes régulières ; (iii) les exigences et les liens de responsabilité vers des agents logiciels ; (iv) les attentes et les liens de responsabilité vers des agents externes de l'environnement (GPS, WIFI, capteur et accéléromètre). Chaque but est décrit en SysML/KAOS d'une façon informelle en langage naturel **InformalDef**.

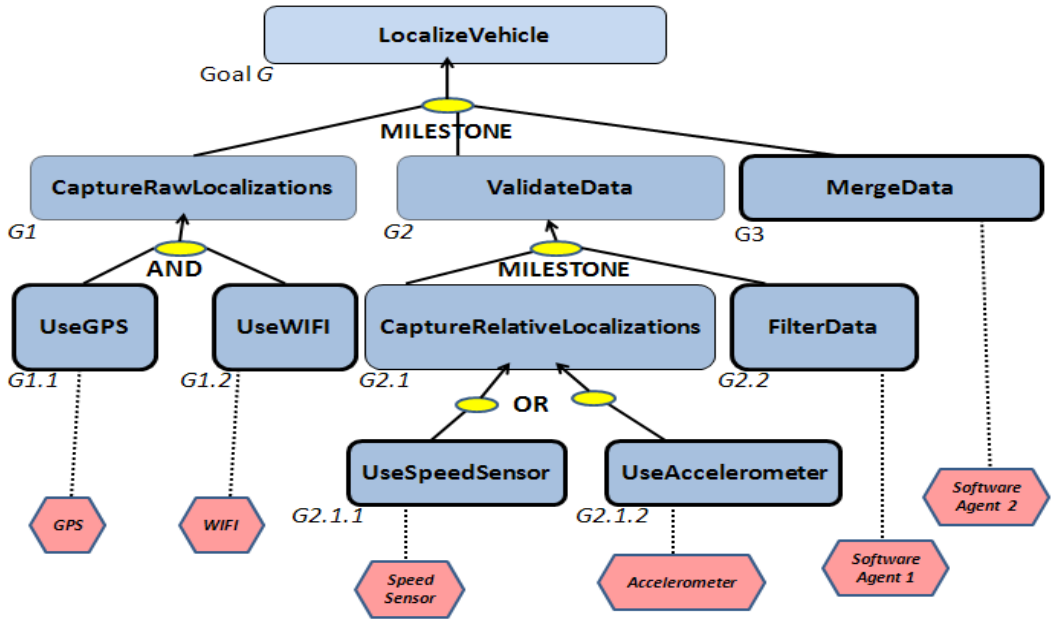


FIGURE 3.6: Le modèle de buts fonctionnels SysML/KAOS du composant de localisation

Le but fonctionnel le plus stratégique G est défini en SysML/KAOS comme suit :

Functional Goal G : Achieve [LocalizeVehicle]
InformalDef : Le véhicule doit être localisé.

Ce but est ensuite raffiné en trois sous-buts selon le raffinement MILESTONE :

Functional Goal G_1 : Achieve [CaptureRawLocalizations]
InformalDef : D'abord, des localisations brutes sont faites.

Functional Goal G_2 : Achieve [ValidateData]
InformalDef : Ensuite, toutes les données de localisation sont validées.

Functional Goal G_3 : Achieve [MergeData]
InformalDef : Enfin, toutes les données sont fusionnées afin d'obtenir une localisation finale.

Le but G_1 est raffiné en deux sous-buts selon le raffinement AND qui précise le type de technologie utilisée pour obtenir des données de localisation :

Functional Goal $G_{1.1}$: Achieve [UseGPS]
InformalDef : Le véhicule doit utiliser le système GPS.

Functional Goal $G_{1.2}$: Achieve [UseWIFI]
InformalDef : Le véhicule doit aussi utiliser le système WIFI.

D'autre part, le but G_2 est raffiné en deux sous-buts selon le raffinement MILESTONE :

Functional Goal $_{2.1}$: Achieve [CaptureRelativeLocalizations]
InformalDef : D'abord, des localisations relatives sont faites.

Functional Goal $_{2.2}$: Achieve [FilterData]
InformalDef : Ensuite, toutes les données de localisation sont filtrées.

Le but $G_{2.1}$ est raffiné en deux sous-buts selon le raffinement OR :

Functional Goal $G_{2.1.1}$: Achieve [UseSpeedSensor]
InformalDef : Le véhicule peut utiliser un système de capteurs.

Functional Goal $G_{2.1.2}$: Achieve [UseAccelerometer]
InformalDef : Ou (exclusif) il peut utiliser un accéléromètre.

3.2.3 Application de l'approche proposée sur l'étude de cas

Les règles de correspondance définies dans la première section de ce chapitre sont appliquées sur le modèle de buts fonctionnels SysML/KAOS du composant de localisation. Pour chaque niveau de la hiérarchie de buts SysML/KAOS, nous décrivons comment obtenir une architecture des machines B qui assure la correspondance avec les buts de ce niveau.

3.2.3.1 Niveau abstrait

Une machine B *Location* est associée au but abstrait *G*. Dans cette machine B, l'opération appelée **locate** traduit ce but ; c'est-à-dire qu'elle va décrire le but *G*, en termes de substitutions généralisées. Si nous considérons que l'obtention d'une localisation est d'avoir une latitude et une longitude, nous obtenons l'opération **locate** présentée dans la Figure 3.7. Notons que le choix de la localisation à ce niveau d'abstraction très élevé est non-déterministe. En d'autres termes, il y a une vision de *haut niveau* de l'opération **locate**. C'est la raison pour laquelle l'opération utilise la substitution ANY. Les ensembles en majuscules sont des ensembles abstraits

utilisés pour typer les variables et sont décrits dans la machine B *Type_sets* comme le montre la Figure 3.8.

```
MACHINE
  Location
SEES
  Type_sets
OPERATIONS

  estimated_location ← locate =
    ANY info_lat , info_long
    WHERE info_lat ∈ LATITUDE ∧ info_long ∈ LONGITUDE
    THEN estimated_location := (info_lat ↦ info_long)
    END
END
```

FIGURE 3.7: La machine B *Location* associée au but abstrait *G*

```
MACHINE Type_sets
SETS
  SUBCOMPONENTS = { gps , wifi } ;
  SUBSENSORS = { speed , accel }
CONCRETE_CONSTANTS
  LATITUDE , LONGITUDE
PROPERTIES
  LATITUDE = NAT
  ∧ LONGITUDE = NAT
END
```

FIGURE 3.8: La machine B *Type_sets* utilisée pour déclarer les ensembles et les constantes

3.2.3.2 Premier raffinement

De même, les sous-buts G_1, G_2 et G_3 sont représentés par trois machines *Raw-Location*, *Validation* et *Fusion* (voir Figures 3.9, 3.10 et 3.11), respectivement. Dans chaque machine B, une opération B traduit ces buts : **locate-raw**, **validate** et **merge**, respectivement. Notons que chaque opération possède des paramètres de sortie. Par ailleurs, les deux dernières opérations ont des paramètres d'entrée avec des pré-conditions typant ces paramètres.

```
MACHINE
  Raw_location
SEES
  Type_sets
OPERATIONS

  subcomponents_locations ← locate_raw =
    ANY sc_locations
    WHERE sc_locations ∈ SUBCOMPONENTS → (LATITUDE × LONGITUDE )
    THEN  subcomponents_locations := sc_locations
    END
END
```

FIGURE 3.9: La machine B *Raw-Location* associée au but asbtrait *G*1

```
MACHINE
  Validation
SEES
  Type_sets
OPERATIONS

  validated_locations ← validate (raw_locations ) =
  PRE

    raw_locations ∈ SUBCOMPONENTS → ( LATITUDE × LONGITUDE )
  THEN
    ANY valid_locations
    WHERE valid_locations ⊆ raw_locations
    THEN validated_locations := valid_locations
    END
  END

END
```

FIGURE 3.10: La machine B *Validation* associée au but *G*2

```
MACHINE
  Fusion
SEES
  Type_sets
OPERATIONS

  merged_location ← merge ( raw_locations ) =
  PRE
    raw_locations ∈ SUBCOMPONENTS ⇸ (LATITUDE × LONGITUDE)
  THEN
    ANY estimate
    WHERE estimate ∈ LATITUDE × LONGITUDE
    THEN merged_location := estimate
    END
  END
END
```

FIGURE 3.11: La machine B *Fusion* associée au but *G*3

Puisque le but G est raffiné en trois sous-buts G_1, G_2 et G_3 selon le patron de raffinement MILESTONE, la machine B abstraite *Location* est raffinée par la machine raffinement *Location_r* (présentée dans la Figure 3.12). Par conséquent, l'opération abstraite **locate** est raffinée par une opération concrète, appelée aussi **locate**. Cette dernière est construite en combinant les opérations **locate-raw**, **validate** et **merge**. Ces dernières sont invoquées en séquence $(;)$ dans l'opération concrète **locate**. Pour cela, la machine raffinement *Location_r* doit inclure les trois machines B *Raw-Location*, *Validation* et *Fusion* qui contiennent les opérations invoquées. Notons que les paramètres de sortie des opérations invoquées **locate-raw** et **validate** sont considérés comme des variables locales (la substitution VAR) dans l'opération concrète **locate**. D'un autre côté, les pré-conditions de typage relatives aux opérations invoquées **validate** et **merge** sont vérifiées dans l'opération concrète **locate** grâce à l'utilisation de la substitution ASSERT. Comme le montre la Figure 3.12, chaque assertion doit précéder l'appel de l'opération invoquée.

```
REFINEMENT
Location_r
REFINES
Location
SEES
Type_sets
INCLUDES
  Raw_location , Validation , Fusion
OPERATIONS

  estimated_location ← locate =
    VAR subcomponents_locations , validated_locations
    IN
      subcomponents_locations ← locate_raw ;

      ASSERT
        subcomponents_locations ∈ SUBCOMPONENTS → ( LATITUDE × LONGITUDE )
      THEN
      validated_locations ← validate (subcomponents_locations ) ;

      ASSERT
        validated_locations ∈ SUBCOMPONENTS ⇸ (LATITUDE × LONGITUDE)
      THEN
      estimated_location ← merge (validated_locations )
        END
      END
    END
END
```

FIGURE 3.12: La machine raffinement *Location_r*

3.2.3.3 Second raffinement

D'une manière analogue aux autres niveaux de la hiérarchie SysML/KAOS, les sous-buts $G_{1.1}$ et $G_{1.2}$ sont représentés par deux machines *GPS* et *WIFI* (voir Figures 3.13 et 3.14), respectivement. Dans chaque machine B, nous avons une opération B qui traduira ces buts : **locate_gps** et **locate_wifi**, respectivement. La machine B abstraite *Raw-Location* est raffinée par la machine raffinement *Raw-Location_r*. Par conséquent, l'opération abstraite **locate-raw** est raffinée par une opération concrète (présentée dans la Figure 3.15). Comme c'est un raffinement AND en SysML/KAOS, l'opération concrète **locate-raw** invoque les opérations (**locate_gps** et **locate_wifi**) et les exécute en même temps en utilisant l'opérateur B de parallélisme sur les substitutions (||).

```
MACHINE
  GPS
SEES
  Type_sets

OPERATIONS
  gps_loc ← locate_gps =
    ANY info_lat , info_long
    WHERE info_lat ∈ LATITUDE ∧ info_long ∈ LONGITUDE
    THEN gps_loc := { (info_lat ↦ info_long ) }
    END

END
```

FIGURE 3.13: La machine B *GPS* associée au but $G_{1.1}$

```
MACHINE
  WIFI
SEES
  Type_sets
OPERATIONS
    wifi_loc ← locate_wifi =
    ANY info_lat , info_long
    WHERE info_lat ∈ LATITUDE ∧ info_long ∈ LONGITUDE
    THEN wifi_loc := {( info_lat ↦ info_long )}
    END
END
```

FIGURE 3.14: La machine B *WIFI* associée au but $G_{1.2}$

```
REFINEMENT
  Raw_location_r
REFINES
  Raw_location
SEES
  Type_sets
INCLUDES
  GPS , WIFI
OPERATIONS

  subcomponents_locations ← locate_raw =
    VAR gps_loc , wifi_loc
    IN
      gps_loc ← locate_gps || wifi_loc ← locate_wifi ;
      subcomponents_locations := ( { gps } × gps_loc ) ∪ ( { wifi } × wifi_loc )
    END
END
```

FIGURE 3.15: La machine raffinement *Raw-Location_r*

D'autre part, le raffinement MILESTONE du but G_2 en deux sous-buts $G_{2.1}$ et $G_{2.2}$ est transformé en B selon le même principe que pour le premier raffinement (voir les Figures 3.16, 3.17 et 3.18).

```
MACHINE
  Rel_location
SEES
  Type_sets
OPERATIONS
  sensors_locations ← locate_rel =
    ANY sens_locations
    WHERE sens_locations ∈ SUBSENSORS ⇸ (LATITUDE × LONGITUDE)
    THEN sensors_locations := sens_locations
    END
END
```

FIGURE 3.16: La machine B *Rel_location* associée au but $G_{2.1}$

```
MACHINE
  Filter
SEES
  Type_sets
OPERATIONS
  kept_locations ← filter_locations (new_raw_locations , sensors_locations ) =
  PRE
     new_raw_locations ∈ SUBCOMPONENTS → (LATITUDE × LONGITUDE)
     ∧ sensors_locations ∈ SUBSENSORS → ( LATITUDE × LONGITUDE)
  THEN

    ANY
      k_locations
    WHERE
      k_locations ⊆ new_raw_locations
    THEN
      kept_locations := k_locations
    END
  END
END
```

FIGURE 3.17: La machine B *Filter* associée au but $G_{2.2}$

```
REFINEMENT
Validation_r
REFINES
  Validation
SEES
  Type_sets
INCLUDES
  Rel_location , Filter
OPERATIONS
    validated_locations ← validate (raw_locations ) =
    VAR sensors_locations
    IN
      sensors_locations ← locate_rel ;
      ASSERT
        raw_locations ∈ SUBCOMPONENTS → (LATITUDE × LONGITUDE )
         ∧ sensors_locations ∈ SUBSENSORS → ( LATITUDE × LONGITUDE )
      THEN
        validated_locations ← filter_locations (raw_locations, sensors_locations)
        END
    END
END
```

FIGURE 3.18: La machine raffinement *Validation_r*

3.2.3.4 Troisième raffinement

De même, les sous-buts $G_{2.1.1}$ et $G_{2.1.2}$ sont représentés par deux machines B *Speed* et *Accel* (voir Figures 3.19 et 3.20), respectivement. Dans chaque machine B, une opération B traduit ces buts : **locate_speed** et **locate_accel** (respectivement).

```
MACHINE
    Speed
SEES
    Type_sets
OPERATIONS
    speed_location ← locate_speed =
    ANY info_lat , info_long
    WHERE info_lat ∈ LATITUDE ∧ info_long ∈ LONGITUDE
    THEN speed_location := { (info_lat ↦ info_long ) }
    END
END
```

FIGURE 3.19: La machine B *Speed* associée au but $G_{2.1.1}$

```
MACHINE
    Accel
SEES
    Type_sets
OPERATIONS
    accel_location ← locate_accel =
    ANY info_lat , info_long
    WHERE info_lat ∈ LATITUDE ∧ info_long ∈ LONGITUDE
    THEN accel_location := { (info_lat ↦ info_long ) }
    END
END
```

FIGURE 3.20: La machine B *Accel* associée au but asbtrait $G_{2.1.2}$

La machine B abstraite *Rel_location* est raffinée par la machine raffinement *Rel_location_r*. Par conséquent, l'opération abstraite **locate_rel** est raffinée par une opération concrète (présenté dans la Figure 3.21). Cette dernière invoque les opérations (**locate_accel** and **locate_speed**) et les exécute d'une façon non-déterministe à travers l'opérateur B sur les substitutions (*CHOICE*).

```
REFINEMENT
Rel_location_r
REFINES
Rel_location
SEES
    Type_sets
INCLUDES
    Speed, Accel
OPERATIONS
    sensors_locations ← locate_rel =
    CHOICE
        VAR speed_loc
        IN
            speed_loc ← locate_speed ;
            sensors_locations := ( {speed} × speed_loc )
        END
    OR
        VAR accel_loc
        IN
            accel_loc ← locate_accel ;
            sensors_locations := ( {accel} × accel_loc )
        END
    END
END
```

FIGURE 3.21: La machine raffinement *Rel_location_r*

3.2.4 Bilan

A la différence de la première architecture des machines B (voir la Figure 3.5), la nouvelle architecture (voir la Figure 3.22) permet d'établir des liens facilitant la correspondance entre les buts SysML/KAOS (et par conséquent les besoins des clients) et les opérations B décrivant ces buts.

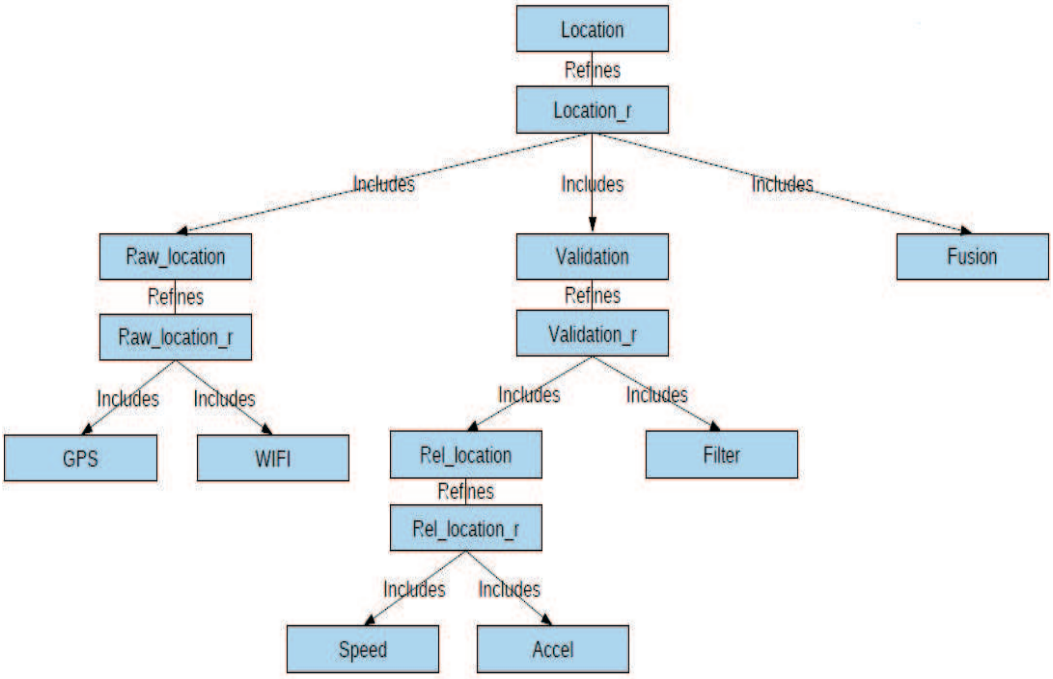


FIGURE 3.22: L'architecture des machines B en appliquant l'approche proposée

En termes d'obligations de preuve, le tableau 3.1 résume le nombre d'obligations de preuve (OP) générées par l'Atelier B. Six obligations de preuve sont générées, dont 4 prouvées automatiquement par le prouveur de l'Atelier B. Cela montre que l'approche proposée permet également de faciliter l'activité de preuve.

Une fois l'architecture d'un modèle B obtenue, la conception du futur logiciel peut commencer. En d'autres termes, le processus de raffinement B vers l'implémentation peut commencer. L'intérêt de l'approche proposée est qu'elle aide l'ingénieur B à la détermination des machines B concernées par cette étape de raffinement. En fait, le processus de raffinement ne concerne que les machines B correspondant aux exigences logicielles (sous la responsabilité des agents logiciels) puisqu'une attente (sous la responsabilité d'un agent externe) ne sera pas implémentée par le futur logiciel. Les machines correspondant aux attentes ne font pas partie du système à développer, elles représentent la spécification de l'environnement du système. Par conséquent, la composante B qui correspond à un tel concept (une attente) est un composant IMPLEMENTATION. Plus précisément, dans notre étude de cas, les machines *GPS*, *WIFI*, *Speed* et *Accel* ne sont pas raffinées puisqu'elles correspondent à des attentes SysML/KAOS. Elles sont donc implémentées

Machine	Nombre d'OP	Automatique	Interactive	%Pr
Location	0	0	0	100
Type_sets	0	0	0	100
Raw_location	0	0	0	100
Validation	0	0	0	100
Fusion	0	0	0	100
Location_r	2	2	0	100
GPS	0	0	0	100
WIFI	0	0	0	100
Raw_location_r	2	2	0	100
Rel_location	0	0	0	100
Filter	0	0	0	100
Validation_r	0	0	0	100
Speed	0	0	0	100
Accel	0	0	0	100
Rel_location_r	2	0	2	100
Total	6	4	2	100

TABLE 3.1: Résultat du nombre d'obligations de preuve obtenues par l'Atelier B

par des composants matériels (un GPS, un WIFI, etc.) dans un véhicule. Seules les machines *Filter* et *Fusion* sont raffinées.

3.3 Discussion de l'approche

3.3.1 Avantages

Nous avons présenté dans ce chapitre une première tentative permettant de faciliter la correspondance entre les buts fonctionnels SysML/KAOS et la spécification B abstraite. L'approche a été validée par une étude de cas qui a montré que SysML/KAOS peut fournir une aide pour la construction et la structuration des spécifications formelles B pertinentes grâce surtout à la dimension graphique de SysML/KAOS (boîtes, flèches, etc.) qui permet d'assister les concepteurs à mieux comprendre les exigences.

D'un autre côté, l'extension de SysML/KAOS avec une étape de formalisation en B permet de fournir aussi une meilleure précision et par conséquent une forme d'analyse plus riche. En effet, cette étape de formalisation permet de prouver des propriétés de cohérence sur le modèle de buts fonctionnels SysML/KAOS en prouvant les obligations de preuve produites par la contrepartie formelle en B. Si nous pouvons prouver par exemple les obligations de preuve du raffinement d'une opération $Ref OP_G$ par un choix entre OP_{G1} et OP_{G2} cela signifie que la décomposition OR de but G est correcte. Autrement, cela signifie qu'une ou plusieurs expressions des opérations ne sont pas correctes ou que certains sous-buts manquent ou que le patron de raffinement de buts est mal appliqué. L'intérêt de cette approche est qu'elle se base sur la technique de preuve de théorème, évitant ainsi le problème de l'explosion du nombre d'états explorés présent avec les techniques de model-checking sur lesquelles se basent les approches existantes (voir la section 2.2) pour

la vérification des modèles des exigences.

Cependant, il n'est pas possible de vérifier qu'un modèle de buts fonctionnels SysML/KAOS et sa contrepartie formelle sont équivalents. En fait, nous ne pouvons jamais assurer que l'expression des opérations en B correspond exactement à l'expression du but lié puisque ce dernier est informel. Cela signifie que les preuves en B ne sont pas suffisantes pour valider la conformité de la spécification vis-à-vis des besoins originaux des clients. Pour cela, l'idée est d'utiliser la technique de l'animation pour valider la spécification formelle dérivée et par conséquent sa contrepartie semi-formelle (le modèle de buts) vis-à-vis des besoins initiaux des clients. Cette étape animation doit se faire à chaque niveau de raffinement de buts SysML/KAOS. L'intérêt de cette étape d'animation est qu'elle indique non seulement qu'il peut y avoir des déviations par rapport aux besoins initiaux, mais aide aussi à la détection de quelques erreurs de spécification. Pour cela, l'utilisation des animateurs associés à B tels que ProB [Leuschel & Butler 2003] peut être utile afin de permettre aux utilisateurs de valider leurs spécifications.

3.3.2 Limites

Malgré ces apports, l'approche proposée dans ce chapitre présente un certain nombre de limites liées essentiellement à la sémantique du raffinement B.

D'un côté, le raffinement B est connu pour sa particularité d'être à « signature identique », c'est-à-dire qu'une opération abstraite est raffinée en B classique par une opération concrète en conservant obligatoirement la même signature (mêmes noms d'opérations et mêmes paramètres d'entrée et de sortie pour chaque opération). Cette signature identique peut poser divers problèmes quant à l'enrichissement ultérieur des opérations concrètes par des buts non-fonctionnels SysML/-KAOS. Par exemple, il se peut qu'un but non-fonctionnel se traduise en B par un nouveau paramètre dans une opération concrète. Cet enrichissement n'est malheureusement pas permis en B sans un enrichissement préalable de l'opération abstraite. De plus, le raffinement à signature identique n'offre pas la possibilité de rajouter de nouvelles opérations durant le raffinement. Or, il se peut qu'un but non-fonctionnel se traduise par une nouvelle opération en B.

D'un autre côté, nous avons présenté dans la section 3.1.2 le problème que peut poser la traduction des CurrentCondition SysML/KAOS vers des pré-conditions B lors du raffinement. Bien que la solution proposée (remonter les pré-conditions) soit correcte, elle ne correspond pas complètement à l'esprit des méthodes GORE qui s'appuient sur l'introduction graduelle des informations.

Ces différentes limites confirment l'idée disant que le B classique n'est pas très approprié pour le développement de systèmes (spécification par observation), contrairement à Event-B (voir section 1.3.2). C'est pourquoi, nous nous intéressons dans le chapitre suivant à la méthode Event-B.

CHAPITRE 4

Une approche dirigée par les buts fonctionnels SysML/KAOS pour la construction de spécifications abstraites Event-B

Sommaire

Dans ce chapitre, nous proposons une autre approche constructive dans laquelle les modèles Event-B sont construits d'une manière incrémentale à partir des modèles de buts fonctionnels SysML/KAOS. En plus de la notion de raffinement, Event-B et SysML/KAOS partagent la notion d'observation qui s'appuie sur l'introduction graduelle des informations. Les résultats présentés dans ce chapitre ont fait l'objet de publications [Matoussi *et al.* 2010b, Matoussi *et al.* 2011].

4.1 Un aperçu général de l'approche

L'approche proposée vise à dériver une représentation Event-B directement à partir du modèle de buts SysML/KAOS. Pour atteindre cet objectif, nous proposons dans ce chapitre de prendre en compte tout d'abord les buts fonctionnels SysML/-KAOS en Event-B. Rappelons qu'un but fonctionnel SysML/KAOS peut être un but « Achieve » ou sa variante duale « Cease ». Un but « Achieve » prescrit des comportements attendus où une certaine condition cible doit être tôt ou tard établie quand une autre condition est vérifiée dans l'état actuel du système. Un but « Achieve » dans SysML/KAOS est dénoté comme suit : *Achieve[**TargetCondition** From **CurrentCondition**]*. La description temporelle et informelle de cette notation est la suivante, sachant que le préfixe ***CurrentCondition*** est optionnel (autrement dit, il peut être vrai) :

*[Si **CurrentCondition** alors] tôt ou tard **TargetCondition**.*

En faisant référence aux concepts de garde et de post-condition qui existent dans Event-B, on peut considérer une **TargetCondition** d'un but fonctionnel SysML/KAOS comme une post-condition du système. La transformation repose ainsi sur le fait de représenter chaque but fonctionnel SysML/KAOS comme un événement Event-B où : (i) la **CurrentCondition** du but est considérée comme la garde **Guard** de l'événement Event-B ; (ii) la **TargetCondition** de ce but est considérée comme l'action **Post** de cet événement.

Considérons que le but fonctionnel « Achieve » G est raffiné en deux sous-buts G_1 et G_2 comme le montre la Figure 4.1. La Figure 4.2 montre la représentation Event-B de ce modèle de buts SysML/KAOS. Chaque niveau i ($i \in [0..n]$) dans la hiérarchie de buts SysML/KAOS est représenté comme une machine Event-B M_i. De plus, chaque machine Event-B M_{i+1} est une machine de raffinement qui raffine la machine Event-B M_i lié au niveau i.

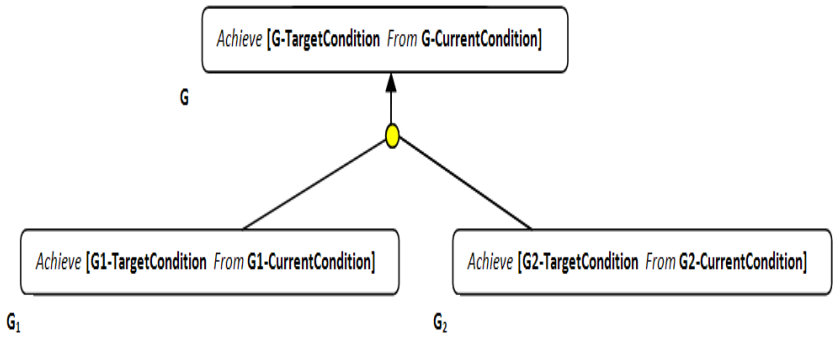


FIGURE 4.1: Exemple d'un modèle de buts fonctionnels SysML/KAOS

MACHINE Abstract M_0 **EVENTS** **Event** **EvG** $\widehat{=}$ **when** *G-Guard* **then** *G-Post* **end**	**MACHINE** First M_1 **REFINES** Abstract M_0 **EVENTS** **Event** **EvG1** $\widehat{=}$ **when** G_1-*Guard* **then** G_1-*Post* **end** **Event** **EvG2** $\widehat{=}$ **when** G_2-*Guard* **then** G_2-*Post* **end**
(a) Modèle Abstrait M_0	(b) Modèle Raffiné M_1

FIGURE 4.2: La représentation Event-B du modèle de buts fonctionnels SysML/KAOS

Jusqu'à présent, les autres éléments d'Event-B (l'initialisation, les variables et leurs invariants de typage, et les contextes) sont manuellement complétés par le concepteur. Nous envisageons comme perspective à ce travail de les dériver à partir du modèle objets KAOS qui contient tous les concepts utilisés dans la définition des buts du modèle de buts. Comme le modèle objet KAOS est un diagramme de classes UML, l'idée est de réutiliser par exemple les travaux UML-B définis par [Mammar & Laleau 2006, Snook & Butler 2006].

On peut se demander comment la contrainte temporelle (le mot-clé « tôt ou tard ») peut être exprimée dans la transformation Event-B, puisqu'il n'y a pas de notion de temps dans ce langage. J.R Abrial [Abrial 1996b] explique que la dimension temporelle n'a pas besoin d'être explicitement considérée et recommande d'utiliser les événements et leurs raffinements pour exprimer cette dimension. En effet, un seul événement (on dit qu'il est « atomique ») peut avoir lieu dans une

unité de temps et c'est le niveau de raffinement qui détermine implicitement la taille de cette unité. Au niveau le plus abstrait par exemple, cette unité peut être aussi grande que souhaitée. Le temps est étiré en se déplaçant d'une abstraction à son raffinement d'une manière analogue à l'étirement de l'espace quand on passe d'une vision normale à une vision microscopique. Cet étirement du temps révèle certains « détails de temps », c'est-à-dire de nouveaux événements. Cette idée est partagée aussi par les auteurs de [Burns & Hayes 2010] quand ils décrivent la notion de « Time Bands ». Nous pensons que cette interprétation du temps correspond bien à la définition d'un but « Achieve » en SysML/KAOS. Cependant, si des délais sont associés à un tel but (un « Achieve » temporisé), il sera nécessaire d'introduire de nouvelles variables afin de modéliser le temps.

Dans les sections suivantes, nous proposons une expression Event-B pour chaque patron de raffinement de buts fonctionnels SysML/KAOS. Basé sur l'ensemble classique des règles d'inférence d'Event-B [Abrial 2010], les obligations de preuve systématiques pour chaque patron de raffinement de buts sont identifiées. Le plan suivant est utilisé pour décrire chaque patron de raffinement :

1. **Description du patron SysML/KAOS :** Une courte définition informelle du patron de raffinement de buts fonctionnels SysML/KAOS est présentée.
2. **Sémantique formelle du patron en Event-B :** Nous proposons de donner une expression Event-B au patron de raffinement de buts fonctionnels SysML/KAOS comme suit :
   (a) **Définition formelle :** Une expression Event-B est donnée au patron de raffinement de buts fonctionnels SysML/KAOS. Pour cela, l'idée est de se baser sur les travaux qui caractérisent le raffinement en terme de comparaisons des traces [Lynch & Vaandrager 1995, Abrial 2010].
   (b) **Identification des obligations de preuve :** Des arguments intuitifs et formels définissant exactement ce qui doit être prouvé pour le patron de raffinement de buts SysML/KAOS sont identifiés.
3. **Synthèse :** L'expression Event-B du patron de raffinement de buts fonctionnels SysML/KAOS est synthétisée.

4.2 L'expression du patron MILESTONE en Event-B

4.2.1 Description du patron SysML/KAOS

Rappelons que le raffinement MILESTONE raffine un but « Achieve » en introduisant des états du jalons intermédiaires $G_1 \ldots, G_n$ pour atteindre un état satisfaisant la **G-TargetCondition** à partir d'un état satisfaisant la **G-CurrentCondition** comme le montre la Figure 4.3 (avec juste deux sous-buts).

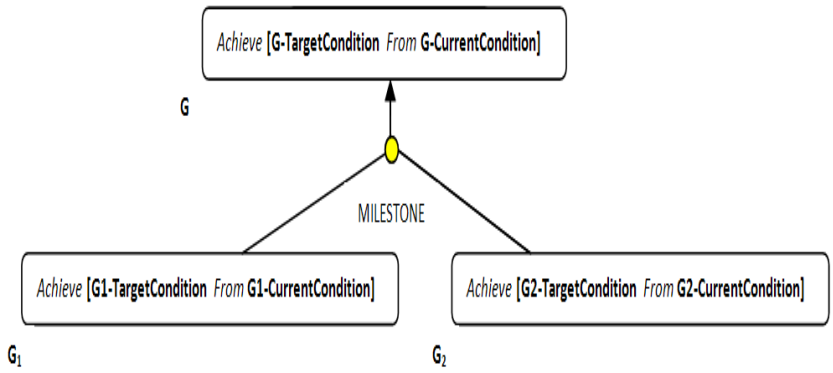


FIGURE 4.3: Le patron de raffinement de buts fonctionnels MILESTONE

Le premier sous-but G_1 est un but « Achieve » avec comme condition du jalon la **G1-TargetCondition**. Ce but exprime que la condition du jalon doit être tôt ou tard établie si **G1-CurrentCondition** est vérifiée dans l'état actuel. Le deuxième sous-but est également un but « Achieve », il exprime que tôt ou tard **G2-TargetCondition** doit être établie si la condition du jalon spécifique **G2-CurrentCondition** (dérivé à partir **G1-TargetCondition**) est vérifiée dans l'état actuel.

4.2.2 Sémantique formelle du patron en Event-B

Puisque la satisfaction de tous les sous-buts (selon un ordre bien déterminé) implique la satisfaction du but parent, l'événement abstrait **EvG** est raffiné par *la séquence* de tous les nouveaux événements (**EvG1**, **EvG2**).

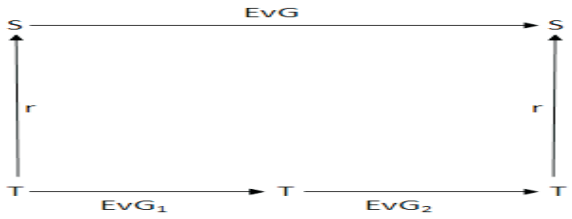
4.2.2.1 Définition formelle

Dans Event-B, les informations sur l'ordonnancement des événements peuvent être exprimées dans les gardes et les actions des événements en utilisant des variables de contrôle. Afin d'éviter ces extra-variables qui alourdissent la spécification Event-B, cette thèse propose une autre solution. Elle consiste à définir une extension syntaxique des règles de preuve de raffinement Event-B afin de supporter qu'un événement abstrait soit raffiné par la séquence de nouveaux événements. Ainsi, l'événement abstrait **EvG** est raffiné comme suit :

(**EvG1** ; **EvG2**) Refines **EvG**

Il est nécessaire de définir les différentes obligations de preuve associées à cette sémantique du raffinement. Pour cela, notre idée est d'essayer tout d'abord de caractériser le raffinement en terme de comparaisons des traces [Lynch & Vaandrager 1995, Abrial 2010] comme le montre le schéma ci-dessous. Notons que *r* est une relation binaire de l'ensemble des variables concrètes

T vers l'ensemble des variables abstraites S. r exprime l'invariant de collage entre l'état du raffinement et l'état de l'abstraction.



Les obligations de preuve d'Event-B doivent établir la condition de « forward simulation » [Lynch & Vaandrager 1995], qui est une condition suffisante pour garantir le raffinement. En effet, cette condition permet de garantir que toute trace du modèle raffiné (**EvG1** ; **EvG2**) est aussi une trace de l'abstraction **EvG**. La condition de « forward simulation » est notée comme suit :

$$r^{-1}; EvG1; EvG2 \subseteq EvG; r^{-1}$$

Afin de pouvoir explorer et traduire cette condition, il suffit de définir d'une manière ensembliste les éléments S, T, EvG, $EvG1$, $EvG2$ et r, comme le montre la Figure 4.4. Notons que $I(v)$ et $J(v,w)$ désignent l'invariant et l'invariant de collage, respectivement.

$S \triangleq \{v | I(v)\}$

$T \triangleq \{w | \exists v \,.\, (I(v) \wedge J(v,w))\}$

$EvG \triangleq \{v \mapsto v' | I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}Post(v,v')\}$

$EvG1 \triangleq \{w \mapsto t | \exists v \,.\, (I(v) \wedge J(v,w)) \wedge G_1\text{-}Guard(w) \wedge G_1\text{-}Post(w,t)\}$

$EvG2 \triangleq \{t \mapsto w' | \exists v' \,.\, (I(v') \wedge J(v',w')) \wedge G_2\text{-}Guard(t) \wedge G_2\text{-}Post(t,w')\}$

$r \triangleq \{w \mapsto v | I(v) \wedge J(v,w))\}$

$r^{-1} \triangleq \{v \mapsto w | I(v) \wedge J(v,w))\}$

FIGURE 4.4: La représentation ensembliste des éléments Event-B

Pour bien exprimer cette sémantique de raffinement, la représentation ensembliste (Figure 4.4) est étendue en exprimant la séquence $EvG1; EvG2$ d'une manière ensembliste, comme suit :

$EvG1; EvG2 \triangleq \{w \mapsto w' | \exists v \,.\, I(v) \wedge J(v,w) \wedge \exists t \,.\, G_1\text{-}Guard(w) \wedge G_1\text{-}Post(w,t) \wedge G_2\text{-}Guard(t) \wedge G_2\text{-}Post(t,w')\}$

Cette définition exprime qu'on commence dans un état w où seulement l'événement **EvG1** pourra être exécuté, c'est-à-dire que sa garde est vraie. Une fois cet événement exécuté (sa post-condition devient vraie), on arrive à un état intermédiaire t dans lequel l'événement **EvG2** peut être exécuté. L'exécution de ce dernier événement permet d'atteindre l'état w'. En se basant sur cette définition, les deux parties de la condition du « forward simulation » peuvent être écrites comme suit :

$$r^{-1};EvG1;EvG2 \triangleq \{v \mapsto w' \mid \exists w \,.\, I(v) \wedge J(v,w) \wedge \exists t.G_1\text{-}Guard(w) \wedge G_1\text{-}Post(w,t) \wedge G_2\text{-}Guard(t) \wedge G_2\text{-}Post(t,w')\}$$

$$EvG;r^{-1} \triangleq \{v \mapsto w' \mid \exists v'.I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w')\}$$

4.2.2.2 Identification des obligations de preuve

En se basant sur la définition ensembliste de la condition de « forward simulation », les règles systématiques définissant exactement ce qui doit être prouvé pour ce patron sont identifiées. En fait, la traduction de cette condition revient à prouver le séquent [1] suivant, par l'application de la règle d'inférence *AND-L* [2] :

$$\begin{array}{l} I(v) \\ J(v,w) \\ G_1\text{-}Guard(w) \\ G_1\text{-}Post(w,t) \\ G_2\text{-}Guard(t) \\ G_2\text{-}Post(t,w') \\ \vdash \\ I(v) \\ G\text{-}Guard(v) \\ \exists v'.G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w') \end{array} \qquad \text{(I)}$$

Comme mentionné auparavant, la définition informelle du patron de raffinement MILESTONE en SysML/KAOS repose sur le fait que **G1-TargetCondition** implique la **G2-CurrentCondition**, ce qui se traduit en Event-B comme suit :

$$G_1\text{-}Post \Rightarrow G_2\text{-}Guard \qquad \textbf{(PO1)}$$

Cette première obligation de preuve permet de simplifier le séquent (I) comme suit :

1. Le symbole $\vdash$ est appelé le tourniquet. Un séquent est une déduction qui exprime que d'un ensemble de prémisses (la partie gauche du tourniquet), on peut déduire un ensemble de conclusions (la partie droite du tourniquet). Page 33 de [Abrial 2010].

2. Elle permet de simplifier les prédicats conjonctifs apparaissant dans les hypothèses d'un séquent. Page 67 de [Abrial 2010]

$$
\begin{array}{|l}
I(v) \\
J(v,w) \\
G_1\text{-}Guard(w) \\
G_1\text{-}Post(w,t) \\
G_2\text{-}Post(t,w') \\
\vdash \\
I(v) \\
G\text{-}Guard(v) \\
\exists v'.G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w') \\
\end{array}
\qquad \text{(I')}
$$

Ce séquent (I') peut être divisé en deux séquents ((I'.1) et (I'.2)) comme suit, par l'application de la règle d'inférence *AND-R*[3] :

$$
\begin{array}{|l}
I(v) \\
J(v,w) \\
G_1\text{-}Guard(w) \\
G_1\text{-}Post(w,t) \\
G_2\text{-}Post(t,w') \\
\vdash \\
I(v) \\
G\text{-}Guard(v) \\
\end{array}
\qquad \text{(I'.1)}
$$

La définition SysML/KAOS d'un raffinement MILESTONE peut nous guider à découvrir intuitivement l'obligation de preuve suffisante pour prouver ce dernier séquent. Cette obligation de preuve est la suivante :

$G_1\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO2)**

$$
\begin{array}{|l}
I(v) \\
J(v,w) \\
G_1\text{-}Guard(w) \\
G_1\text{-}Post(w,t)) \\
G_2\text{-}Post(t,w') \\
\vdash \\
I(v) \\
\exists v'.(G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w')) \\
\end{array}
\qquad \text{(I'.2)}
$$

Comme pour le séquent (I'.1), l'obligation de preuve suivante est suffisante pour prouver le séquent (I'.2) :

$G_2\text{-}Post \Rightarrow G\text{-}Post$ **(PO3)**

3. Elle permet de simplifier les prédicats conjonctifs apparaissant dans les conclusions d'un séquent. Page 67 de [Abrial 2010]

4.2.3 Synthèse

Cette sous-section résume les règles systématiques définissant exactement ce qui doit être prouvé pour ce patron pour assurer que la séquence des événements (**EvG1 ; EvG2**) raffine l'événement abstrait **EvG**. En fait, en plus de l'obligation de preuve de *faisabilité* liée à chaque événement (voir section 1.3.6.2), ce type de raffinement nécessite de prouver trois différents lemmes :

– *La contrainte d'ordonnancement* exprime la caractéristique « par jalon » entre les événements Event-B.

$$G_1\text{-}Post \Rightarrow G_2\text{-}Guard \qquad \textbf{(PO1)}$$

– *Le renforcement de la garde* assure que la garde concrète de la séquence (la garde du premier événement dans la séquence) implique la garde abstraite.

$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \qquad \textbf{(PO2)}$$

– *La simulation d'action* assure que l'achèvement de la séquence des événements concrets (plus précisément l'action du dernier événement dans la séquence) transforme les variables concrètes d'une manière qui ne contredit pas l'événement abstrait.

$$G_2\text{-}Post \Rightarrow G\text{-}Post \qquad \textbf{(PO3)}$$

4.3 L'expression du patron AND en Event-B

4.3.1 Description du patron SysML/KAOS

Rappelons que ce raffinement raffine un but « Achieve » en deux (ou plusieurs) sous-buts quand la conjonction de ces sous-buts est suffisante pour établir la satisfaction du but parent *G* comme le montre la Figure 4.5 (avec juste deux sous-buts).

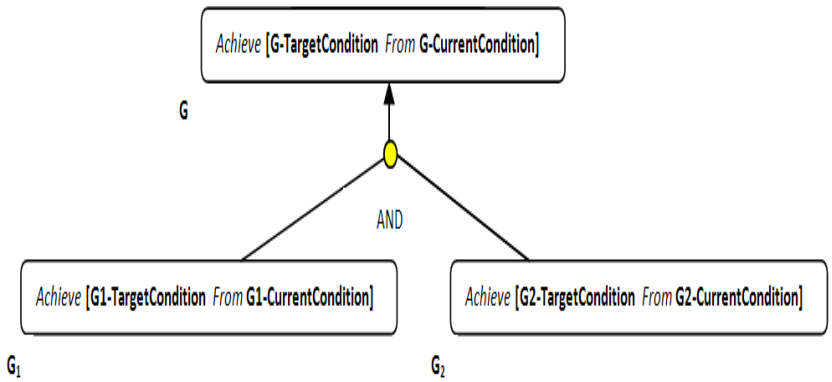


FIGURE 4.5: Le patron de raffinement de buts « AND »

Restriction : En SysML/KAOS, rien n'est dit sur l'ordre d'exécution des différents sous-buts pour un raffinement AND de buts fonctionnels. La transformation en Event-B révèle que cela peut constituer un sérieux problème quand ces sous-buts manipulent des variables partagées. Pour mieux expliquer ce problème, supposons

par exemple qu'une variable partagée x soit modifiée par deux sous-buts (G_1 et G_2). G_1 augmente x de 6 et G_2 multiplie x par 4. Ainsi, il est facile de constater que l'ordre d'exécution des deux sous-buts est important. Pour éviter ce problème, une condition suffisante est de forcer un raffinement AND à manipuler seulement *des ensembles de variables disjoints*. Cette solution est similaire à la solution proposée par Abrial [Abrial 1996a] pour les variables figurant dans des substitutions parallèles, et que nous avons adoptée dans l'approche dédiée au B classique (voir section 3.1.4). S'il y a un raffinement AND avec *des variables partagées*, l'idée est de transformer cette forme de raffinement AND en un raffinement MILESTONE et ensuite explicitement spécifier l'ordre des modifications sur ces variables partagées.

4.3.2 Sémantique formelle du patron

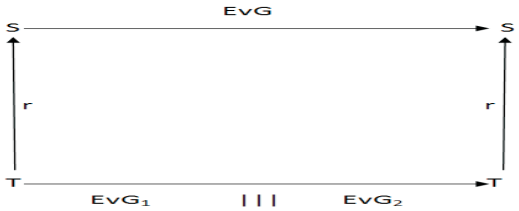
De manière analogue au raffinement MILESTONE, la satisfaction de tous les sous-buts implique la satisfaction du but parent. Cependant, l'exécution des nouveaux événements Event-B (correspondant aux sous-buts) ne doit pas nécessairement suivre un ordre spécifique. L'idée est de s'inspirer de la sémantique de l'opérateur *d'entrelacement* dans les algèbres de processus [Sangiorgi 1996]. Les nouveaux événements (**EvG1**, **EvG2**) sont ainsi exécutés dans un ordre arbitraire : **EVG1 ;EVG2** ou **EVG2 ;EVG1**.

4.3.2.1 Définition formelle

Une extension syntaxique des règles de preuve de raffinement Event-B est proposée afin de supporter qu'un événement abstrait **EvG** soit raffiné par l'entrelacement de nouveaux événements comme suit :

(**EvG1** ||| **EvG2**) Refines **EvG**

D'une manière similaire au raffinement MILESTONE, l'idée est de caractériser ce raffinement en terme de comparaisons des traces, comme illustré dans le schéma ci-dessous.



La condition de « forward simulation », assurant que toute trace d'un modèle raffiné soit aussi une trace de l'abstraction, est définie comme suit :

$$r^{-1}; (EvG1|||EvG2) \subseteq EvG; r^{-1}$$

Pour bien exprimer cette nouvelle sémantique de raffinement, l'idée est d'étendre la représentation ensembliste (Figure 4.4) en exprimant la relation d'entrelacement $EvG1|||EvG2$ d'une manière ensembliste, comme suit :

$$EvG1|||EvG2 \triangleq \{w \mapsto w' | \exists v \ . \ I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge ((G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \Rightarrow G_1\text{-}Post(w,w') \wedge G_2\text{-}Post(w,w'))\}$$

Cette définition exprime qu'on commence dans un état w où les deux événements **EvG1** et **EvG2** pourraient être exécutés, c'est-à-dire que leurs gardes sont vraies. On peut ensuite atteindre un autre état w' si et seulement si les deux événements sont exécutés. En se basant sur cette définition, les deux parties de la condition de « forward simulation » peuvent être écrites comme suit :

$$r^{-1};(EvG1|||EvG2) \triangleq \{v \mapsto w' | \exists w \ . \ I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge ((G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \Rightarrow G_1\text{-}Post(w,w') \wedge G_2\text{-}Post(w,w'))\}$$

$$EvG;r^{-1} \triangleq \{v \mapsto w' | \exists v'.I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w')\}$$

4.3.2.2 Identification des obligations de preuve

En se basant sur cette dernière condition de « forward simulation », les obligations de preuve définissant exactement ce qui doit être prouvé pour ce patron sont identifiées. En fait, la traduction de cette condition revient à prouver le séquent suivant, en appliquant la règle d'inférence *AND-L* :

$$\begin{array}{l} I(v) \\ J(v,w) \\ (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \\ (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \Rightarrow (G_1\text{-}Post(w,w') \wedge G_2\text{-}Post(w,w')) \\ \vdash \\ I(v) \\ G\text{-}Guard(v) \\ \exists v'.G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w') \end{array} \qquad \text{(II)}$$

Ce séquent peut être simplifié en utilisant les règles d'inférence *IMP-L* [4] et *AND-L* comme suit :

4. Elle permet de simplifier les prédicats d'implication apparaissant dans les hypothèses d'un séquent. Page 75 de [Abrial 2010]

$$\boxed{\begin{array}{l} I(v) \\ J(v,w) \\ G_1\text{-}Guard(w) \\ G_2\text{-}Guard(w) \\ G_1\text{-}Post(w,w') \wedge G_2\text{-}Post(w,w') \\ \vdash \\ I(v) \\ G\text{-}Guard(v) \\ \exists v'.G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w') \end{array}} \quad \text{(II')}$$

La définition SysML/KAOS d'un raffinement AND nous aide à extraire intuitivement les obligations de preuve suffisantes pour prouver le séquent :

$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \quad \textbf{(PO1)}$$
$$G_2\text{-}Guard \Rightarrow G\text{-}Guard \quad \textbf{(PO2)}$$
$$(G_1\text{-}Post \wedge G_2\text{-}Post) \Rightarrow G\text{-}Post \quad \textbf{(PO3)}$$

4.3.3 Synthèse

Afin de s'assurer que l'entrelacement des événements (**EvG1** ||| **EvG2**) raffine l'événement abstrait **EvG**, trois différents lemmes (bien sûr, en plus des obligations de preuve de faisabilité des événements) doivent être prouvées :

- *Le renforcement de la garde* assure que la garde concrète est plus forte que la garde abstraite. La garde concrète de **EvG1** ||| **EvG2** peut être l'une ou l'autre G_1-*Guard* (si **EvG1** est exécuté d'abord) ou G_2-*Guard* (si **EvG2** est exécuté d'abord). Par conséquent, il faut prouver que :
  $$G_1\text{-}Guard \Rightarrow G\text{-}Guard \quad \textbf{(PO1)}$$
  $$G_2\text{-}Guard \Rightarrow G\text{-}Guard \quad \textbf{(PO2)}$$
- *La simulation d'action* assure que l'entrelacement des événements concrets transforme les variables concrètes d'une manière qui ne contredit pas l'événement abstrait.
  $$(G_1\text{-}Post \wedge G_2\text{-}Post) \Rightarrow G\text{-}Post \quad \textbf{(PO3)}$$

Une autre sémantique de trace est donnée dans [Matoussi *et al.* 2010a] permettant d'identifier ces mêmes obligations de preuve. Elle se base sur la description de tous les comportements possibles.

4.4 L'expression du patron OR en Event-B

4.4.1 Description du patron SysML/KAOS

Rappelons que ce raffinement raffine un but « Achieve » en deux (ou plus) sous-buts si seulement l'un ou l'autre (pas les deux) de ses sous-buts est réalisé, comme le montre la Figure 4.6. Dans chaque sous-but, une condition cible spécifique doit être atteinte pour atteindre **G-TargetCondition**. Rappelons également que ce patron de raffinement ne précise pas quel sous-but sera effectivement choisi et implanté, introduisant ainsi une certaine forme de non-déterminisme borné qui sera résolu plus tard dans la phase d'implémentation.

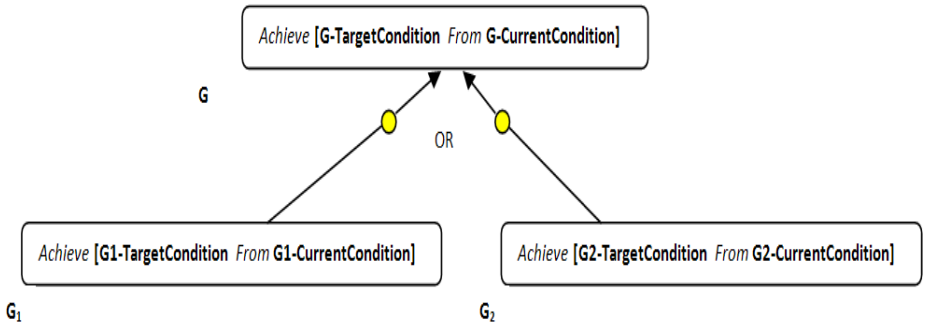


FIGURE 4.6: Le patron de raffinement de but « OR »

4.4.2 Sémantique formelle du patron

4.4.2.1 Définition formelle

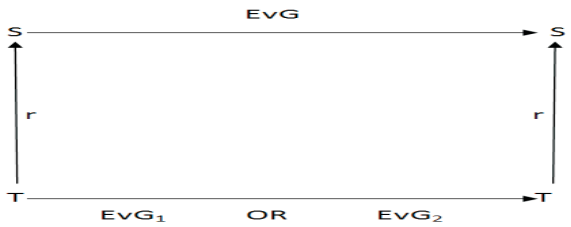
Puisque la satisfaction d'exactement un sous-but SysML/KAOS implique la satisfaction du but parent, l'idée est de raffiner l'événement abstrait **EvG** comme suit :

(**EvG1** *XOR* **EvG2**) Refines **EvG**

Ce raffinement OU-exclusif peut être considéré comme un raffinement OU-inclusif avec une caractéristique supplémentaire d'exclusivité. Cette définition, notée *(*XOR.def), est ainsi présentée comme suit :

$$(\textbf{EvG1}\ \textit{XOR}\ \textbf{EvG2})\ \text{Refines}\ \textbf{EvG} = \begin{cases} (\textbf{EvG1}\ \textit{OR}\ \textbf{EvG2})\ \text{Refines}\ \textbf{EvG} \\ \\ \text{Caractéristique d'exclusivité} \end{cases}$$

Comme pour les autres patrons de raffinement de buts, la première partie de la dernière définition *(XOR.def)* peut être caractérisée en termes de comparaisons des traces, comme illustré dans le schéma ci-dessous.



La condition du « forward simulation » suivante assure que toute trace d'un modèle raffiné doit être aussi une trace de l'abstraction :

$$r^{-1};(EvG1 \text{ OR } EvG2) \subseteq EvG;r^{-1}$$

Pour bien exprimer cette nouvelle sémantique de raffinement, la représentation ensembliste (Figure 4.4) est étendue en exprimant la relation du choix $EvG1$ OR $EvG2$ d'une manière ensembliste, comme suit :

$$EvG1 \text{ OR } EvG2 \triangleq \{w \mapsto w' | \exists v \;.\; I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge ((G_1\text{-}Guard(w) \Rightarrow G_1\text{-}Post(w,w')) \vee (G_2\text{-}Guard(w) \Rightarrow G_2\text{-}Post(w,w')))\}$$

Cette définition exprime qu'on commence dans un état w où les deux événements **EvG1** et **EvG2** pourraient être exécutés, c'est-à-dire que leurs gardes sont vraies. On peut ensuite atteindre un autre état w' si au moins un des deux événements est exécuté. Notons que la caractéristique d'exclusivité entre ces deux événements est une contrainte supplémentaire qui sera considérée plus tard. En se basant sur cette définition, les deux parties de la condition du « forward simulation » peuvent être écrites comme suit :

$$r^{-1};(EvG1 \text{ OR } EvG2) \triangleq \{v \mapsto w' | \exists w \;.\; I(v) \wedge J(v,w) \wedge (G_1\text{-}Guard(w) \wedge G_2\text{-}Guard(w)) \wedge ((G_1\text{-}Guard(w) \Rightarrow G_1\text{-}Post(w,w')) \vee (G_2\text{-}Guard(w) \Rightarrow G_2\text{-}Post(w,w')))\}$$

$$EvG;r^{-1} \triangleq \{v \mapsto w' | \exists v'.I(v) \wedge G\text{-}Guard(v) \wedge G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w')\ \}$$

4.4.2.2 Identification des obligations de preuve

La traduction de la condition « forward simulation » revient donc à prouver le séquent suivant, par l'application de la règle d'inférence *AND-L* :

$$\begin{array}{l} I(v) \\ J(v,w) \\ G_1\text{-}Guard(w) \\ G_2\text{-}Guard(w) \\ ((G_1\text{-}Guard(w) \Rightarrow G_1\text{-}Post(w,w')) \vee (G_2\text{-}Guard(w) \Rightarrow G_2\text{-}Post(w,w'))) \\ \vdash \\ I(v) \\ G\text{-}Guard(v) \\ \exists v'.G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w') \end{array} \quad \text{(III)}$$

L'application de la règle d'inférence *OR-L* [5] puis la règle *IMP-L* et enfin la règle *MON* [6] permet de simplifier le dernier séquent en obtenant les deux séquents suivants :

$$\begin{array}{l} I(v) \\ J(v,w) \\ G_1\text{-}Guard(w) \\ G_1\text{-}Post(w,w') \\ \vdash \\ I(v) \\ G\text{-}Guard(v) \\ \exists v'.G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w') \end{array} \quad \text{(III.1)}$$

La définition d'un raffinement OR en SysML/KAOS nous aide à extraire intuitivement les obligations de preuve suffisantes pour prouver le dernier séquent :

$G_1\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO1)**
$G_1\text{-}Post \Rightarrow G\text{-}Post$ **(PO3)**

$$\begin{array}{l} I(v) \\ J(v,w) \\ G_2\text{-}Guard(w) \\ G_2\text{-}Post(w,w') \\ \vdash \\ I(v) \\ G\text{-}Guard(v) \\ \exists v'.G\text{-}Post(v,v') \wedge I(v') \wedge J(v',w') \end{array} \quad \text{(III.2)}$$

5. Il correspond à la technique classique d'une preuve par cas. Plus précisément, afin de prouver une conclusion sous la disjonction des prémisses $P \vee Q$, il suffit de prouver d'une façon indépendante la même conclusion sous l'hypothèse P et aussi sous l'hypothèse Q. Page 46 de [Abrial 2010]

6. Il exprime que pour avoir une preuve d'un but G sous deux ensemble d'hypothèses $H1$ et $H2$, il suffit d'avoir une preuve de G, sous $H1$ seulement. Page 36 de [Abrial 2010]

Comme pour le premier séquent, ce séquent peut être prouvé en prouvant les obligations de preuve suivantes :

$G_2\text{-}Guard \Rightarrow G\text{-}Guard$ **(PO2)**
$G_2\text{-}Post \Rightarrow G\text{-}Post$ **(PO4)**

La deuxième partie de la définition de *(XOR.def)* qui exprime la caractéristique d'exclusivité revient à prouver ces deux obligations de preuve :

$$G_1\text{-}Post \Rightarrow \neg G_2\text{-}Guard \quad \textbf{(PO5)}$$
$$G_2\text{-}Post \Rightarrow \neg G_1\text{-}Guard \quad \textbf{(PO6)}$$

4.4.3 Synthèse

Afin de s'assurer que **EvG1** XOR **EvG2** raffine l'événement abstrait **EvG**, les obligations de preuve suivantes doivent être prouvées (bien sûr, en plus des obligations de preuve de faisabilité des événements) :

– *Le renforcement de la garde* assure que chaque garde concrète (G_1-*Guard* ou G_2-*Guard*) est plus forte que la garde abstraite de l'événement **EvG**.

$$G_1\text{-}Guard \Rightarrow G\text{-}Guard \quad \textbf{(PO1)}$$
$$G_2\text{-}Guard \Rightarrow G\text{-}Guard \quad \textbf{(PO2)}$$

– *La simulation d'action* assure que chaque événement concret (**EvG1** ou **EvG2**) transforme les variables concrètes d'une manière qui ne contredit pas l'événement abstrait **EvG**.

$$G_1\text{-}Post \Rightarrow G\text{-}Post \quad \textbf{(PO3)}$$
$$G_2\text{-}Post \Rightarrow G\text{-}Post \quad \textbf{(PO4)}$$

– *La caractéristique d'exclusivité* assure que seulement un événement (soit **EvG1** ou **EvG2**) peut être exécuté et pas tous les deux. Par conséquent, l'exécution d'un événement interdit le déclenchement de l'autre événement.

$$G_1\text{-}Post \Rightarrow \neg G_2\text{-}Guard \quad \textbf{(PO5)}$$
$$G_2\text{-}Post \Rightarrow \neg G_1\text{-}Guard \quad \textbf{(PO6)}$$

Notons qu'il est possible de ne pas ajouter explicitement les quatre premières obligations de preuve comme des théorèmes à prouver. En effet, ces obligations de preuve sont générées par la version actuelle de RODIN en considérant que chaque événement concret raffine l'événement abstrait. Néanmoins, les deux dernières obligations de preuve sont ajoutées nécessairement comme des théorèmes pour exprimer la caractéristique d'exclusivité.

4.5 Synthèse et discussion de l'approche

4.5.1 Synthèse de l'approche

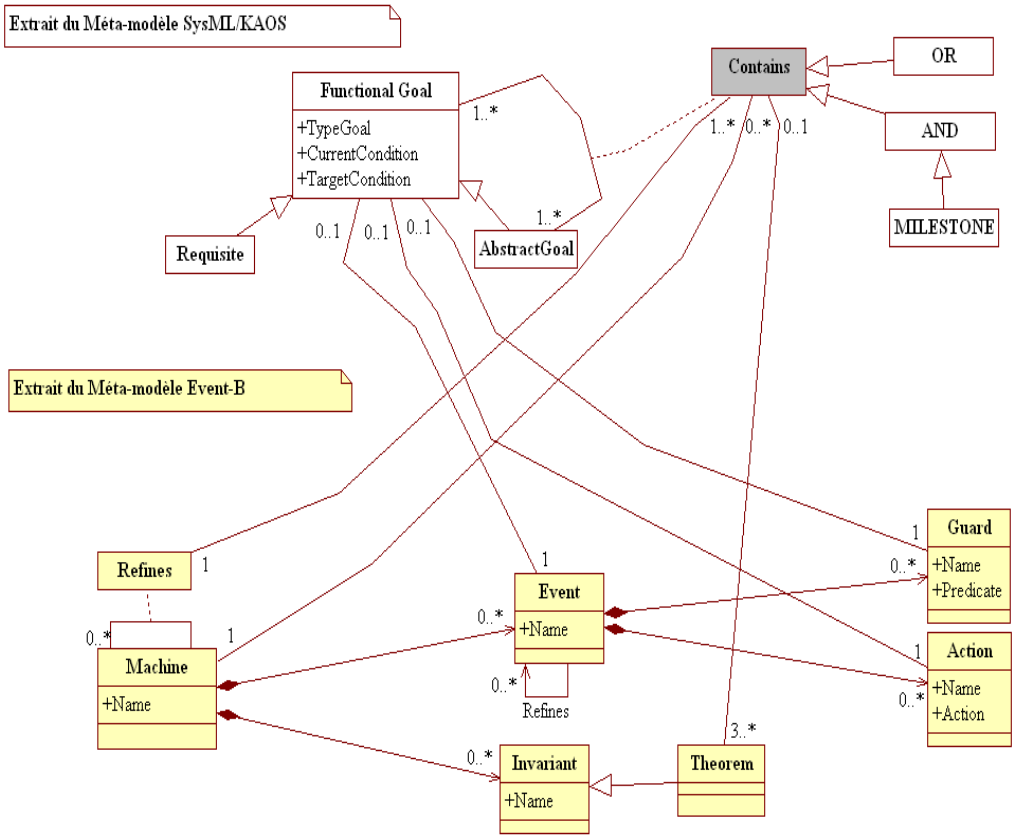


FIGURE 4.7: Combinaisons des méta-modèles SysML/KAOS et Event-B pour les buts fonctionnels

Le méta-modèle de la Figure 4.7 permet de représenter les associations entre le méta-modèle de SysML/KAOS et celui de Event-B [7] pour les buts fonctionnels. Il sert de base à l'outil de traduction SysKAOS2EventB décrit dans le chapitre 6. L'association entre la classe association **Contains** de SysML/KAOS et la classe association **Refines** d'Event-B indique qu'un raffinement d'une machine Event-B correspond à $1..*$ raffinements de buts SysML/KAOS puisque chaque niveau dans la hiérarchie de buts peut contenir plusieurs raffinements. L'association entre la classe association **Contains** et la classe **Machine** permet d'indiquer quant à elle que chaque raffinement de buts SysML/KAOS est associé à une seule machine Event-B. L'autre multiplicité de cette association est mise à $0..*$ car la machine Event-B du niveau le plus abstrait n'est associée à aucun raffinement de buts SysML/KAOS. Les obligations de preuve (au moins au nombre de 3) identifiées

7. http ://wiki.event-b.org/index.php/EMF_framework_for_Event-B

pour chaque raffinement SysML/KAOS seront ajoutées comme des théorèmes en Event-B. C'est pourquoi une association entre la classe association **Contains** de SysML/KAOS et la classe **Theorem** d'Event-B est définie. Ces différentes associations seront complétées par plusieurs autres associations une fois que les buts non-fonctionnels et leurs impacts sur les buts fonctionnels [Gnaho & Semmak 2011] seront pris en compte dans le chapitre 5 de cette thèse.

D'un autre côté, cette approche a nécessité l'expression explicite de la notion d'ordonnancement, d'entrelacement et d'exclusivité entre les événements. C'est un ajout très utile à la sémantique de raffinement d'Event-B. En effet, l'encodage de ces notions est possible dans l'état actuel d'Event-B en utilisant des variables de contrôle intégrées dans les gardes et les actions des événements. L'inconvénient de ces variables de contrôle est l'enchevêtrement avec les spécifications fonctionnelles. Ce dernier complique la traçabilité entre les exigences et les spécifications formelles, comme le confirme [Iliasov 2009]. L'extension du raffinement Event-B (sans les variables de contrôle) rend ces modèles Event-B plus lisibles.

4.5.2 Discussion à propos des propriétés temporelles

On peut se demander si la transformation des **TargetCondition** SysML/KAOS comme des post-conditions Event-B est adéquate puisque l'exécution des événements Event-B n'est pas obligatoire. Dans la sémantique Event-B, tous les événements dont la garde est vraie peuvent être exécutés et il y a nécessairement un événement qui sera exécuté. Le choix se fait de manière non-déterministe. Pour assurer que les contraintes impliquées par les raffinements de buts seront respectées par la spécification en Event-B, nous partons du principe suivant : le but de plus haut niveau est exprimé sous la forme d'un événement abstrait qui est le seul à pouvoir se déclencher. Par conséquent, la sémantique d'Event-B assure que le but sera atteint tôt ou tard. Pour les sous-buts dérivés à partir du but de plus haut niveau, les obligations de preuve permettent d'assurer que les contraintes AND, OR et MILESTONE seront respectées.

Néanmoins, l'étude de la combinaison et de l'interaction entre les différents patrons de raffinement de buts SysML/KAOS révèle que les obligations de preuve ne sont pas toujours suffisantes pour prouver la cohérence globale du modèle de buts et pour vérifier des propriétés de type logique temporelle. En effet, la dérivation d'une spécification Event-B à partir d'un modèle de buts contenant un raffinement MILESTONE qui interfère avec d'autres patrons de raffinement ou autres événements Event-B nécessite de prendre en compte des contraintes temporelles entre les événements. Ces contraintes doivent être nécessairement et explicitement contenues dans la spécification des événements, par exemple dans la garde. Pour mieux expliquer ce point, considérons le modèle de buts SysML/KAOS décrit dans la Figure 4.8.

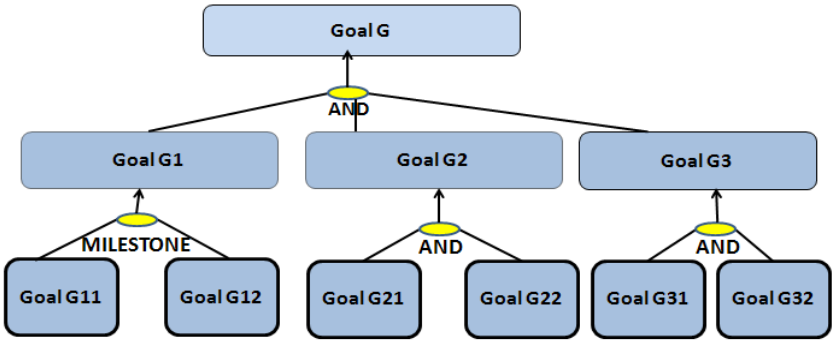


FIGURE 4.8: Un cas de figure d'un modèle de buts SysML/KAOS

Selon notre sémantique Event-B du raffinement AND, les événements du premier raffinement sont entrelacés comme suit :

$$\textbf{EvG1} \mid\mid\mid \textbf{EvG2} \mid\mid\mid \textbf{EvG3}$$

Pour le second raffinement, l'une des traces d'exécution valides est la suivante :

$$\textbf{EvG11} ; \textbf{EvG21} ; \textbf{EvG22} ; \textbf{EvG12} ; \textbf{EvG31} ; \textbf{EvG32}$$

Le principal problème est que l'exécution de **EvG21** ou de **EvG22** peut impliquer que la garde de **EvG12** devienne fausse. Dans ce cas, l'obligation de preuve « contrainte d'ordonnancement » du raffinement MILESTONE ($EvG11\text{-Post} \Rightarrow EvG12\text{-Guard}$) ne pourra jamais être déchargée. L'idée est de relâcher cette obligation de preuve en la remplaçant par la formule de logique temporelle suivante :

$$\Box(EvG11\text{-Post} \Rightarrow \Diamond EvG12\text{-Guard})$$

Cette sémantique correspond à l'opérateur temporel [Manna & Pnueli 1984] *leadsto* ($\rightsquigarrow$) qui exprime qu'une fois $EvG11$-Post vérifiée dans un état alors $EvG12$-Guard deviendra vraie un jour :

$$EvG11\text{-Post} \rightsquigarrow EvG12\text{-Guard}$$

Les techniques classiques de preuve ne peuvent pas vérifier des propriétés de ce type contenant des opérateurs temporels, contrairement aux techniques de « model-checking ». De là, l'idée est de mélanger des obligations de preuve locales (générées par le prouveur d'Event-B) avec des preuves produites par le « model-checking » (en utilisant le model-checker ProB [Leuschel & Butler 2003]). Récemment, [Iliasov 2009] a présenté un premier travail qui propose une extension d'Event-B avec un mécanisme pour raisonner sur l'ordre des événements en se basant uniquement sur les preuves de théorèmes, ce qui évite le problème d'explosion d'états causé par le model-checking. Un certain nombre d'obligations de

preuve sont générées lorsque des flux sont attachés à un modèle Event-B. Le déchargement de ces obligations de preuve démontre l'évolution du modèle Event-B. En outre, l'auteur de [Iliasov 2009] affirme que cette technique de contrôle de flux peut également contribuer à assurer la propriété de non-blocage, de vivacité et d'accessibilité d'un modèle Event-B. Par conséquent, il serait avantageux d'explorer cette technique afin de vérifier les contraintes temporelles entre les événements sans utiliser les techniques de model-checking.

4.6 Application de l'approche sur l'étude de cas

L'approche proposée dans ce chapitre a été validée aussi par l'étude de cas du composant de localisation dont le modèle de buts SysML/KAOS (voir la Figure 3.6) est largement détaillé dans la section 3.2.2.

4.6.1 Machine abstraite

Une machine abstraite Event-B *Localization* (voir la Figure 4.10) est associée au niveau le plus abstrait de la hiérarchie de buts. Dans cette machine Event-B, un événement appelé **LocalizeVehicle** traduit ce but. Afin de faciliter la traçabilité, un commentaire est attaché à cet événement. La localisation d'un véhicule consiste à obtenir une *estimated_loc* qui est un couple (*latitude*, *longitude*). A ce niveau d'abstraction, il n'est pas nécessaire de préciser la façon dont cette information est calculée. Ainsi, nous utilisons la substitution généralisée non déterministe à travers le symbole $:\in$ qui spécifie un choix non borné : *estimated_loc* peut prendre n'importe quelle valeur dans l'ensemble $((LATITUDE \setminus \{null\}) \times (\text{LONGITUDE} \setminus \{null\}))$. La valeur *null* sert à l'initialisation du système. Notons qu'à ce niveau d'abstraction l'événement **LocalizeVehicle** peut toujours se produire. Par conséquent, sa garde est toujours vraie. Notons aussi que le nom de la garde est mis à *CurrentG* et celui de l'action à *TargetG* pour assurer une traçabilité explicite avec la **CurrentCondition** et la **TargetCondition** du but *G*, respectivement. Les ensembles en majuscules sont des ensembles abstraits utilisés pour typer les variables et sont décrits dans le contexte Event-B *Type_sets* (voir la Figure 4.9).

4.6.2 Premier raffinement

La machine de raffinement Event-B *Localization1* (voir la Figure 4.11) est associée au premier niveau de la hiérarchie du graphe de buts SysML/KAOS. Comme pour le niveau abstrait, les sous-buts G_1, G_2 et G_3 sont représentés par trois événements Event-B **CaptureRawLocalizations**, **ValidateData** et **MergeData**, respectivement. Le premier renvoie un ensemble de couples (latitude, longitude), un pour chaque composant utilisé pour la localisation brute d'un véhicule. Le second événement quant à lui filtre l'ensemble retourné des valeurs brutes en choisissant

CONTEXT TypeSets
SETS
SUBCOMPONENTS
SUBSENSORS
CONSTANTS
null, LATITUDE, LONGITUDE
gps, wifi, speed, accel
AXIOMS
axm1 : $partition(SUBCOMPONENTS, \{gps\}, \{wifi\})$
axm2 : $LATITUDE = \mathbb{N} \cup \{null\}$
axm3 : $LONGITUDE = \mathbb{N} \cup \{null\}$
axm4 : $partition(SUBSENSORS, \{speed\}, \{accel\})$
axm5 : $\forall x \cdot x \in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}) \Rightarrow x \in (LATITUDE \times LONGITUDE)$
axm6 : $null \notin \mathbb{N}$
END

FIGURE 4.9: Le context Event-B *TypeSets*

MACHINE Localization // traduit le niveau le plus abstrait du modèle de buts SysML/KAOS
SEES TypeSets
VARIABLES
estimated_loc
INVARIANTS
inv1 : $estimated\_loc \in LATITUDE \times LONGITUDE$
EVENTS
Initialisation
begin
act1 : $estimated\_loc := null \mapsto null$
end
Event *LocalizeVehicle* $\widehat{=}$ // traduit le but fonctionnel G
when
CurrentG : *True*
then
TargetG : $estimated\_loc :\in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})$
end

FIGURE 4.10: La machine abstraite du composant de localisation

les valeurs acceptables. Le dernier événement renvoie la localisation finale calculée à partir des valeurs retournées de **ValidateData**. Comme le montre le commentaire associé à la clause REFINES dans la Figure 4.11, l'événement abstrait **Localize-Vehicle** est raffiné par la séquence de nouveaux événements. Cette sémantique de raffinement nécessite de décharger quatre obligations de preuve (*MilestoneTheo1, MilestoneTheo2, MilestoneTheo3, MilestoneTheo4*) qui sont considérées comme des théorèmes Event-B [8].

8. Pour des raisons de simplification et de lisibilité, on a pris les noms des prédicats au lieu de prendre les valeurs de ces prédicats. Ainsi, l'obligation de preuve *MilestoneTheo3* par exemple dans la Figure 4.11 revient à prouver en réalité $Valeur(CurrentG1) \Rightarrow Valeur(CurrentG)$, qui est $True \Rightarrow True$

MACHINE Localization1 // traduit le premier niveau de raffinement du modèle de buts
REFINES Localization //(CaptureRawLocalizations ; ValidateData ; MergeData) *Refines* LocalizeVehicle
SEES TypeSets
VARIABLES

... // Les variables déjà déclarées

`subcomponents_loc`, `validated_loc`, `merged_loc`

INVARIANTS

`inv1` : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow (LATITUDE \times LONGITUDE)$
`inv2` : $validated\_loc \in SUBCOMPONENTS \nrightarrow (LATITUDE \times LONGITUDE)$
`inv3` : $merged\_loc \in LATITUDE \times LONGITUDE$
`inv4` : $estimated\_loc = merged\_loc$
`MilestoneTheo1` : $TargetG1 \Rightarrow CurrentG2$
`MilestoneTheo2` : $TargetG2 \Rightarrow CurrentG3$
`MilestoneTheo3` : $CurrentG1 \Rightarrow CurrentG$
`MilestoneTheo4` : $TargetG3 \Rightarrow TargetG$

EVENTS
Initialisation
begin

... // Les initialisations déjà faites

`act2` : $subcomponents\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$
`act3` : $validated\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$
`act4` : $merged\_loc := null \mapsto null$

end

Event *CaptureRawLocalizations* $\widehat{=}$ // traduit le but fonctionnel G1
when

`CurrentG1` : $True$

then

`TargetG1` : $subcomponents\_loc :\in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

end

Event *ValidateData* $\widehat{=}$ // traduit le but fonctionnel G2
when

`CurrentG2` : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

then

`TargetG2` : $validated\_loc : |(validated\_loc' \in \mathscr{P}(subcomponents\_loc)) \wedge (ran(validated\_loc') \subseteq ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})))$

end

Event *MergeData* $\widehat{=}$ // traduit le but fonctionnel G3
when

`CurrentG3` : $(validated\_loc \in \mathscr{P}(subcomponents\_loc)) \wedge (ran(validated\_loc) \subseteq ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})))$

then

`TargetG3` : $merged\_loc :\in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})$

end

END

FIGURE 4.11: La première machine de raffinement du composant de localisation

4.6.3 Second raffinement

Maintenant, le deuxième niveau de la hiérarchie de buts SysML/KAOS est considéré. De la même manière, une machine de raffinement Event-B *Localization2* (voir la Figure 4.12) est créée et doit raffiner la machine précédente *Localiza-*

tion1. Ce second raffinement Event-B va encapsuler la traduction de deux patrons de raffinement SysML/KAOS :

- Le raffinement AND : les sous-buts $G_{1.1}$ et $G_{1.2}$ sont représentés par deux événements Event-B **UseGPS** et **UseWIFI** en utilisant les mêmes règles de traduction que pour l'événement **LocalizeVehicle** dans la machine abstraite. Comme le montre le premier commentaire associé à la clause REFINES dans la Figure 4.12, l'événement abstrait **CaptureRawLocalizations** est raffiné par l'entrelacement de deux nouveaux événements. Cette sémantique de raffinement nécessite de prouver trois théorèmes (*AndTheo1*, *AndTheo2*, *AndTheo3*).
- Le raffinement MILESTONE : les sous-buts $G_{2.1}$ et $G_{2.2}$ sont traduits par deux événements Event-B **CaptureRelativeLocalizations** et **FilterData**. Comme le montre le second commentaire associé à la clause REFINES dans la Figure 4.12, l'événement abstrait **ValidateData** est raffiné par la séquence de deux nouveaux événements. Cette sémantique de raffinement nécessite de prouver trois théorèmes (*MilestoneTheo1*, *MilestoneTheo2*, *MilestoneTheo3*).

Notons que l'événement **MergeData** (correspondant à un but-feuille SysML/-KAOS dans la machine précédente *Localization1*) n'est pas modifié et est raffiné par lui-même.

MACHINE Localization2 // traduit le second niveau de raffinement du modèle de buts
REFINES Localization1 //(UseGPS ||| UseWIFI) *Refines* CaptureRawLocalizations
//(CaptureRelativeLocalizations ; FilterData) *Refines* ValidateData
SEES TypeSets
VARIABLES
... // Les variables déjà déclarées
`gps_loc, wifi_loc, sensors_loc, kept_loc`
INVARIANTS
`inv1 :` $gps\_loc \in \{gps\} \rightarrow (LATITUDE \times LONGITUDE)$
`inv2 :` $wifi\_loc \in \{wifi\} \rightarrow (LATITUDE \times LONGITUDE)$
`inv3 :` $subcomponents\_loc = gps\_loc \cup wifi\_loc$
`inv4 :` $sensors\_loc \in SUBSENSORS \nrightarrow (LATITUDE \times LONGITUDE)$
`inv5 :` $kept\_loc \in SUBCOMPONENTS \nrightarrow (LATITUDE \times LONGITUDE)$
`inv6 :` $validated\_loc = kept\_loc$
`AndTheo1 :` $CurrentG11 \Rightarrow CurrentG1$
`AndTheo2 :` $CurrentG12 \Rightarrow CurrentG1$
`AndTheo3 :` $TargetG11 \wedge TargetG12 \Rightarrow TargetG1$ // Elle assure que *subcomponents_loc* (dans l'événement abstrait **CaptureRawLocalizations**) est une fonction totale.
`MilestoneTheo1 :` $TargetG21 \Rightarrow CurrentG22$
`MilestoneTheo2 :` $CurrentG21 \Rightarrow CurrentG2$
`MilestoneTheo3 :` $TargetG22 \Rightarrow TargetG2$
EVENTS
Initialisation
begin
... // Les initialisations déjà faites
`act5 :` $gps\_loc :\in \{gps\} \rightarrow (\{null\} \times \{null\})$
`act6 :` $wifi\_loc :\in \{wifi\} \rightarrow (\{null\} \times \{null\})$
`act7 :` $sensors\_loc :\in SUBSENSORS \rightarrow (\{null\} \times \{null\})$
`act8 :` $kept\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$

end

Event *UseGPS* $\widehat{=}$ // traduit le but fonctionnel G11

when

`CurrentG11 :` *True*

then

`TargetG11 :` $gps\_loc :\in \{gps\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

end

Event *UseWIFI* $\widehat{=}$ // traduit le but fonctionnel G12

when

`CurrentG12 :` *True*

then

`TargetG12 :` $wifi\_loc :\in \{wifi\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

end

Event *CaptureRelativeLocalizations* $\widehat{=}$ // traduit le but fonctionnel G21

when

`CurrentG21 :` $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

then

`TargetG21 :` $sensors\_loc :\mid (sensors\_loc' \in SUBSENSORS \nrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))) \land sensors\_loc' \neq \varnothing$

end

Event *FilterData* $\widehat{=}$ // traduit le but fonctionnel G22

when

`CurrentG22 :` $sensors\_loc \in SUBSENSORS \nrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})) \land sensors\_loc \neq \varnothing$

then

`TargetG22 :` $kept\_loc :\mid (kept\_loc' \in \mathbb{P}(subcomponents\_loc)) \land (ran(kept\_loc') \subseteq ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})))$

end

Event *MergeData* $\widehat{=}$ // Le même que celui dans la machine précédente
refines *MergeData*

END

FIGURE 4.12: La seconde machine de raffinement du composant de localisation

4.6.4 Troisième raffinement

Une machine de raffinement Event-B *Localization3* (voir la Figure 4.13) est associée à ce troisième niveau de la hiérarchie du graphe de buts SysML/KAOS. Comme pour le niveau abstrait, les sous-buts $G_{2.1}$ et $G_{2.2}$ sont représentés par deux événements Event-B **UseSpeedSensor** et **UseAccelerometer**, respectivement. Comme le montre le commentaire associé à la clause REFINES dans la Figure 4.13, l'événement abstrait **CaptureRelativeLocalizations** est raffiné par un choix exclusif entre les deux nouveaux événements. Ce raffinement peut se traduire par le fait de : (i) considérer que chaque nouvel événement raffine l'événement abstrait **CaptureRelativeLocalizations** ; (ii) ajouter deux obligations de preuve comme des théorèmes (*ORTheo1*, *ORTheo2*) afin d'exprimer la caractéristique d'exclusivité. Notons que les événements correspondants à des buts-feuilles SysML/KAOS dans toutes les machines de raffinement précédentes (**UseGPS**, **UseWIFI**, **FiterData** et **MergeData**) ne sont pas modifiés et sont raffinés par eux-mêmes.

MACHINE Localization3 // traduit le troisième niveau de raffinement du modèle de buts
REFINES Localization2 //(UseSpeedSensor *XOR* UseAccelerometer) *Refines* CaptureRelativeLocalizations
SEES TypeSets
VARIABLES

... // Les variables déjà déclarées

`speed_loc, accel_loc`

INVARIANTS

`inv1 :` $speed\_loc \in \{speed\} \rightarrow (LATITUDE \times LONGITUDE)$

`inv2 :` $accel\_loc \in \{accel\} \rightarrow (LATITUDE \times LONGITUDE)$

`inv3 :` $(sensors\_loc = speed\_loc) \vee (sensors\_loc = accel\_loc) \vee (sensors\_loc = speedloc \cup accel\_loc)$

`ORTheo1 :` $TargetG211 \Rightarrow \neg CurrentG212$

`ORTheo2 :` $TargetG212 \Rightarrow \neg CurrentG211$

EVENTS
Initialisation

begin

... // Les initialisation déjà faites

`act9 :` $speed\_loc :\in \{speed\} \rightarrow (\{null\} \times \{null\})$

`act10 :` $accel\_loc :\in \{accel\} \rightarrow (\{null\} \times \{null\})$

end

Event *UseSpeedSensor* $\widehat{=}$ // traduit le but fonctionnel G211
refines *CaptureRelativeLocalizations*

when

`CurrentG21 :` $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

`CurrentG211 :` $accel\_loc \in \{accel\} \rightarrow (\{null\} \times \{null\})$

then

`TargetG211 :` $speed\_loc, sensors\_loc :\mid (speed\_loc' \in \{speed\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))) \wedge sensors\_loc' = speed\_loc'$

end

Event *UseAccelerometer* $\widehat{=}$ // traduit le but fonctionnel G212
refines *CaptureRelativeLocalizations*

when

`CurrentG21 :` $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

`CurrentG212 :` $speed\_loc \in \{speed\} \rightarrow (\{null\} \times \{null\})$

then

`TargetG212 :` $accel\_loc, sensors\_loc :\mid (accel\_loc' \in \{accel\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))) \wedge sensors\_loc' = accel\_loc'$

end

Event *UseGPS* $\widehat{=}$ // Le même que celui dans la machine précédente
refines *UseGPS*

Event *UseWIFI* $\widehat{=}$ // Le même que celui dans la machine précédente
refines *UseWIFI*

Event *FilterData* $\widehat{=}$ // Le même que celui dans la machine précédente
refines *FilterData*

Event *MergeData* $\widehat{=}$ // Le même que celui dans la machine précédente
refines *MergeData*

END

FIGURE 4.13: La troisième machine de raffinement du composant de localisation

4.6.5 Bilan

C'est la dernière étape du processus de raffinement abstrait puisque tous les buts SysmL/KAOS sont soit des exigences, soit des attentes (voir la Figure 3.6). En ce qui concerne l'activité de preuve liée à l'utilisation d'Event-B, 62 obligations de preuve sont générées au total (voir section 5.4.5). En plus des obligations de preuve classiques telles que la faisabilité des événements et la préservation d'invariants de typage, la spécification Event-B a nécessité de prouver 12 théorèmes liés à la sémantique de raffinement proposée.

Il serait intéressant maintenant d'établir la correspondance entre la spécification abstraite Event-B obtenue à partir du modèle de buts SysML/KAOS et les phases postérieures de développement Event-B. En effet, le modèle Event-B obtenu peut être raffiné vers une implémentation B. Comme pour l'approche dédiée au B classique (décrite dans le chapitre 3), ce processus ne concerne que les événements Event-B correspondant aux exigences logicielles (sous la responsabilité des agents logiciels). Les événements Event-B correspondants aux attentes (sous la responsabilité des agents externes) ne seront donc pas implémentés par le futur logiciel. Plus précisément, dans notre étude de cas, les événements Event-B **UseGPS**, **UseWIFI**, **UseSpeedSensor** et **UseAccelerometer** ne sont pas raffinés et ils sont donc implémentés par des composants matériels (un GPS, un WIFI, etc.) dans un véhicule. Seuls les événements Event-B **FilterData** et **MergeData** seront donc raffinés. Par exemple, le raffinement de l'événement Event-B **MergeData** (voir la Figure 4.11) conduit à un logiciel qui implémente l'algorithme choisi pour réaliser la fusion, grâce à l'opération B **OPMerge** comme le montre la Figure 4.14. Dans un premier temps, cette opération récupère toutes les données de localisation des GPS et WIFI grâce à l'appel des opérations *get_lat* et *get_long*. Ensuite, l'appel de l'opération *get_pond* sert à vérifier si les valeurs renvoyées de GPS et WIFI sont validées ou non. Si ces valeurs sont validées, alors *get_pond* retourne une valeur de pondération de valeur 1 (0 sinon). Enfin, l'opération calcule la latitude et la longitude finales sur la base des différentes valeurs de pondération.

```
lat, long ← OPMerge =
  VAR lat_gps, long_gps, lat_wifi, long_wifi, ponderation_gps, ponderation_wifi
  IN
    lat_gps := get_lat(gps_loc)||
    long_gps := get_long(gps_loc) ||
    ponderation_gps := get_pond(gps_loc) ||
    lat_wifi := get_lat(wifi_loc)||
    long_wifi := get_long(wifi_loc) ||
    ponderation_wifi := get_pond(wifi_loc) ;

      lat := ((lat_gps * ponderation_gps) + (lat_wifi * ponderation_wifi)) /
(ponderation_gps + ponderation_wifi)      ||
    long := ((long_gps * ponderation_gps) + (long_wifi * ponderation_wifi)) /
(ponderation_gps + ponderation_wifi)
  END
```

FIGURE 4.14: L'opération B **OPMerge**

Un résultat intéressant est que le lien entre l'opération B **OPMerge** et l'événement abstrait Event-B **MergeData** peut être assuré. Tandis que l'événement abstrait **MergeData** décrit les propriétés que le programme final doit satisfaire, l'opération B **OPMerge** décrit l'algorithme contenu dans le programme. Ainsi, l'événement **MergeData** décrit la manière par laquelle nous pouvons éventuellement juger que le programme final **OPMerge** est adéquat : (i) l'appel des opérations *get_lat*, *get_long* et *get_pond* assure la garde de **MergeData** ; (iii) le résultat final de l'opération *(lat, long)* assure la satisfaction de la post-condition de l'événement abstrait **MergeData**.

4.7 Conclusion

Ce chapitre présente une nouvelle approche constructive et dirigée par les buts qui permet de dériver une spécification formelle abstraite à partir d'un modèle de buts fonctionnels SysML/KAOS. Cette approche montre encore une fois qu'il est possible d'exprimer le modèle de buts SysML/KAOS avec une méthode formelle en restant au même niveau d'abstraction, c'est-à-dire au niveau analyse des exigences. L'approche confirme également que SysML/KAOS peut fournir une aide pour la construction et la structuration des spécifications formelles. Outre sa dimension graphique qui aide les concepteurs à mieux comprendre les exigences, SysML/KAOS : (i) encourage les concepteurs à prendre en compte à la fois le système et son environnement ; (ii) permet aux concepteurs de bien structurer la spécification formelle Event-B en choisissant à chaque fois le bon niveau de raffinement pour introduire les différents événements.

L'approche proposée offre également les mêmes avantages que celles de l'approche dédiée au B classique (voir section 3.3.1) en ce qui concerne surtout la preuve des propriétés de cohérence sur le modèle de buts fonctionnels SysML/-KAOS en prouvant et validant (par animation) la contrepartie formelle Event-B. Par ailleurs, cette approche permet de surmonter les limites de l'approche dédiée au B classique (présentées dans la section 3.3.2) dans la mesure où :

- Event-B n'a pas la notion de signature identique comme en B classique, ce qui permet par exemple d'introduire de nouveaux événements dans le raffinement.
- Event-B n'a pas besoin d'utiliser les paramètres d'entrée ou de sortie pour communiquer avec l'environnement. En fait, Event-B modélise un système avec l'ensemble de toutes les interactions avec son environnement, c'est-à-dire un système fermé.
- La notion de garde est plus proche de la sémantique d'un but SysML/KAOS. De plus, le renforcement des gardes offerte par Event-B est mieux approprié que la notion d'affaiblissement des pré-conditions en B pour exprimer le raffinement des buts. Par exemple, au niveau le plus abstrait, la garde de **LocalizeVehicle** (voir la Figure 4.10) est définie à TRUE pour exprimer que

l'événement est toujours faisable. La garde définitive de **LocalizeVehicle** est construite pendant le processus de raffinement.

L'approche proposée dans ce chapitre nécessite encore un travail d'extension en ce qui concerne surtout la modularité, contrairement à l'approche dédiée au B classique qui explore le mécanisme d'inclusion des machines. En effet, les machines Event-B obtenues peuvent êtres volumineuses incluant beaucoup d'événements et leur analyse devient complexe. L'idée est de bénéficier du mécanisme de la décomposition, récemment ajouté à Event-B [Butler 2009, Abrial 2010] pour réduire la complexité de la spécification Event-B à analyser. C'est un point que nous envisageons de considérer dans nos travaux futurs.

CHAPITRE 5

La prise en compte des buts non-fonctionnels en Event-B

Sommaire

Le cadre général de cette thèse vise à définir un couplage entre un modèle de buts exprimé en SysML/KAOS et des spécifications formelles abstraites, tout en garantissant une distinction entre les buts fonctionnels et ceux non-fonctionnels. Pour cela, nous avons défini dans le chapitre précédent une approche qui se sert du

premier modèle SysML/KAOS (décrivant les buts fonctionnels) pour dériver des spécifications formelles abstraites Event-B. Ce chapitre prolonge ce travail en présentant les différentes manières de prendre en compte les buts non-fonctionnels et leurs impacts dans les modèles abstraits Event-B déjà obtenus. Des liens de correspondance sont également définis entre les buts non-fonctionnels et les différents éléments Event-B afin de faciliter la gestion de l'évolution des exigences non-fonctionnelles. Le travail présenté dans ce chapitre a fait l'objet de la publication [Matoussi & Laleau 2011b].

5.1 Les buts non-fonctionnels SysML/KAOS et leur prise en compte en Event-B

5.1.1 Le modèle de buts non-fonctionnels en SysML/KAOS

Rappelons qu'un but non-fonctionnel en SysML/KAOS (voir la Figure 5.2 et la section 1.2.7.2) peut être soit un but abstrait (méta-classe **Abstract NFG**), soit un but élémentaire (méta-classe **Elementary NFG**). Un but élémentaire peut être associé à plusieurs buts de contribution (méta-classe **Contribution Goal**) où chaque but de contribution représente une façon de satisfaire le but élémentaire. La classe association **Impact** permet de relier les buts non-fonctionnels aux buts fonctionnels. Elle représente le fait qu'un but de contribution a un effet (+ ou -) sur un but fonctionnel. Dans cette thèse, nous nous intéressons seulement aux impacts positifs (+). Les impacts négatifs (-) seront étudiés dans des futurs travaux.

Pour mieux illustrer ces concepts, considérons le modèle de buts SysML/KAOS du composant de localisation (voir la Figure 5.1). Ce modèle de buts est obtenu par la construction en parallèle du modèle de buts fonctionnels et du modèle de buts non-fonctionnels. Le modèle de buts fonctionnels est largement décrit dans la section 3.2.2. Dans ce qui suit, nous décrivons un extrait du modèle de buts non-fonctionnels traitant seulement de la précision des données de localisation. D'autres buts non-fonctionnels tels que le temps de réponse, la confiance de ces données et la consommation d'énergie peuvent enrichir ce modèle de buts.

La Figure 5.1 montre que le but de contribution $NFR1$ est une contribution directe positive au but élémentaire non-fonctionnel *Precision [Localization data]*. Ce but de contribution est raffiné en trois buts de contribution $NFR11$, $NFR12$, $NFR13$. Le but de contribution $NFR11$ est raffiné lui-même en deux buts de contribution $NFR111$ et $NFR112$. Ces différents buts de contribution et leurs éventuels impacts sur les buts fonctionnels sont définis comme suit :

Contribution Goal $NFR1$: *Obtain good precision*

InformalDef : Il impacte positivement et directement le but fonctionnel G. Ce dernier doit satisfaire que la localisation retournée ait une bonne précision.

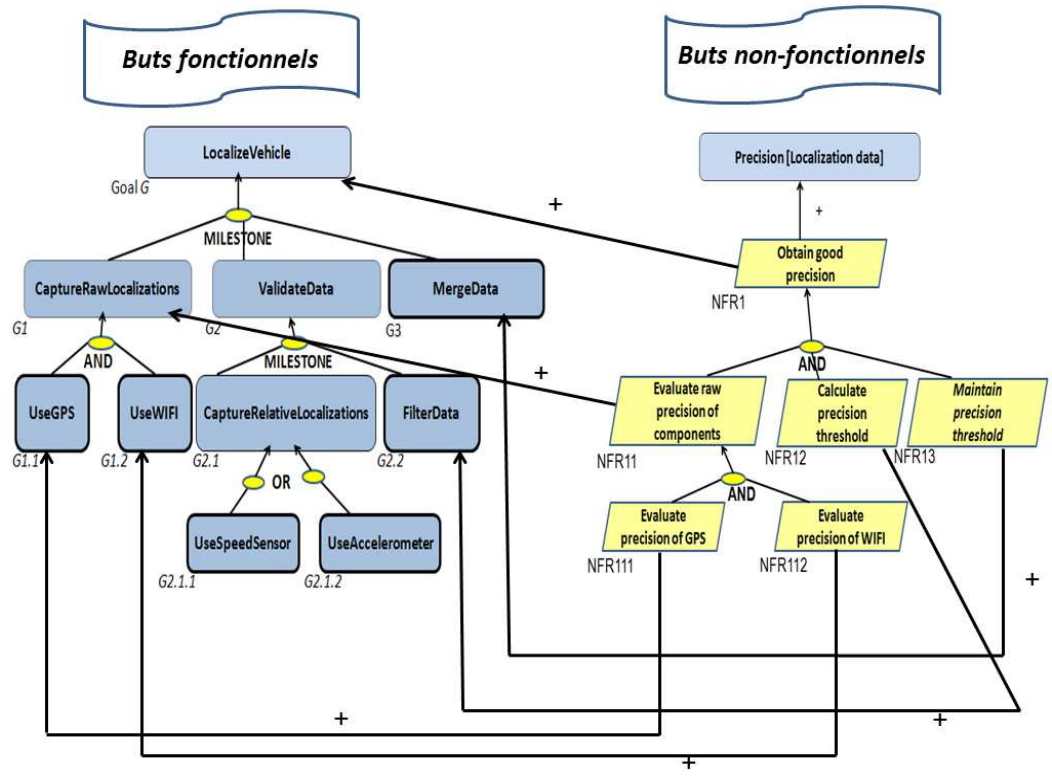


FIGURE 5.1: Un extrait du modèle de buts SysML/KAOS (fonctionnels et non-fonctionnels) du composant de localisation

Contribution Goal $NFR11$: *Evaluate raw precision of components*

InformalDef : Il impacte positivement et directement le but fonctionnel G_1 qui doit satisfaire que les composants retournent des valeurs exprimant la précision.

Contribution Goal $NFR12$: *Calculate precision threshold*

InformalDef : Il impacte positivement et directement le but fonctionnel $G_{2.2}$ qui ne peut pas être réalisé sans le calcul d'un seuil de précision.

Contribution Goal $NFR13$: *Maintain precision threshold*

InformalDef : Il impacte positivement et directement le but fonctionnel G_3. Ce but de contribution indique qu'il faut s'assurer que la localisation finale retournée par le but fonctionnel G_3 doit être toujours au moins égale à un seuil de précision.

Contribution Goal $NFR111$: *Evaluate precision of GPS*

InformalDef : Il impacte positivement et directement le but fonctionnel $G_{1.1}$. Le GPS doit retourner une valeur de précision.

Contribution Goal $NFR112$: *Evaluate precision of WIFI*

InformalDef : Il impacte positivement et directement le but fonctionnel $G_{1.2}$. Le WIFI doit retourner une valeur de précision.

5.1.2 La prise en compte des buts non-fonctionnels en Event-B

Les boîtes grises de la Figure 5.2 montre qu'un but non-fonctionnel, et plus précisément un but de contribution, peut être pris en compte en Event-B de différentes manières :

- *Un événement* : Un but de contribution peut se traduire en Event-B par un nouvel événement. Cela concerne surtout les buts de contribution exprimant qu'une opération doit être réalisée.
- *Des actions dans des événements existants* : Un but de contribution peut se traduire par une nouvelle action d'un événement lié à un but fonctionnel déjà existant. Ce type de traduction concerne souvent les buts fonctionnels impactés par la nécessité de satisfaire une action exprimée par le but de contribution.
- *Des gardes dans des événements existants* : Un but de contribution peut se traduire par une nouvelle garde d'un événement lié à un but fonctionnel déjà existant.
- *Des paramètres dans des événements existants* : Lorsqu'il impacte une action ou une garde d'un événement existant, un but de contribution peut faire apparaître de nouvelles variables locales (les paramètres).
- *Des variables* : Un but de contribution peut ajouter de nouvelles variables.
- *Des invariants* : L'ajout de nouvelles variables implique l'ajout des invariants de typage de ces variables. De plus, si le but de contribution exprime une propriété à maintenir, il se traduit alors par un nouvel invariant (une contrainte).
- *Des éléments d'un contexte* : Un but de contribution peut se traduire dans le contexte Event-B par l'ajout de constantes, d'ensembles ou d'axiomes. Ce type de traduction concerne souvent les buts de contribution qui n'ont aucun impact sur les buts fonctionnels. En effet, ces buts de contributions sont généralement vus comme des propriétés du domaine (des déclarations descriptives sur l'environnement).

Notons qu'un seul but de contribution peut se traduire par un ou plusieurs concepts Event-B à la fois. Une autre remarque importante concerne les buts de contribution exprimant des propriétés de vivacité. Ces buts ne peuvent pas être exprimés comme des invariants Event-B mais seront considérés plutôt comme des expressions en logique temporelle linéaire LTL [Pnueli 1977] ou en logique temporelle branchante CTL [Clarke & Emerson 1981]. Ces expressions sont vérifiables par model-checking, en utilisant par exemple le model-checker ProB [Leuschel & Butler 2003].

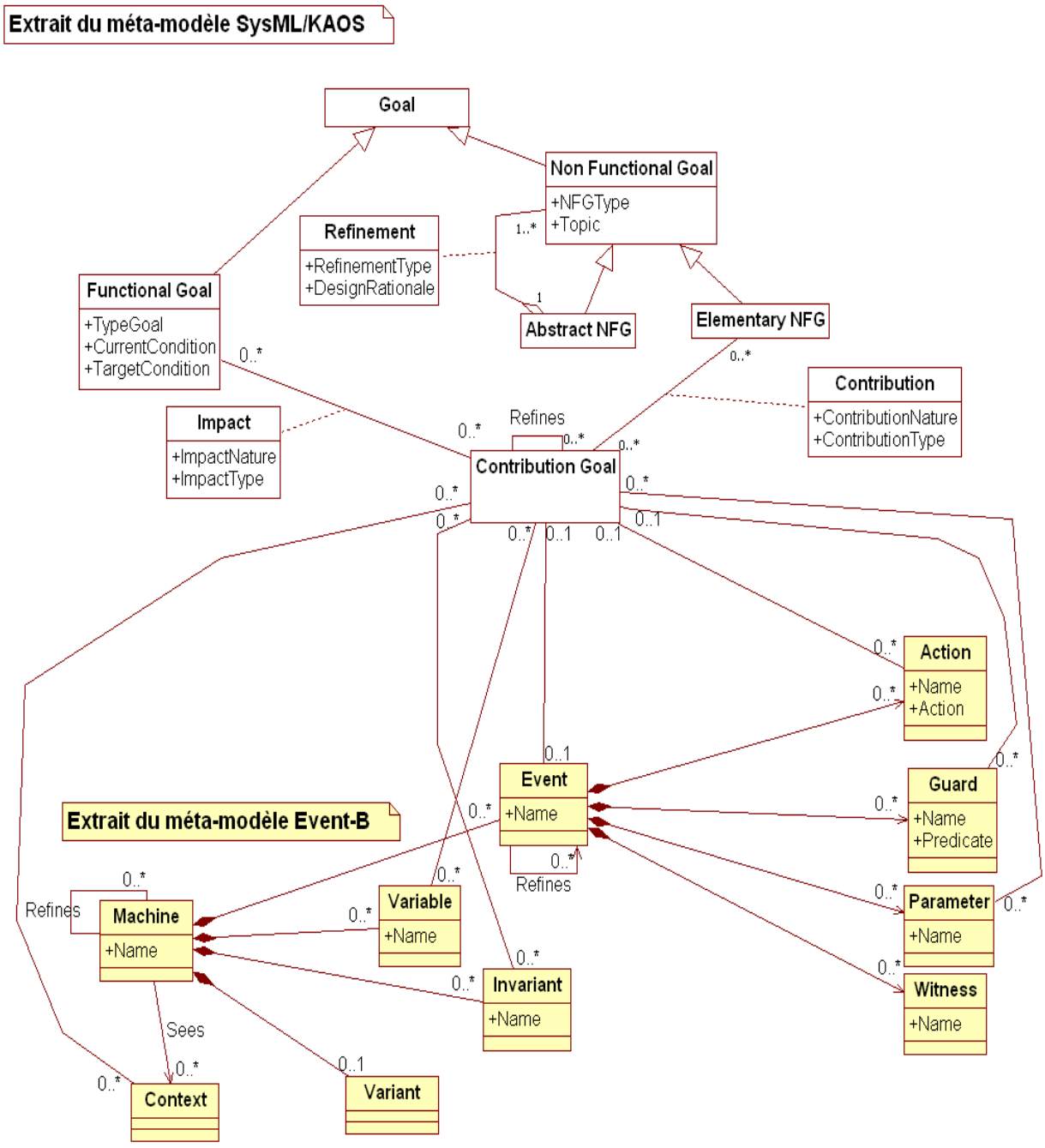


FIGURE 5.2: Un aperçu général des liens entre les méta-modèles SysML/KAOS et Event-B pour les buts non-fonctionnels

5.1.3 Méthodologie de la transformation du modèle de buts fonctionnels et non-fonctionnels en Event-B

Tandis que le modèle de buts SysML/KAOS est obtenu généralement par la construction parallèle du modèle des buts fonctionnels et du modèle des buts non-fonctionnels comme le souligne [Gnaho & Semmak 2011], le processus de transformation de ce modèle en Event-B peut se faire de deux manières :

1. *La transformation parallèle* : Cela veut dire que pour chaque niveau de la hiérarchie de buts fonctionnels SysML/KAOS, il faut : (i) faire la transformation en Event-B des buts fonctionnels de ce niveau ; (ii) prendre en compte les buts non-fonctionnels impactant les buts fonctionnels de ce niveau.
2. *La transformation séquentielle* : Cela veut dire qu'il faut : (i) transformer tout le modèle de buts fonctionnels en Event-B ; (ii) prendre en compte les buts non-fonctionnels et leurs impacts en Event-B d'une manière incrémentale pour chaque niveau de raffinement. Notons que cette prise en compte des buts non-fonctionnels ne doit pas forcément commencer depuis la machine abstraite Event-B vers la dernière machine de raffinement (*une technique top-down*). Il est même utile dans certains cas d'assurer cette prise en compte selon *une technique bottom-up* ; c'est-à-dire en commençant par la dernière machine de raffinement Event-B et en remontant jusqu'à la machine abstraite. Il est possible également de mixer ces deux techniques.

Nous recommandons la transformation séquentielle afin d'obtenir tout d'abord une contrepartie formelle Event-B du modèle de buts fonctionnels, indépendamment de la considération des buts non-fonctionnels. Comme le montre le chapitre 4 de cette thèse, cela va permettre de vérifier et valider le modèle de buts, en ce qui concerne surtout le raffinement de buts. Cela n'est pas possible avec une transformation parallèle où les deux types de buts sont mélangés. Toutefois, la transformation séquentielle (à la différence de la transformation parallèle) peut obliger le concepteur à refaire l'activité de preuve en Event-B. En effet, les injections des buts non-fonctionnels dans les machines Event-B peuvent avoir des effets sur les obligations de preuve déjà faites lors de la transformation de buts fonctionnels seuls. Pour cela, il est nécessaire d'étudier les différents cas d'injection des buts non-fonctionnels dans les machines Event-B. Cette étude a révélé jusqu'à maintenant un certain nombre de règles, de recommandations et de techniques permettant de préserver au maximum les obligations de preuve déjà faites sans avoir à les refaire, comme le montrent les sections 5.2 et 5.3.

5.2 Effet sur les OP suite à l'ajout de gardes et d'actions dans des événements déjà existants

Les obligations de preuve identifiées lors de la transformation des buts fonctionnels en Event-B sont considérées comme des théorèmes qui prennent la forme d'une implication logique $A \Rightarrow B$ où A et B sont appelés l'antécédent et le conséquent, respectivement. Si l'ajout d'information dans l'antécédent (le remplacement de A par A et C par exemple) maintient la vérité de la formule $A \Rightarrow B$, une modification dans le conséquent peut lui faire perdre sa vérité. Prenons par exemple *la contrainte d'ordonnancement* du raffinement MILESTONE en Event-B (voir la Figure 5.3) qui est comme suit :

$$G_1\text{-}Post \Rightarrow G_2\text{-}Guard \qquad \textbf{(PO1)}$$

La preuve de ce théorème est maintenue tant que le conséquent G_2-*Guard* (la garde de l'événement **EvG2**) n'est pas enrichi. En d'autres termes, la vérité du théorème est perdue quand la garde de l'événement **EvG2** est modifiée par l'ajout d'une nouvelle garde, nommée par exemple G_2-$GrdNFR$. Dans ce cas, le théorème devient comme suit :

$$G_1\text{-}Post \Rightarrow G_2\text{-}Guard \wedge G_2\text{-}GrdNFR \qquad \textbf{(PO1')}$$

La définition logique de l'implication permet d'obtenir ces deux théorèmes :

$$G_1\text{-}Post \Rightarrow G_2\text{-}Guard \qquad \textbf{(PO1)}$$
$$G_1\text{-}Post \Rightarrow G_2\text{-}GrdNFR \qquad \textbf{(PO1\_NFR)}$$

Cela montre que l'ajout d'une garde dans l'événement **EvG2** n'impacte pas l'obligation de preuve **PO1** déjà faite. Néanmoins, il faut prouver une obligation de preuve supplémentaire **(PO1_NFR)**.

Dans les sous-sections suivantes, nous appliquons le même raisonnement sur toutes les obligations de preuve (les théorèmes) liées aux différents raffinements. Cela a permis de montrer que l'ajout de gardes et d'actions dans des événements déjà existants n'impacte pas les obligations de preuve déjà faites. Néanmoins, chaque raffinement doit prouver des théorèmes supplémentaires.

5.2.1 Le raffinement MILESTONE

La Figure 5.3 rappelle ce que nous devons prouver pour ce patron (voir la section 4.2.3) pour s'assurer que la séquence d'événements (**EvG1 ;EvG2**) raffine l'événement abstrait **EvG**.

MACHINE Abstract M_0 **EVENTS** **Event** **EvG** $\widehat{=}$ **when** *G-Guard* **then** *G-Post* **end**	**MACHINE** First M_1 **REFINES** Abstract M_0 // (**EvG1** ; **EvG2**) Refines **EvG** **INVARIANTS** *PO1* : G_1-*Post* $\Rightarrow$ G_2-*Guard* *PO2* : G_1-*Guard* $\Rightarrow$ *G-Guard* *PO3* : G_2-*Post* $\Rightarrow$ *G-Post* **EVENTS** **Event** **EvG1** $\widehat{=}$ **when** G_1-*Guard* **then** G_1-*Post* **end** **Event** **EvG2** $\widehat{=}$ **when** G_2-*Guard* **then** G_2-*Post* **end**
(a) Modèle Abstrait M_0	(b) Modèle Raffiné M_1

FIGURE 5.3: L'expression Event-B du raffinement MILESTONE et ses obligations de preuve

En plus des obligations de preuve déjà faites, le raffinement MILESTONE nécessite de prouver ces théorèmes supplémentaires :

– Si la garde de l'événement concret **EvG2** est enrichie par une nouvelle garde G_2-*GrdNFR*, alors il faut prouver que :

$$G_1\text{-}Post \Rightarrow G_2\text{-}GrdNFR \qquad \textbf{(PO1\_NFR)}$$

Pour pouvoir prouver cette nouvelle obligation de preuve, l'idée est que la post-condition G_1-*Post* (relative à l'événement **EvG1**) puisse être également enrichie par une nouvelle action qui implique la garde nommée G_2-*GrdNFR*. Ainsi, l'ajout d'une nouvelle garde dans un événement appartenant à une séquence d'événements qui raffine un événement abstrait nécessite l'ajout d'une nouvelle action (qui doit impliquer la garde) dans l'événement qui le précède dans cette séquence.

– Si la garde de l'événement abstrait **EvG** est enrichie par une nouvelle garde *G-GrdNFR*, alors il faut prouver que :

$$G_1\text{-}Guard \Rightarrow G\text{-}GrdNFR \qquad \textbf{(PO2\_NFR)}$$

Pour pouvoir prouver ce nouveau théorème, l'idée est que la garde du premier événement dans la séquence (ici G_1-*Guard*) puisse être aussi enrichie par une nouvelle garde qui implique la garde nommée *G-GrdNFR*.

– Si la post-condition de l'événement abstrait **EvG** est enrichie par une nouvelle action *G-ActNFR*, alors il faut prouver que :

$$G_2\text{-}Post \Rightarrow G\text{-}ActNFR \qquad \textbf{(PO3\_NFR)}$$

Pour pouvoir prouver ce nouveau théorème qui assure la simulation d'action, l'idée est que l'action du dernier événement dans la séquence (ici G_2-*Post*) puisse être aussi modifiée en ajoutant une nouvelle action qui implique l'action*G-ActNFR*.

5.2.2 Le raffinement AND

La Figure 5.4 rappelle ce que nous devons prouver pour ce patron (voir la section 4.3.3) pour s'assurer que l'entrelacement des événements (**EvG1** ||| **EvG2**) raffine l'événement abstrait **EvG**.

MACHINE Abstract M_0 **EVENTS** **Event** **EvG** $\widehat{=}$ **when** *G-Guard* **then** *G-Post* **end**	**MACHINE** First M_1 **REFINES** Abstract M_0 // (**EvG1** ||| **EvG2**) Refines **EvG** **INVARIANTS** *PO1* : G_1-*Guard* $\Rightarrow$ *G-Guard* *PO2* : G_2-*Guard* $\Rightarrow$ *G-Guard* *PO3* : $(G_1\text{-}Post \wedge G_2\text{-}Post) \Rightarrow$ *G-Post* **EVENTS** **Event** **EvG1** $\widehat{=}$ **when** G_1-*Guard* **then** G_1-*Post* **end** **Event** **EvG2** $\widehat{=}$ **when** G_2-*Guard* **then** G_2-*Post* **end**
(a) Modèle Abstrait M_0	(b) Modèle Raffiné M_1

FIGURE 5.4: L'expression Event-B du raffinement AND et ses obligations de preuve

L'ajout des gardes et des actions dans des événements déjà existants n'impacte pas les obligations de preuve, déjà faites. Néanmoins, le raffinement AND nécessite de prouver ces théorèmes supplémentaires :

– Si la garde de l'événement abstrait **EvG** est enrichie par une nouvelle garde *G-GrdNFR*, alors il faut prouver que :

$$G_1\text{-}Guard \Rightarrow G\text{-}GrdNFR \qquad \textbf{(PO1\_NFR)}$$
$$G_2\text{-}Guard \Rightarrow G\text{-}GrdNFR \qquad \textbf{(PO2\_NFR)}$$

Pour pouvoir prouver ces nouveaux théorèmes, l'idée est que la garde de chaque événement de l'entrelacement (ici G_1-*Guard* et G_2-*Guard*) puisse être aussi enrichie par une nouvelle garde qui implique la garde nommée *G-GrdNFR*.

– Si la post-condition de l'événement abstrait **EvG** est enrichie par une nouvelle action *G-ActNFR*, alors il faut prouver que :

$$(G_1\text{-}Post \wedge G_2\text{-}Post) \Rightarrow G\text{-}ActNFR \qquad \textbf{(PO3\_NFR)}$$

Afin de prouver ce nouveau théorème qui assure la simulation d'action, l'idée est que la post-condition d'au moins un événement de l'entrelacement (ici G_1-*Post* ou G_2-*Post*) puisse être aussi enrichie en ajoutant une nouvelle action qui implique l'action *G-ActNFR*.

5.2.3 Le raffinement OR

La Figure 5.5 rappelle ce que nous devons prouver pour ce patron (voir la section 4.4.3) pour s'assurer que (**EvG1** XOR **EvG2**) raffine l'événement abstrait **EvG**.

MACHINE Abstract M_0 **EVENTS** **Event** **EvG** **when** *G-Guard* **then** *G-Post* **end**	**MACHINE** First M_1 **REFINES** Abstract M_0 // (**EvG1** XOR **EvG2**) Refines **EvG** **INVARIANTS** *PO1* : G_1-*Guard* $\Rightarrow$ *G-Guard* *PO2* : G_2-*Guard* $\Rightarrow$ *G-Guard* *PO3* : G_1-*Post* $\Rightarrow$ *G-Post* *PO4* : G_2-*Post* $\Rightarrow$ *G-Post* *PO5* : G_1-*Post* $\Rightarrow \neg G_2$-*Guard* *PO6* : G_2-*Post* $\Rightarrow \neg G_1$-*Guard* **EVENTS** **Event** **EvG1** $\widehat{=}$ **when** G_1-*Guard* **then** G_1-*Post* **end** **Event** **EvG2** $\widehat{=}$ **when** G_2-*Guard* **then** G_2-*Post* **end**
(a) Modèle Abstrait M_0	(b) Modèle Raffiné M_1

FIGURE 5.5: L'expression Event-B du raffinement OR et ses obligations de preuve

En plus des obligations de preuve déjà faites, le raffinement OR nécessite de prouver de nouveaux théorèmes :

– Si la garde de l'événement abstrait **EvG** est enrichie par une nouvelle garde *G-GrdNFR*, alors il faut prouver que :

$$G_1\text{-}Guard \Rightarrow G\text{-}GrdNFR \qquad \textbf{(PO1\_NFR)}$$
$$G_2\text{-}Guard \Rightarrow G\text{-}GrdNFR \qquad \textbf{(PO2\_NFR)}$$

Pour pouvoir prouver ces nouveaux théorèmes, l'idée est que la garde de chaque événement participant au raffinement (ici G_1-*Guard* et G_2-*Guard*) puisse être aussi enrichie par une nouvelle garde qui implique la garde nommée *G-GrdNFR*.

– Si la post-condition de l'événement abstrait **EvG** est enrichie par une nouvelle action *G-ActNFR*, alors il faut prouver que :

$$G_1\text{-}Post \Rightarrow G\text{-}ActNFR \qquad \textbf{(PO3\_NFR)}$$
$$G_2\text{-}Post \Rightarrow G\text{-}ActNFR \qquad \textbf{(PO4\_NFR)}$$

Afin de prouver ces nouveaux théorèmes assurant la simulation d'action, l'idée est que la post-condition de chaque événement participant au raffinement (ici G_1-*Post* et G_2-*Post*) puisse être aussi enrichie en ajoutant une nouvelle action qui implique l'action *G-ActNFR*.

5.3 Effet sur les OP suite à l'ajout de nouveaux événements

La prise en compte des buts non-fonctionnels en Event-B peut se traduire également par l'ajout de nouveaux événements. Cette prise en compte a révélé jusqu'à maintenant quelques cas de figures qui peuvent être regroupés en trois catégories.

5.3.1 Première catégorie : l'événement ajouté précède l'événement impacté

Supposons que l'un des événements participant au raffinement (par exemple l'événement **EvG2** dans la Figure 5.3) soit impacté par la nécessité d'ajouter un nouvel événement **EvNFR** ayant une post-condition *Post-NFR* et que l'événement ajouté **EvNFR** précède l'événement **EvG2** dans la séquence des événements qui devient comme suit :

(**EvG1** ; (**EvNFR** ; **EvG2**)) *Refines* **EvG**

Il est clair que l'interférence de ce nouvel événement **EvNFR** avec les événements fonctionnels déjà existants peut avoir des effets sur les obligations de preuve déjà faites. Pour faire face à ce problème, nous proposons la technique de *la redondance de garde* qui consiste à : (i) affecter la garde G_2-*Guard* relative à l'événement impacté **EvG2** à la garde de l'événement ajouté **EvNFR** ; (ii) renforcer la garde de l'événement impacté **EvG2** par une nouvelle garde G_2-$GrdNFR$ afin de s'assurer que cet événement ne peut se déclencher qu'après l'exécution de l'événement **EvNFR** (sa post-condition *Post-NFR* devient vraie). Avec une telle technique, toutes les obligations de preuve déjà faites sont préservées. Néanmoins, il faut prouver une obligation de preuve supplémentaire afin d'exprimer la nouvelle contrainte d'ordonnancement entre l'événement ajouté **EvNFR** et l'événement impacté **EvG2** comme suit :

$$Post\text{-}NFR \Rightarrow G_2\text{-}GrdNFR \qquad \textbf{(PO\_NFR)}$$

Il est évident qu'avec une telle technique, nous retombons sur le cas présenté dans la section 5.2 où il y a un enrichissement de la garde de l'événement impacté **EvG2**. Par conséquent, des obligations de preuve supplémentaires peuvent être ajoutées si l'événement **EvG2** est raffiné par la suite. La technique de *la redondance de garde* peut être également appliquée à d'autres cas de figures similaires tels que :

(**EvG1** ||| (**EvNFR** ; **EvG2**)) *Refines* **EvG**

ou encore

(**EvG1** XOR (**EvNFR** ; **EvG2**)) *Refines* **EvG**

Pour une meilleure illustration, cette technique est utilisée dans le second raffinement de l'étude de cas (voir la section 5.4.3) de ce chapitre.

5.3.2 Deuxième catégorie : l'événement ajouté s'entrelace avec l'événement impacté

Supposons que l'un des événements (par exemple l'événement **EvG2** dans la Figure 5.3) soit impacté par la nécessité d'ajouter un nouvel événement **EvNFR** et que ce dernier s'entrelace avec l'événement **EvG2** comme suit :

(**EvG1** ; (**EvNFR** ||| **EvG2**)) *Refines* **EvG**

Pour ce cas de figure, nous proposons la technique de *la fusion-absorption* qui permet à l'événement impacté **EvG2** d'absorber le comportement de l'événement ajouté **EvNFR**, c'est-à-dire prendre la garde et la post-condition de l'événement **EvNFR**. Ce dernier disparaît et toutes les obligations de preuve déjà faites sont préservées.

D'une manière analogue à la technique de la *redondance de garde*, l'application de la technique de *la fusion-absorption* fait que nous retombons sur le cas présenté dans la section 5.2 où il y a un enrichissement de la garde et de la post-condition de l'événement impacté **EvG2**. Par conséquent, des obligations de preuves supplémentaires peuvent être ajoutées si l'événement **EvG2** est raffiné par la suite. Notons que cette technique peut être également appliquée à d'autres cas de figures similaires tels que :

(**EvG1** ||| (**EvG2** ||| **EvNFR**)) *Refines* **EvG**
(**EvG1** *XOR* (**EvNFR** ||| **EvG2**)) *Refines* **EvG**

5.3.3 Troisième catégorie : un choix exclusif entre l'événement ajouté et l'événement impacté

Supposons que la prise en compte d'un but non-fonctionnel en Event-B impacte l'un des événements (par exemple l'événement **EvG2** dans la Figure 5.3) par l'ajout d'un nouvel événement **EvNFR** comme suit :

(**EvG1** ; (**EvNFR** *XOR* **EvG2**)) *Refines* **EvG**

Ce type d'enrichissement ne doit être jamais autorisé dans la mesure où ça peut contredire les buts fonctionnels. En fait, l'ajout d'un tel événement **EvNFR** peut empêcher le déclenchement de l'événement impacté **EvG2**. Or, ce dernier est lié à un buts fonctionnel, c'est-à-dire il exprime ce qu'on attend « obligatoirement » du futur système. Il faut donc s'assurer toujours que cet événement est exécuté. Notons que cette interdiction doit être appliquée également à d'autres cas de figures similaires tels que :

(**EvG1** ||| (**EvNFR** *XOR* **EvG2**)) *Refines* **EvG**
(**EvG1** *XOR* (**EvG2** *XOR* **EvNFR**)) *Refines* **EvG**

5.4 Illustration de l'approche sur l'étude de cas

L'approche proposée est appliquée sur le modèle de buts SysML/KAOS du composant de localisation (voir la Figure 5.1). Nous décrivons dans ce qui suit comment prendre en compte les buts de contribution et leurs impacts en adoptant un processus de transformation séquentiel. En d'autres termes, ces buts de contribution et leurs impacts sont injectés dans les différentes machines Event-B déjà obtenues à partir du modèle de buts fonctionnels (voir section 4.6). Nous choisissons que cette injection se réalise selon *une technique top-down*, c'est-à-dire depuis la machine abstraite Event-B jusqu'à la dernière machine de raffinement.

5.4.1 Retour sur la machine abstraite

Comme l'impact du but de contribution $NFR1$ sur le but fonctionnel G exprime une donnée à satisfaire, l'événement Event-B traduisant le but G **LocalizeVehicle** ajoute une nouvelle action *actNFR1* qui va assurer que la valeur de la précision est positive ($precision :\in \mathbb{N}_1$). Comme le montre la Figure 5.6, une nouvelle variable *precision*, un nouvel invariant de typage *invNFR1* et une initialisation de cette variable *actiniNFR1* sont également ajoutés.

MACHINE Localization // traduit le niveau le plus abstrait du modèle de buts SysML/KAOS
SEES TypeSets
VARIABLES
`estimated_loc`
`precision` // Variable créée par NFR1
INVARIANTS
`inv1 :` $estimated\_loc \in LATITUDE \times LONGITUDE$
`invNFR1 :` $precision \in \mathbb{N}$ // Invariant de typage pour NFR1
EVENTS
Initialisation
begin
`act1 :` $estimated\_loc := null \mapsto null$
`actiniNFR1 :` $precision := 0$
end
Event *LocalizeVehicle* $\widehat{=}$ // Evénement impacté par NFR1
when
`CurrentG :` *True*
then
`TargetG :` $estimated\_loc :\in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})$
`actNFR1 :` $precision :\in \mathbb{N}_1$
end

FIGURE 5.6: La prise en compte des buts non-fonctionnels dans la machine abstraite Event-B du composant de localisation

5.4.2 Retour sur la première machine de raffinement

Comme l'impact du but de contribution $NFR11$ sur le but fonctionnel G_1 exprime une donnée à satisfaire, l'événement Event-B traduisant le but G_1 **Captu-**

reRawLocalizations ajoute une nouvelle action *actNFR11* qui va assurer la satisfaction de cette donnée. Comme le montre la Figure 5.7, une nouvelle variable *raw_precision*, un nouvel invariant de typage *invNFR11* et une initialisation de cette variable *actiniNFR11* sont également ajoutés.

D'un autre côté, le but fonctionnel G_3 est impacté par le but de contribution $NFR13$ qui exprime une propriété à maintenir. Comme le montre la Figure 5.7, cela se traduit en Event-B par : (i) un nouvel invariant (une contrainte) *contrainteNFR13* exprimant cette propriété ; (ii) une nouvelle action *actNFR13* dans l'événement Event-B traduisant le but G_3 (**MergeData**) afin de respecter l'invariant *contrainteNFR13*. De plus, cet impact nécessite l'ajout d'une nouvelle variable *precision_threshold*, d'un nouvel invariant de typage *invNFR13* et d'une initialisation de la variable ajoutée *actiniNFR13*.

La Figure 5.7 montre que tous les théorèmes déjà prouvés ne sont pas remis en cause. Il est cependant nécessaire de prouver un théorème supplémentaire *MilestoneTheo4NFR* afin de préserver le raffinement MILESTONE, comme expliqué dans la section 5.2.1 de ce chapitre.

MACHINE Localization1 // traduit le premier niveau de raffinement du modèle de buts
REFINES Localization //(CaptureRawLocalizations ; ValidateData ; MergeData) *Refines* LocalizeVehicle
SEES TypeSets
VARIABLES

... // Les variables déjà déclarées
`subcomponents_loc`, `validated_loc`, `merged_loc`
`raw_precision` // Variable créée par NFR11
`precision_threshold` // Variable créée par NFR13

INVARIANTS

`inv1 :` $subcomponents\_loc \in SUBCOMPONENTS \rightarrow (LATITUDE \times LONGITUDE)$
`inv2 :` $validated\_loc \in SUBCOMPONENTS \nrightarrow (LATITUDE \times LONGITUDE)$
`inv3 :` $merged\_loc \in LATITUDE \times LONGITUDE$
`inv4 :` $estimated\_loc = merged\_loc$
`invNFR11 :` $raw\_precision \in SUBCOMPONENTS \rightarrow \mathbb{N}$
`invNFR13 :` $precision\_threshold \in \mathbb{N}_1$
`contrainteNFR13 :` $(merged\_loc \in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})) \Rightarrow$
$(precision >= precision\_threshold)$ // Invariant pour NFR13
`MilestoneTheo1 :` $TargetG1 \Rightarrow CurrentG2$ // OP inchangée.
`MilestoneTheo2 :` $TargetG2 \Rightarrow CurrentG3$ // OP inchangée.
`MilestoneTheo3 :` $CurrentG1 \Rightarrow CurrentG$ // OP inchangée.
`MilestoneTheo4 :` $TargetG3 \Rightarrow TargetG$ // OP inchangée.
`MilestoneTheo4NFR :` $actNFR13 \Rightarrow actNFR1$ // OP ajoutée.

EVENTS
Initialisation

begin

... // Les initialisations déjà faites
`act2 :` $subcomponents\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$
`act3 :` $validated\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$
`act4 :` $merged\_loc := null \mapsto null$
`actiniNFR11 :` $raw\_precision :\in SUBCOMPONENTS \rightarrow \{0\}$
`actiniNFR13 :` $precision\_threshold := 1$

end

Event *CaptureRawLocalizations* $\widehat{=}$ // Evénement impacté par NFR11
when
`CurrentG1 :` $True$
then
`TargetG1 :` $subcomponents\_loc :\in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
`actNFR11 :` $raw\_precision :\in SUBCOMPONENTS \rightarrow \mathbb{N}$
end
Event *ValidateData* $\widehat{=}$ // traduit le but fonctionnel G2
when
`CurrentG2 :` $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
then
`TargetG2 :` $validated\_loc : |(validated\_loc' \in \mathscr{P}(subcomponents\_loc)) \wedge (ran(validated\_loc') \subseteq ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})))$
end
Event *MergeData* $\widehat{=}$ // Evénement impacté par NFR13
when
`CurrentG3 :` $(validated\_loc \in \mathscr{P}(subcomponents\_loc)) \wedge (ran(validated\_loc) \subseteq ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})))$
then
`TargetG3 :` $merged\_loc :\in (LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})$
`actNFR13 :` $precision : |precision' >= precision\_threshold$
end

FIGURE 5.7: La prise en compte des buts non-fonctionnels dans la première machine de raffinement Event-B du composant de localisation

5.4.3 Retour sur la seconde machine de raffinement

Une fois les localisations relatives faites, le but fonctionnel $G_{2.2}$ est impacté par le but de contribution $NFR12$ par la nécessité de présence de données. En d'autres termes, il est nécessaire tout d'abord de calculer un seuil de précision ($NFR12$) pour pouvoir ensuite assurer le filtrage des données ($G_{2.2}$). Comme le montre la Figure 5.8, l'impact de ce but de contribution va se traduire par un nouvel événement Event-B **CalculatePrecisionThreshold** (ayant comme action *actNFR12* qui assure le calcul du seuil de précision) qui précède l'événement impacté **FilterData**, ce qui perturbe l'ordonnancement des événements concrets. En fait, la séquence d'événements concrets raffinant l'événement abstrait **ValidateData** est modifiée, comme le montre le second commentaire associé à la clause REFINES dans la Figure 5.8. Afin de préserver les obligations de preuve déjà faites, nous utilisons la technique de *la redondance de garde* présentée dans la section 5.3.1 qui consiste à : (i) affecter la garde *CurrentG22* de l'événement impacté **FilterData** à la garde de l'événement ajouté **CalculatePrecisionThreshold** ; (ii) ajouter une nouvelle garde *grdNFR12* à l'événement **FilterData** pour s'assurer que cet événement ne peut se déclencher qu'après l'exécution de l'événement **CalculatePrecisionThreshold** ; (iii) prouver une contrainte d'ordonnancement supplémentaire *MilestoneTheo4NFR*. Notons que cet impact ne nécessite pas l'ajout de nouvelles

variables (ainsi que l'invariant de typage et l'initialisation correspondants) puisque la variable utilisée *precision_threshold* a été déjà introduite dans le raffinement précédent.

D'un autre côté, ce second niveau raffinement doit prendre en compte l'impact du but de contribution $NFR111$ (respectivement $NFR112$) sur le but fonctionnel $G_{1.1}$ (respectivement $G_{1.2}$) qui consiste en une donnée à satisfaire. Pour cela, l'événement Event-B correspondant au but $G_{1.1}$ **UseGPS** (respectivement au but $G_{1.2}$ **UseWIFI**) ajoute une nouvelle action *actNFR111* (respectivement *actNFR112*) afin d'assurer la satisfaction de cette donnée. Une nouvelle variable *gps_precision* (respectivement *wifi_precision*), un nouvel invariant de typage *invNFR111* (respectivement *invNFR112*) et une initialisation de cette variable *actiniNFR111* (respectivement *actiniNFR112*) sont également ajoutés. Notons qu'un invariant de collage *gluingInvNFR11* liant les deux variables concrètes (*wifi_precision*, *gps_precision*) à la variable abstraite *raw_precision* est également défini. En terme d'obligations de preuve, l'application du raisonnement expliqué dans la section 5.2.2 montre que le raffinement AND nécessite d'assurer une simulation d'action supplémentaire *AndTheo3NFR* suite à l'ajout des actions correspondants aux buts de contributions $NFR11$, $NFR111$ et $NFR112$.

MACHINE Localization2 // traduit le second niveau de raffinement du modèle de buts
REFINES Localization1 //(UseGPS ||| UseWIFI) *Refines* CaptureRawLocalizations
//(CaptureRelativeLocalizations ; CalculatePrecisionThreshold ; FilterData) *Refines* ValidateData
SEES TypeSets
VARIABLES

... // Les variables déjà déclarées
`gps_loc, wifi_loc, sensors_loc, kept_loc`
`gps_precision` // Variable créée par NFR111
`wifi_precision` // Variable créée par NFR112

INVARIANTS

`inv1 :` $gps\_loc \in \{gps\} \rightarrow (LATITUDE \times LONGITUDE)$
`inv2 :` $wifi\_loc \in \{wifi\} \rightarrow (LATITUDE \times LONGITUDE)$
`inv3 :` $subcomponents\_loc = gps\_loc \cup wifi\_loc$
`inv4 :` $sensors\_loc \in SUBSENSORS \nrightarrow (LATITUDE \times LONGITUDE)$
`inv5 :` $kept\_loc \in SUBCOMPONENTS \nrightarrow (LATITUDE \times LONGITUDE)$
`inv6 :` $validated\_loc = kept\_loc$
`invNFR111 :` $gps\_precision \in \{gps\} \rightarrow \mathbb{N}$
`invNFR112 :` $wifi\_precision \in \{wifi\} \rightarrow \mathbb{N}$
`gluingInvNFR11 :` $raw\_precision = gps\_precision \cup wifi\_precision$
`AndTheo1 :` $CurrentG11 \Rightarrow CurrentG1$ // OP inchangée.
`AndTheo2 :` $CurrentG12 \Rightarrow CurrentG1$ // OP inchangée.
`AndTheo3 :` $TargetG11 \wedge TargetG12 \Rightarrow TargetG1$ // OP inchangée.
`AndTheo3NFR :` $actNFR111 \wedge actNFR112 \Rightarrow actNFR11$ // OP ajoutée.
`MilestoneTheo1 :` $TargetG21 \Rightarrow CurrentG22$ // OP inchangé.
`MilestoneTheo2 :` $CurrentG21 \Rightarrow CurrentG2$ // OP inchangée.
`MilestoneTheo3 :` $TargetG22 \Rightarrow TargetG2$ // OP inchangée.
`MilestoneTheo4NFR :` $actNFR12 \Rightarrow grdNFR12$ // OP ajoutée.

EVENTS
Initialisation

begin

... // Les initialisations déjà faites
`act5 :` $gps\_loc :\in \{gps\} \rightarrow (\{null\} \times \{null\})$
`act6 :` $wifi\_loc :\in \{wifi\} \rightarrow (\{null\} \times \{null\})$

act7 : $sensors\_loc :\in SUBSENSORS \rightarrow (\{null\} \times \{null\})$
act8 : $kept\_loc :\in SUBCOMPONENTS \rightarrow (\{null\} \times \{null\})$
actiniNFR111 : $gps\_precision :\in \{gps\} \rightarrow \{0\}$
actiniNFR112 : $wifi\_precision :\in \{wifi\} \rightarrow \{0\}$

end

Event *UseGPS* $\widehat{=}$ // Evénement impacté par NFR111

when

CurrentG11 : $True$

then

TargetG11 : $gps\_loc :\in \{gps\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
actNFR111 : $gps\_precision :\in \{gps\} \rightarrow \mathbb{N}_1$

end

Event *UseWIFI* $\widehat{=}$ // Evénement impacté par NFR112

when

CurrentG12 : $True$

then

TargetG12 : $wifi\_loc :\in \{wifi\} \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$
actNFR112 : $wifi\_precision :\in \{wifi\} \rightarrow \mathbb{N}_1$

end

Event *CaptureRelativeLocalizations* $\widehat{=}$ // traduit le but fonctionnel G21

when

CurrentG21 : $subcomponents\_loc \in SUBCOMPONENTS \rightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))$

then

TargetG21 : $sensors\_loc : |(sensors\_loc' \in SUBSENSORS \nrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\}))) \wedge sensors\_loc' \neq \varnothing$

end

Event *CalculatePrecisionThreshold* $\widehat{=}$ // Evénement ajouté traduisant NFR12

when

CurrentG22 : $sensors\_loc \in SUBSENSORS \nrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})) \wedge sensors\_loc \neq \varnothing$

then

actNFR12 : $precision\_threshold :\in \mathbb{N}_1$

end

Event *FilterData* $\widehat{=}$ // Evénement impacté par NFR12

when

CurrentG22 : $sensors\_loc \in SUBSENSORS \nrightarrow ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})) \wedge sensors\_loc \neq \varnothing$
grdNFR12 : $precision\_threshold \in \mathbb{N}_1$

then

TargetG22 : $kept\_loc : |(kept\_loc' \in \mathscr{P}(subcomponents\_loc)) \wedge (ran(kept\_loc') \subseteq ((LATITUDE \setminus \{null\}) \times (LONGITUDE \setminus \{null\})))$

end

Event *MergeData* $\widehat{=}$ // Le même que celui dans la machine précédente
refines *MergeData*

END

FIGURE 5.8: La prise en compte des buts non-fonctionnels dans la seconde machine de raffinement Event-B du composant de localisation

5.4.4 Retour sur la troisième machine de raffinement

La troisième machine de raffinement Event-B *Localization3* (voir la Figure 4.13) n'est pas modifiée pour deux raisons. D'une part, le troisième niveau de raffinement dans la hiérarchie de buts fonctionnels SysML/KAOS n'est pas impacté par aucun but non-fonctionnel (comme le montre la Figure 5.1). D'autre part, il n'y a pas d'obligations de preuve supplémentaires à prouver, et par conséquent il n'y a pas d'enrichissement à faire dans les gardes et les post-conditions des événements de cette machine (**UseSpeedSensor** et **UseAccelerometer**). Ce second point s'explique par le fait que la garde et la post-condition de l'événement abstrait **CaptureRelativeLocalizations** ne sont pas modifiées dans la machine de raffinement précédente, comme le montre la Figure 5.8.

5.4.5 Bilan des preuves

Le tableau 5.1 résume le nombre d'obligations de preuve (OP) générées par la plate-forme RODIN pour le modèle Event-B du composant de localisation. Au total, 90 obligations de preuve sont générées (les théorèmes ajoutés, la faisabilité des événements, la préservation d'invariants). 28 d'entres elles sont générées suite à la prise en compte des NFRs, comme le montre le tableau 5.1. Les obligations de preuve sont quasiment toutes déchargées automatiquement par le prouveur de la plate-forme RODIN sauf quelques unes qui ont nécessité l'intervention du concepteur dans une preuve interactive. La preuve automatique concerne essentiellement les OP liées aux théorèmes ajoutés et aux invariants de typage alors que la preuve interactive est nécessaire pour prouver surtout certaines OP liées à la faisabilité des événements.

Machine Event-B	**Nombre d'OP avant la prise en compte des NFRs**	**Nombre d'OP liées à la prise en compte des NFRs**	**Total**
Localization	3	3	6
Localization1	17	10	27
Localization2	27	15	42
Localization3	15	0	15
Total	62	28	90

TABLE 5.1: Bilan du nombre d'obligations de preuve pour le modèle Event-B du composant de localisation.

5.5 Conclusion

Dans ce chapitre, nous avons proposé une première solution pour prendre en compte les buts non-fonctionnels et leurs impacts sur les buts fonctionnels afin de compléter et enrichir les modèles abstraits Event-B, obtenus à partir des buts fonctionnels. Notre proposition consiste à étiqueter les éléments de la spécification Event-B qui sont liés aux buts non-fonctionnels. Ces étiquettes permettent de localiser dans la spécification formelle les impacts des buts non-fonctionnels sur les buts fonctionnels. Les différents liens de correspondance ainsi définis peuvent faciliter la gestion de l'évolution des exigences non-fonctionnelles. Ces étiquettes permettent également de retrouver les éléments non-fonctionnels concernés par une obligation de preuve non déchargée. D'un autre côté, nous avons étudié l'effet de la prise en compte des buts non-fonctionnels dans les machines Event-B en ce qui concerne les obligations de preuve déjà faites. Cette étude a identifié jusqu'à maintenant un certain nombre de règles, de recommandations et de techniques permettant de préserver les obligations de preuve déjà faites sans avoir à les refaire. Néanmoins, la preuve des théorèmes supplémentaires peut être nécessaire.

La suite du travail consiste à améliorer et compléter les résultats actuels. L'idée est de se servir tout d'abord des catalogues définis dans le NFR framework [Chung *et al.* 2000] afin d'identifier les divers types de buts non-fonctionnels SysML/KAOS et de faciliter également la tâche de la découverte des buts de contribution. Nous comptons par la suite étendre ces catalogues pour stocker les règles de la prise en compte des buts de contribution en Event-B. Cela va permettre d'une part d'étudier divers types de buts non-fonctionnels (sécurité, sûreté, performance, etc.), et d'autre part de semi-automatiser la prise en compte des buts non-fonctionnels en Event-B. Nous comptons également étudier la possibilité [1] de la prise en compte en Event-B des conflits qui peuvent intervenir entre des buts non-fonctionnels SysML/KAOS. De même, les impacts négatifs des buts de contribution sur les buts fonctionnels seront étudiés dans des futurs travaux.

1. Le terme *possibilité* est employé car à priori, ces conflits doivent être préalablement résolus au niveau analyse (au niveau SysML/KAOS). Par conséquent, le modèle de buts non-fonctionnels que nous utilisons pour appliquer la transformation en Event-B sera un modèle de buts non-fonctionnels sans conflits.

Troisième partie

Implémentation

CHAPITRE 6

SysKAOS2EventB : Un outil de construction de spécifications abstraites Event-B

Sommaire

Dans ce chapitre, nous présentons la mise en œuvre, appelée SysKAOS2EventB, implémentant l'approche dédiée à Event-B proposée dans le chapitre 4. Notre proposition s'inscrit dans le cadre de l'ingénierie dirigée par les modèles et se base essentiellement sur les technologies de transformation de modèles. Le travail présenté dans ce chapitre a fait récemment l'objet de la publication [Matoussi & Laleau 2011a].

6.1 Préliminaires

6.1.1 L'ingénierie dirigée par les modèles

L'Ingénierie Dirigée par les Modèles (IDM) est une discipline récente du génie logiciel qui met les modèles au premier plan au sein du processus du développement logiciel [Bézivin 2004]. L'IDM repose ainsi sur l'utilisation de modèles à différents niveaux d'abstraction. Plusieurs approches orientées modèles partagent

les concepts de l'IDM dont par exemple MDA [1], acronyme de Model Driven Architecture, qui est proposé par l'OMG [2]. Dans un processus de développement orienté MDA, tout est considéré comme modèle. En partant de la phase de l'analyse des exigences jusqu'à la phase d'implémentation, les modèles sont utilisés pour représenter les artefacts logiciels manipulés durant le processus de construction de l'application. Le MDA classe les modèles en modèles indépendants des plates-formes appelés PIM (Platform-Independent Models) et en modèles spécifiques appelés PSM (Platform-Specific Models) [Blanc 2005]. L'approche MDA permet de déployer un même modèle de type PIM sur plusieurs plates-formes (modèles PSM). L'approche MDA repose sur plusieurs standards de l'OMG tels que :

- MOF (Meta Object Facility) : c'est un formalisme de modélisation de méta-modèles, appelé méta-méta-modèle.
- XMI (XML Metadata Interchange) : ce standard permet de représenter les modèles sous forme de documents XML pour des besoins d'interopérabilité.
- MOF 2.0 QVT (Query, View, Transformation) : ce standard apporte des solutions pour la problématique de transformation de modèles dans MDA en définissant un méta-modèle permettant l'élaboration de modèles de transformation.

6.1.2 La transformation de modèles

La transformation de modèles est une opération fondamentale dans toute approche orientée modèles. Elle permet de générer un ou plusieurs modèles cibles à partir d'un ou plusieurs modèles sources [Bézivin 2004]. L'OMG a lancé en 2002 un appel à proposition industriel intitulé RFP MOF/QVT [3] (*Request For Proposal : MOF 2.0 Query/Views/Transformations*) dont le but était de définir une sorte de langage unifié de transformation ou plutôt une famille coordonnée de tels langages. L'idée est de considérer la transformation comme une application définie par un modèle de transformation (Mt), c'est-à-dire modéliser la transformation elle-même en lui appliquant les principes de l'IDM. Comme le montre la Figure 6.1, le modèle Mt doit être lui-même conforme à un méta-modèle (MMt), une définition abstraite du langage de transformation [Bézivin 2004]. Cette approche de définition de transformation est dite par *méta-modélisation*, par opposition aux autres approches telles que l'approche par *template* utilisée généralement pour les transformations « modèle-vers-texte » (génération de code ou de documentation) ou l'approche par *programmation* [Blanc 2005]. Une transformation de modèles définie par un modèle Mt est donc l'opération qui permet de transformer un modèle Ma (conforme à son méta-modèle MMa) en un modèle Mb (conforme à son méta-modèle MMb), comme le montre la Figure 6.1. En fonction de la nature des

1. http://www.omg.org/mda/
2. L'OMG (Object Management Group) est un consortium regroupant des industriels et des chercheurs dont l'objectif principal est l'établissement des standards pour résoudre par exemple les problèmes d'interopérabilité entre les systèmes d'information.
3. http://www.omg.org/cgi-bin/doc?ad/2002-04-10

méta-modèles source et cible impliqués, nous distinguons les transformations dites *endogènes* dont les modèles source et cible sont conformes au même méta-modèle, des transformations dites *exogènes* dont les méta-modèles source et cible sont différents [Diaw *et al.* 2010].

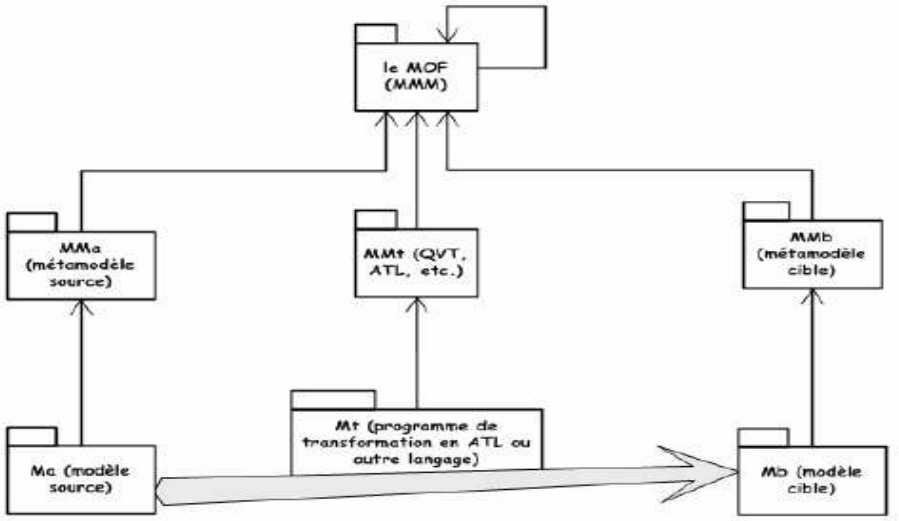


FIGURE 6.1: La transformation de modèles basée sur les méta-modèles [Bézivin 2004]

Plusieurs langages de transformation ont été proposés pour réaliser des transformations de modèles selon l'approche par *méta-modélisation*, parmi lesquels le standard MOF2.0 QVT[4], le langage Kermeta [Muller *et al.* 2005] et le langage ATL [Bézivin *et al.* 2003, Bézivin *et al.* 2004]. Dans cette thèse, notre choix s'est porté sur ce dernier langage. En effet, ATL est largement utilisé maintenant par la communauté parce qu'il est créé pour s'inscrire dans une approche MDA et il est aussi considéré comme un standard de transformation dans Eclipse.

6.1.3 ATL

ATL [Bézivin *et al.* 2003], acronyme de « ATLAS Transformation Language », est un langage de transformation de modèles « hybride », déclaratif et impératif. Un modèle de transformation ATL se base sur des définitions de méta-modèles au format XMI. ATL est défini par un modèle MOF pour sa syntaxe abstraite et possède également une syntaxe concrète textuelle. ATL utilise des requêtes sous forme d'expressions OCL afin d'accéder aux éléments d'un modèle. Notons qu'une requête permet de naviguer entre les éléments d'un modèle et d'appeler des opérations sur ceux-ci. Comme le montre la Figure 6.2, une règle déclarative d'ATL (appelée *matched rule*) est spécifiée par un nom, un ensemble de patrons sources *InPattern* (matchés avec les éléments source) et un ensemble de patrons cibles *OutPattern* (représentent les éléments à créer dans le modèle cible). D'un autre côté, le

4. http://www.omg.org/spec/QVT/1.0/

style impératif d'ATL est supporté par deux constructions différentes : (i) les règles impératives appelées *called rule* ; (ii) un bloc d'instructions impératives (*Action-Block*) utilisé avec les deux types de règles. Notons qu'une *called rule* est appelée explicitement en utilisant son nom et en initialisant ses paramètres.

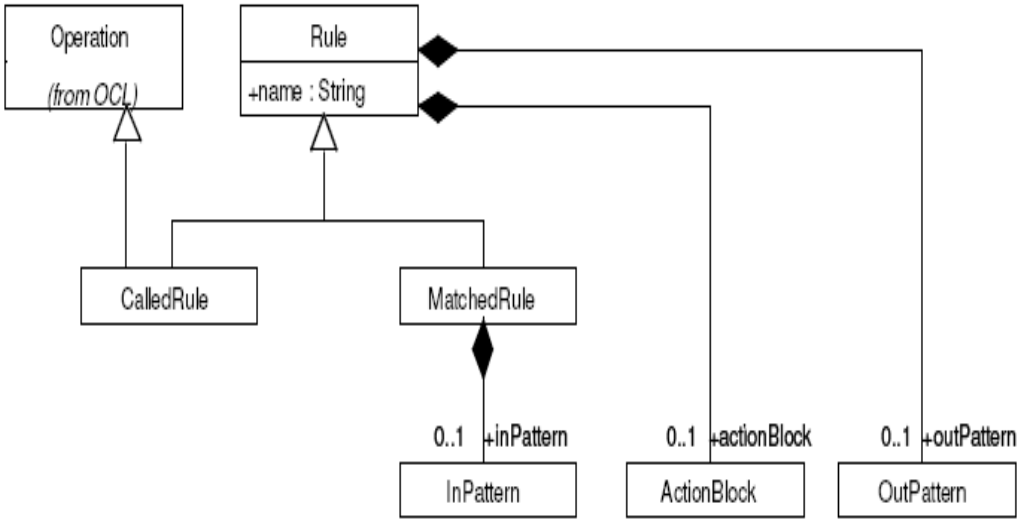


FIGURE 6.2: Syntaxe abstraite d'une règle de transformation ATL [Bézivin *et al.* 2003]

ATL, qui fait partie depuis 2007 du projet Eclipse M2M [5], est outillé sous forme de plug-in ADT (*ATL Development Tool*) pour l'environnement de développement Eclipse.

6.2 Un aperçu du plug-in SysKAOS2EventB

6.2.1 Le processus de développement

Le plug-in SysKAOS2EventB est une mise en œuvre de l'approche proposée dans le chapitre 4. Pour concevoir des modèles de buts SysML/KAOS, un autre plug-in, nommé SysML/KAOS [Gnaho & Semmak 2011], a été élaboré sur la plateforme TOPCASED. Notre plug-in SysKAOS2EventB est développé aussi sous TOPCASED en se basant sur les technologies de transformation de « modèle-vers-modèle » , dans le but d'assurer la liaison avec la plateforme RODIN. TOPCASED et RODIN sont toutes les deux basées sur la plateforme Eclipse profitant ainsi des technologies de gestion de modèles offertes par Eclipse dont par exemple le framework EMF (Eclipse Modeling Framework) [Steinberg *et al.* 2009]. L'intérêt majeur du framework EMF est qu'il permet de décrire un méta-modèle conforme à Ecore, l'implémentation Eclipse de la spécification MOF, et de générer par la suite le code de manipulation et d'édition des modèles conforme à ce méta-modèle. Afin que la technologie EMF soit exploitable dans les outils RODIN, [Snook *et al.* 2010] ont

5. http://www.eclipse.org/m2m/

défini un front-end EMF pour Event-B que nous utilisons dans notre outil SysKAOS2EventB.

Comme le montre la Figure 6.3, l'utilisateur commence par éditer un modèle de buts fonctionnels (le diagramme .sys) sous TOPCASED en utilisant le plug-in SysML/KAOS. Le plug-in SysKAOS2EventB permet par la suite de générer une architecture de raffinement Event-B. En effet, le plug-in SysKAOS2EventB crée un projet RODIN (avec plusieurs machines Event-B) en appliquant un processus de transformation basée sur le langage ATL. La Figure 6.3 montre les deux principales étapes du processus de développement du plug-in SysKAOS2EventB :

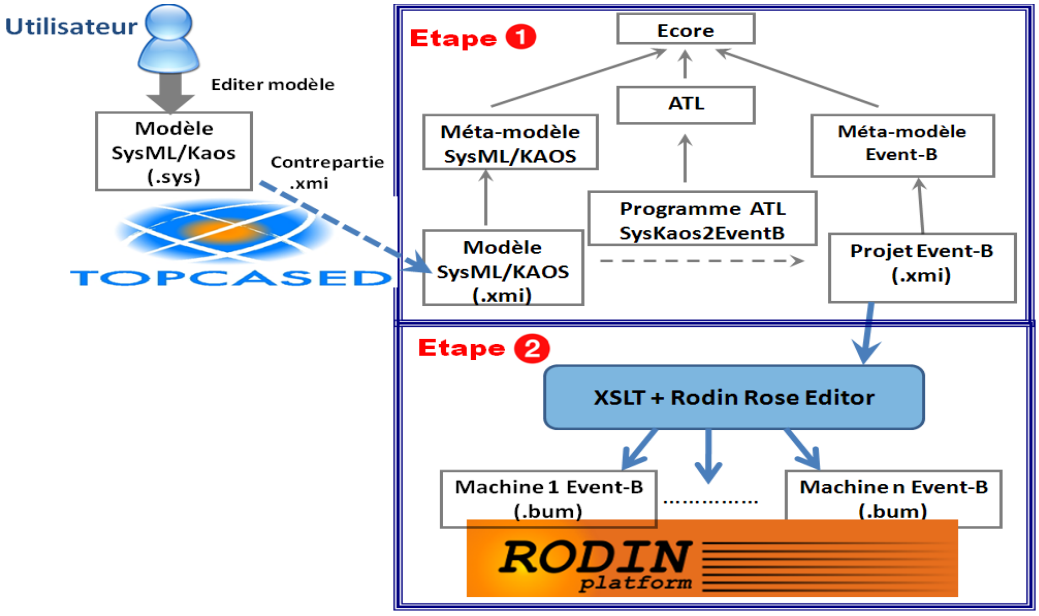


FIGURE 6.3: Un aperçu du plug-in SysKAOS2EventB

1. L'objectif de la première étape est de transformer le diagramme de buts SysML/KAOS (plus précisément la contrepartie .xmi du modèle de buts .sys) vers un fichier cible .xmi représentant toutes les machines Event-B. Pour cela, ATL est choisi comme langage de transformation de modèles. Rappelons qu'un programme de transformation ATL référence un méta-modèle source et un méta-modèle cible afin de définir comment des éléments source sont transformés vers des éléments cibles en utilisant un ensemble de règles déclaratives et impératives. Les méta-modèles source et cible sont décrits en Ecore et doivent être ainsi conformes au méta-méta-modèle Ecore défini dans le cadre EMF d'Eclipse. Pour notre plug-in, nous utilisons deux méta-modèles : un pour SysML/KAOS et le second pour Event-B. En fait, notre programme *SysKAOS2eventB.atl* prend en entrée un modèle source conforme au méta-modèle de SysML/KAOS et produit un modèle cible conforme au méta-modèle Event-B. De même, le programme de transformation ATL *SysKAOS2EventB.atl* est un modèle qui doit être conforme au méta-modèle ATL qui doit être à son tour conforme au méta-méta-modèle Ecore.

2. Le fichier .xmi obtenu après l'application des règles ATL et qui contient les différents machines Event-B ne peut pas être directement chargé par RODIN. Pour cela, la deuxième étape consiste à utiliser XSLT [6] afin d'obtenir des fichiers .xmi, chacun correspondant à une machine Event-B. Chaque fichier .xmi obtenu est ensuite chargé dans RODIN grâce au plug-in Rose Editor [7] qui permet d'obtenir une extension du fichier .xmi compréhensible par RODIN (l'extension .bum).

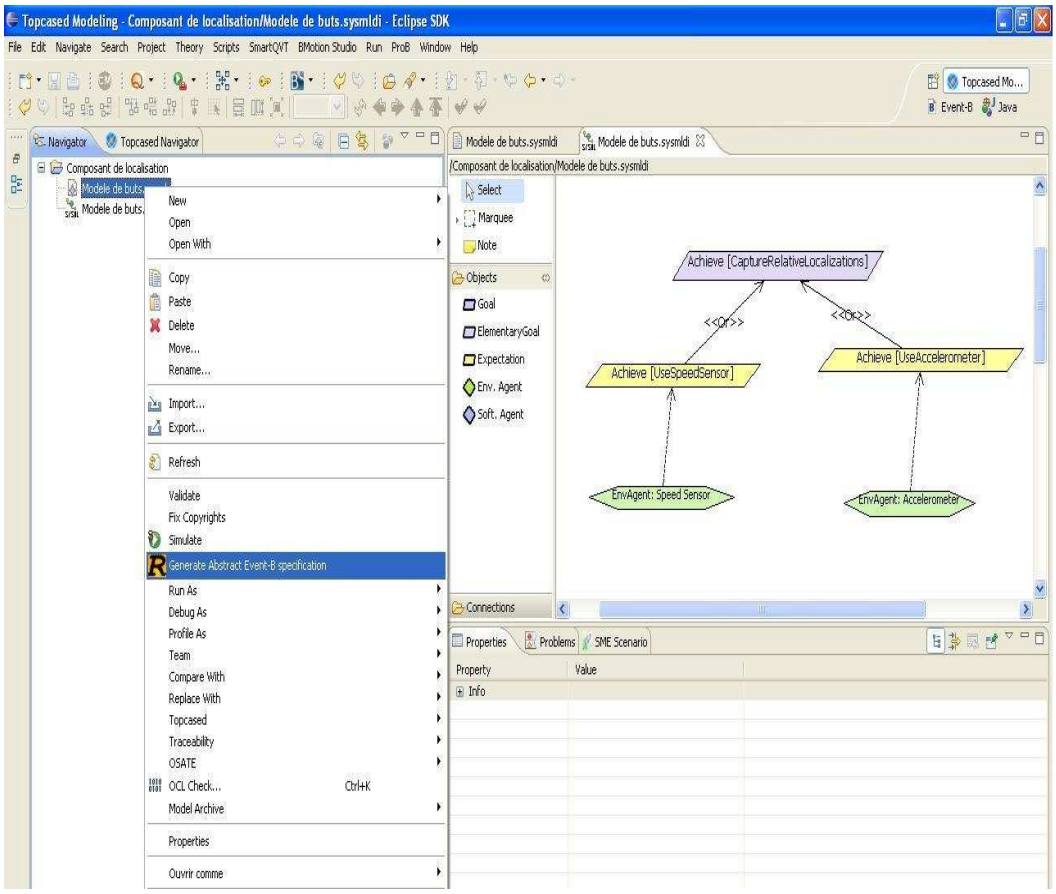


FIGURE 6.4: Fonctionnalité offerte par le plug-in SysKAOS2EventB à la vue de gestion de projets de TOPCASED

6.2.2 Un scénario d'utilisation

L'utilisation du plug-in est appliquée sur le cas d'étude que nous avons présenté dans la section 4.6. Comme le montre la Figure 6.4, l'utilisateur commence par concevoir le modèle de buts fonctionnels SysML/KAOS, en utilisant le plug-in SysML/KAOS sous TOPCASED. L'éditeur de propriétés permet de définir les

6. http://www.w3.org/TR/xslt20/
7. http://wiki.event-b.org/index.php/Rose_(Structured)_Editor

attributs (nom, identifiant, *CurrentCondition* et *TargetCondition*) liés à chaque but SysML/KAOS. Notons que les *CurrentCondition* et les *TargetCondition* acceptent pour le moment des valeurs écrites seulement selon la notation Event-B afin de faciliter la transformation ATL. Le plug-in SysKAOS2EventB est par la suite disponible (via la fonction *Generate Abstract Event-B specification*) par un clic droit de la souris dans la vue de gestion de projets de TOPCASED en sélectionnant un fichier .sys, comme le montre la Figure 6.4.

Le plug-in SysKAOS2EventB génère une architecture de raffinement Event-B qui représente la contre-partie formelle du modèle de buts SysML/KAOS, comme le montre la Figure 6.5. Ce modèle Event-B est exploitable en ouvrant la perspective RODIN. Rappelons que jusqu'à présent, les autres éléments d'Event-B (l'initialisation, les variables et leurs invariants de typage, et les contextes) sont manuellement complétés par le concepteur. Il est possible maintenant de vérifier et valider le modèle Event-B (et par conséquent sa contrepartie semi-formelle SysML/-KAOS) en explorant l'ensemble des plug-in RODIN tels que le prouveur, le model-checker ou l'animateur. Dans le cas où le modèle Event-B n'est pas prouvé ou validé, il est possible de retourner au modèle de buts SysML/KAOS pour le corriger, en basculant vers la perspective TOPCASED.

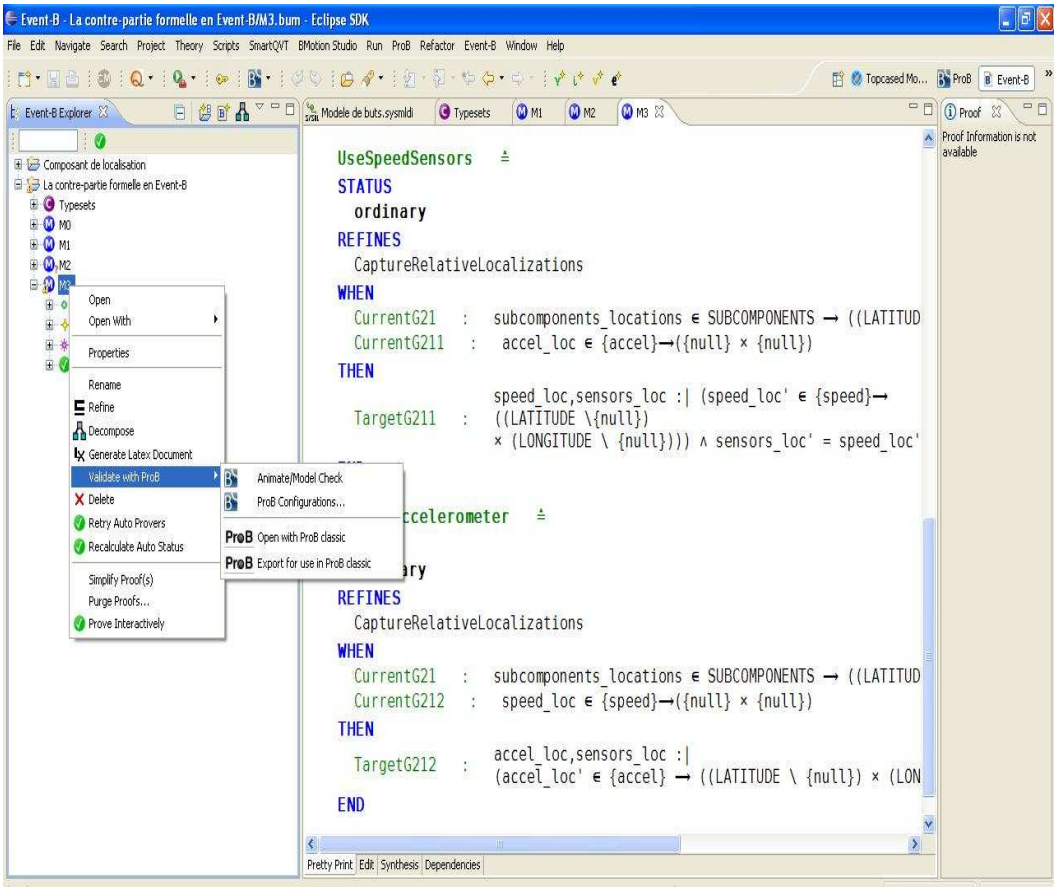


FIGURE 6.5: La vue offerte par les plug-in de la plateforme RODIN

6.3 Description de la transformation ATL

Parmi les différents langages de transformation de modèles, nous avons privilégier l'approche déclarative afin de ne pas se préoccuper des contraintes relatives à l'implantation, notamment l'ordre d'exécution des règles. Pour cela, notre choix s'est porté sur le langage ATL [Bézivin *et al.* 2003] d'autant plus qu'il partage plusieurs caractéristiques du standard QVT de l'OMG [Jouault & Kurtev 2006]. Le langage ATL permet de définir une transformation par le biais de règles déclaratives pouvant faire appel à des fonctions auxiliaires (helpers) qui peuvent être récursives. Rappelons que ces règles sont appliquées sur une instance d'un méta-modèle source et permet de générer une instance conforme à un autre méta-modèle cible. Comme la plupart des langages de transformation, ATL offre la possibilité de faire des transformations de modèles aussi bien endogènes qu'exogènes puisque les méta-modèles source et cible peuvent être différents.

Un programme ATL qui définit la transformation d'un modèle en un autre modèle est appelé *module* et est stocké dans un fichier (.atl). Le module est composé d'une section entête, d'une section d'importation de bibliothèques, d'une section de déclaration des fonctions auxiliaires (helpers) et d'une section de spécification des règles de transformation. Les règles de transformation en ATL peuvent être spécifiées dans un style déclaratif (matched/lazy rules) ou impératif (called rules). Dans ce qui suit, nous décrivons en détails les différentes sections ATL utilisées dans le cadre de notre programme de transformation ATL SysKAOS2EventB.

6.3.1 L'en-tête (header)

SysML est un profil UML bien qu'il n'en reprenne pas tous ses éléments. La partie d'UML qui est réutilisée par SysML est appelée UML4SysML. Comme notre méta-modèle source SysML/KAOS hérite du méta-modèle SysML qui lui-même hérite du méta-modèle UML, nous avons exploité le principe de "multiple pattern matching" offert par ATL et qui permet d'accepter plusieurs méta-modèles sources en entrée. Ainsi, la Figure 6.6 montre que l'en-tête de notre module ATL SysKAOS2EventB contient : (i) la partie **from** qui indique les trois modèles sources utilisés ; (ii) la partie **create** qui dénote le modèle de sortie à générer qui doit être conforme au méta-modèle Event-B.

```
module SysKAOS2EventB;
create OUT : eventb from IN : kaos, IN1 : sysml, IN2 : uml;
```

FIGURE 6.6: L'en-tête du module ATL SysKAOS2EventB

6.3.2 Les fonctions auxiliaires (helpers)

Les fonctions auxiliaires (helpers) permettent de factoriser des parties de code souvent utilisées afin d'éviter leur redondance. Les helpers servent à calculer une

information utilisable par des règles de transformation ou par d'autres helpers en spécifiant des opérations de navigation sur les modèles source. Les helpers ATL peuvent avoir des paramètres et ils utilisent des expressions OCL pour calculer la valeur de retour. Dans ce qui suit, nous présentons les principaux helpers du module ATL SysKAOS2EventB.

6.3.2.1 Quelques helpers de base

```
helper context uml!PackageableElement def :AllGoals : Sequence(uml!packagedElement
    ) =
    self.packagedElement->reject(t|t.name.startsWith('And') or t.name.startsWith('
        Or') or t.name.startsWith('Milestone'))->collect(r|r) ;

helper context uml!PackageableElement def :AllRelationships : Sequence(uml!
    packagedElement) =
   self.packagedElement->select(t|t.name.startsWith('And') or t.name.startsWith('
       Or') or t.name.startsWith('Milestone'))->collect(r|r);


helper context uml!PackageableElement def :AllRelationshipsSuppliers : Sequence(
    uml!packagedElement) =
   self.packagedElement->select(t|t.name.startsWith('And') or t.name.startsWith('
       Or') or t.name.startsWith('Milestone'))->collect(r|r.supplier);


helper context uml!PackageableElement def :GetRelationshipFromClient(goal:Sequence
    (uml!PackageableElement)) : Sequence(uml!PackageableElement) =
  self.AllRelationships->select(t|t.client=goal.first())->collect(r|r.name);


helper context uml!PackageableElement def :GetClient(goal:Sequence(uml!
    PackageableElement)) : Sequence(uml!PackageableElement) =
   self.AllRelationships->select(t|t.supplier.first()=goal->flatten().first())->
       collect(r|r.client);


helper context uml!PackageableElement def :GetParentsOfGoals(goal:Sequence(uml!
    PackageableElement)) : Set(uml!PackageableElement) =
   goal->iterate(rt; p: Set(OclAny)=Set{}|
                p.union(
                          Set{self.GetClient(rt)}));

helper context uml!PackageableElement def :GetChildrensofOneGoal(goal:Sequence(uml
    !PackageableElement)) : Sequence(uml!PackageableElement) =
   self.AllRelationships->select(t|t.client=goal->flatten().asSequence())->collect
       (r|r.supplier);
```

FIGURE 6.7: Quelques helpers de base du module ATL SysKAOS2EventB

La Figure 6.7 présente quelques helpers de base permettant de récupérer certaines informations d'un modèle de buts SysML/KAOS. Par exemple, le helper **AllGoals** permet de collecter tous les buts figurant dans un modèle de buts tandis que le helper **AllRelationships** permet de collecter tous les patrons de raffinement. Ce dernier helper est appelé par d'autres helpers dont par exemple le helper **GetRelationshipFromClient** qui permet de trouver la relation de raffinement relative

à un but « client » (un but-père) donné. Le helper **AllRelationshipsSuppliers** permet quant à lui de collecter tous les buts « supplier » (les buts-fils) des relations de raffinement. Le helper **GetParentsOfGoals** permet de collecter la liste de buts-pères à partir d'un ensemble de buts passés comme paramètre. Pour cela, ce helper fait un traitement itératif sur les buts en appelant le helper **GetClient** qui permet de trouver le but-père correspondant à un but donné. Le helper **GetChildrensofOneGoal** permet de trouver tous les sous-buts liés à un but donné. Pour cela, le helper ajoute les « supplier » de toutes les relations de raffinement dont le « client » est égal au but passé comme paramètre.

6.3.2.2 Le helper trouvant le but le plus abstrait

D'une manière analogue aux précédents helpers, la Figure 6.8 présente le helper **AbsGoal** qui consiste à faire un traitement itératif sur tout le modèle de buts afin de trouver le but le plus abstrait. Ce dernier est le but qui ne figure pas dans la liste des buts-fils des relations de raffinement. Notons que la liste de tous les buts d'un modèle de buts et celle incluant tous les fils des relations de raffinement sont récupérées grâce à l'appel des helpers **AllGoals** et **AllRelationshipsSuppliers** (présentés dans la Figure 6.7), repectivement.

```
helper context uml!PackageableElement def :AbstGoal:Sequence(uml!
    PackageableElement)=
self.AllGoals->iterate(goal; w: Sequence(OclAny)=Sequence{}|
  w.union(
       if (self.AllRelationshipsSuppliers.includes(Sequence{goal}))then
           Sequence{}
       else
           Sequence{goal}
       endif));
```

FIGURE 6.8: Le helper trouvant le but le plus abstrait

6.3.2.3 Les helpers organisant la hiérarchie de buts

Le helper **GoalsOfOneLevel**, présenté dans la Figure 6.9, permet d'obtenir la séquence de tous les sous-buts liés à un ensemble de buts passés comme paramètre. Ce helper fait un traitement itératif sur ces buts en appelant le helper **GetChildrensofOneGoal** (présenté dans la Figure 6.7) qui ajoute à chaque fois les sous-buts trouvés dans la séquence résultat. Ce helper est important dans la mesure où il va nous permettre de trouver les buts liés à un niveau donné de la hiérarchie de raffinement. D'ailleurs, c'est dans cet esprit qu'il est utilisé par le helper **GoalsHierarchy** qui fait un traitement récursif afin d'organiser la hiérarchie de buts par niveau de raffinement. En effet, le helper **GoalsHierarchy** permet d'obtenir une séquence finale contenant des sous-séquences où chaque sous-séquence contient tous les buts d'un niveau donné de la hiérarchie de buts.

```
helper context uml!PackageableElement def : GoalsOfOneLevel(client:Sequence(uml!
    PackageableElement)):Sequence(uml!PackageableElement)=
    if (client.size()=1) then self.GetChildrensofOneGoal(client)
    else
            client->iterate(fil; w: Sequence(OclAny)=Sequence{}|
           w.union(
              self.GetChildrensofOneGoal(fil)))
       endif;


helper context uml!PackageableElement def :GoalsHierarchy(pere:Sequence(uml!
    PackageableElement),L:Sequence(uml!PackageableElement))
:Sequence(OclAny)=

       if (not( pere=Sequence{})) then

        let L1 :Sequence(uml!PackageableElement) = L.including((self.
            GoalsOfOneLevel(pere)))->reject(t|t=Sequence{}) in
         self.GoalsHierarchy((self.GoalsOfOneLevel(pere)),L1)

        else
              L
       endif;
```

FIGURE 6.9: Les helpers organisant la hiérarchie de buts

6.3.2.4 Les helpers définissant l'expression Event-B des patrons de raffinement

La Figure 6.10 présente le helper **AllRefinementPatterns** qui permet d'avoir l'expression Event-B des différents patrons de raffinement de buts, en les organisant par niveau de raffinement. C'est pourquoi ce helper fait appel au helper **GoalsHierarchy** (présenté dans la Figure 6.9) pour pouvoir récupérer toute la hiérarchie de buts organisée par niveau de raffinement, en éliminant bien sûr le niveau abstrait. Le helper **AllRefinementPatterns** fait par la suite appel au helper **PatternsEventBSemantics** qui définit l'expression Event-B des patrons de raffinement de buts. Pour cela, ce dernier helper prend en entrée tout d'abord un ensemble de buts. Puis, il récupère la liste des pères de ces différents buts grâce à l'appel du helper **GetParentsOfGoals**. Le helper fait ensuite un traitement itératif sur l'ensemble des buts-pères récupérés. Il récupère ainsi le patron de raffinement relatif au but-père en appelant le helper **GetRelationshipFromClient**. En fonction du type de raffinement (AND, OR, MILESTONE), le helper **PatternsEventBSemantics** définit enfin l'expression Event-B. Par exemple, la définition de l'expression Event-B d'un raffinement AND consiste à appeler le helper **SetStringAndList**, comme le montre la Figure 6.10.

```
helper context uml!PackageableElement def : AllRefinementPatterns:Sequence(OclAny)
    =

    (self.GoalsHierarchy(self.AbstGoal,Sequence{})->reject(y|y=self.AbstGoal))->
        iterate(o; j: Sequence(OclAny)=Sequence{}|
                j.union(
                      Sequence{self.PatternsEventBSemantics(o)}))
       ;

helper context uml!PackageableElement def :PatternsEventBSemantics(goal:Sequence(
    uml!PackageableElement)) : String=

        self.GetParentsOfGoals(goal)->iterate(k; p: String=''|
        p+(
                if self.GetRelationshipFromClient(k).first().startsWith('And')
                          then ('(' +self.SetStringAndList(k)+')'+' Refines '+ 'Ev
                              '+(k)->flatten().first().name+'    ')


                else if self.GetRelationshipFromClient(k).first().startsWith('Or')
                          then ('(' +self.SetStringOrList(k)+')'+' Refines '+ 'Ev
                              '+(k)->flatten().first().name+'    ')


                else  if self.GetRelationshipFromClient(k).first().startsWith('
                    Milestone') then( '(' +self.SetStringMilestoneList(k)+')'+'
                    Refines '+ 'Ev'+(k)->flatten().first().name+'    ')

                 else    ''
        endif
        endif
        endif))
;

helper context uml!PackageableElement def :SetStringAndList(k:Sequence(uml!
    PackageableElement)) : String =

        let Chaine : String = (self.GetChildrensofOneGoal(k)->flatten())->iterate(
            m; a: String='' |
               a+('Ev'+ m.name + ' ||| ')) in Chaine.substring(1,Chaine.size()-5)
;
```

FIGURE 6.10: Les helpers définissant l'expression Event-B des patrons de raffinement

6.3.3 Les règles de correspondance (matched rules)

Rappelons que les règles de correspondance (matched rule) en ATL sont des règles déclaratives qui sont appliquées automatiquement par le moteur de transformation ATL. Chaque règle est spécifiée par un nom, un ensemble de patrons sources mappés avec les éléments sources (dans la partie **from**), un ensemble de patrons cibles représentant les éléments créés dans le modèle cible (dans la partie **to**). Une règle de correspondance est appliquée une et une seule fois pour chaque élément source (*match*) retrouvé dans les modèles sources.

Il y a aussi d'autres types de règles de correspondance qui sont dites paresseuses (lazy rules). Une règle paresseuse est presque la même qu'une règle de correspondance standard sauf qu'elle doit être explicitement appelée pour être réalisée. Ainsi,

le moteur de transformation ATL n'applique jamais automatiquement une règle de correspondance paresseuse qui doit donc être déclenchée par d'autres règles. De plus, à la différence des règles de correspondance standard, les règles paresseuses peuvent êtres appliquées à plusieurs reprises pour un seul *match* en produisant à chaque fois un nouvel ensemble d'éléments cibles (sauf pour les « unique lazy rule ») .

```
rule Main_Transformation {

from  u:  uml!PackageableElement

using {
   L : Sequence(uml!PackageableElement) = Sequence{} ;
   racine : Sequence(uml!PackageableElement) = u.AbstGoal ;
  refinementsequence : Sequence(uml!PackageableElement) = u.GoalsHierarchy(racine,
      L)->reject(y|y=u.AbstGoal);
       }

to  abst : eventb!Machine (
      events <- u.AbstGoal->collect(r|thisModule.NewEvent(r))),

    refi : distinct eventb!Machine foreach(level in refinementsequence)(
      events<-u.AllEvents,
      comment<-u.AllRefinementPatterns,
      refinesNames <- 'M'+ (refinementsequence.indexOf(level)-1).toString(),
      invariants<-u. AllProofObligations)
}
```

FIGURE 6.11: La règle de correspondance principale du module ATL SysKAOS2EventB

Dans notre contexte, nous avons une seule règle de correspondance principale, nommée **Main_transformation**, comme le montre la Figure 6.11. Cette règle utilise des variables (dans la partie *using*) qui servent à déclarer par exemple : (i) la séquence *racine* qui contient le but le plus abstrait en faisant appel au helper **AbsGoal** (présenté dans la Figure 6.8) ; (ii) la séquence *refinementsequence* qui fait appel au helper **GoalsHierarchy** (présenté dans la Figure 6.9) afin d'obtenir tous les buts (sauf le but le plus abstrait) organisés par niveau de raffinement.

La règle **Main transformation** fait tout d'abord un traitement spécifique pour le premier niveau du modèle de buts SysML/KAOS (défini par le but le plus abstrait) afin de le transformer en une machine abstraite Event-B. Pour cela, la règle fait appel à la règle paresseuse **NewEvent** qui permet de créer un nouvel événement à partir d'un but. Comme le montre la Figure 6.12, cette règle paresseuse fait à son tour appel à d'autres règles paresseuses **NewGuard** et **NewAction** pour la construction de la garde et de l'action de l'événement, respectivement.

Pour pouvoir transformer les autres niveaux du modèle de buts SysML/KAOS dans des machines de raffinement Event-B, la règle de correspondance principale **Main_transformation** (voir la Figure 6.11) fait un traitement itératif sur la hiérarchie de buts SysML/KAOS (la séquence *refinementsequence*) en appelant un ensemble de helpers. Par exemple, le helper **AllEvents** (voir la Figure 6.13) permet d'avoir tous les événements Event-B d'un niveau donné de la hiérarchie de

```
lazy rule NewGuard {

  from b : kaos!Goal

  to g : eventb!Guard (
         name <- 'Current' + b.name,
         predicate <- b.CurrentCondition )
   }

 lazy rule NewAction {

   from b : kaos!Goal

   to  a : eventb!Action (
           name <- 'Target' + b.name,
           action <- b.TargetCondition )
 }

 lazy rule NewEvent {

   from b : kaos!Goal

   to e : eventb!Event (
          name <- 'Ev' + b.name,
          guards <-thisModule.NewGuard(b),
          actions<-thisModule.NewAction(b) )
}
```

FIGURE 6.12: Quelques règles paresseuses du module ATL SysKAOS2EventB

```
helper context uml!PackageableElement def : AllEvents:Sequence(OclAny) =

     (self.GoalsHierarchy(self.AbstGoal,Sequence{})->reject(y|y=self.AbstGoal))->
         iterate(o; j: Sequence(OclAny)=Sequence{}|
                 j.union(
                         Sequence{self.GoalsToEvents(o)}))
        ;


 helper context uml!PackageableElement def : GoalsToEvents(goal:Sequence(uml!
     PackageableElement)):Sequence(OclAny) =

        if  (not (Sequence{goal}.size()=1)) then

         (goal)->iterate(rt; p: Sequence(OclAny)=Sequence{}|
                 p.union(
                         Sequence{thisModule.NewEvent(rt->flatten()->first())}))
        else
                (goal->flatten())->iterate(rt; p: Sequence(OclAny)=Sequence{}|
                p.union(

                         Sequence{thisModule.NewEvent(rt)}))->flatten()
                endif

                 ;
```

FIGURE 6.13: Les helpers permettant d'avoir les événements Event-B d'un niveau donné de raffinement

buts. C'est pourquoi ce helper fait appel tout d'abord au helper **GoalsHierarchy** (présenté dans la Figure 6.9) pour pouvoir récupérer toute la hiérarchie de buts organisée par niveau de raffinement, en éliminant bien sûr le niveau abstrait. Le helper **AllEvents** fait par la suite un traitement itératif sur les niveaux de raffinement afin de créer les nouveaux événements à partir des buts. En fait, cette création des événements Event-B se fait par l'appel du helper **GoalsToEvents** (voir la Figure 6.13) qui fait à son tour appel à la règle paresseuse **NewEvent** (présentée dans la Figure 6.12).

Pour chaque niveau de raffinement, la règle **Main_transformation** (voir la Figure 6.11) ajoute aussi l'expression Event-B des différents patrons de raffinement de ce niveau, en appelant le helper **AllRefinementPatterns** (voir la Figure 6.10). Chaque expression est considérée comme un commentaire Event-B, comme par exemple les commentaires associés à la clause REFINES dans les figures 4.11, 4.12 et 4.13. De la même manière, la règle **Main_transformation** appelle le helper **AllProofObligations** qui permet de calculer les différents obligations de preuve et de les ajouter comme des théorèmes Event-B sachant qu'un théorème Event-B est une forme spécifique d'invariant.

6.4 Conclusion

En terme quantitatif, notre module ATL est composé de 32 helpers, de 6 règles paresseuses et d'une seule règle de correspondance principale (soit environ 500 lignes de code ATL). Notre module ATL n'utilise pas par conséquent de règles contenant des constructions impératives (les called rules par exemple). Cela nous a permis de ne pas nous préoccuper des contraintes liées à l'ordre d'exécution des règles. A la différence des outils existants qui se basent sur l'outil propriétaire Objectiver, l'avantage de notre plug-in est que c'est un logiciel libre construit à partir de logiciels libres disponibles (TOPCASED et RODIN). Néanmoins, le plug-in nécessite encore quelques améliorations et extensions afin de semi-automatiser par exemple la prise en compte des buts non-fonctionnels, selon l'approche proposée dans le chapitre 5 de cette thèse.

Conclusion Générale

L'objectif de cette thèse est de faciliter la transition de la phase d'analyse à la phase de spécification, tout en garantissant une distinction entre les exigences fonctionnelles et non-fonctionnelles. L'étude bibliographique a montré que plusieurs approches ont été proposées afin de faciliter le passage vers les spécifications formelles, en faisant appel au paradigme GORE. Cette étude nous a permis d'identifier un ensemble de limites. Des problèmes liés à la non-considération du modèle de buts en entier (les buts abstraits et les types de raffinement de buts) au moment de la dérivation des modèles formels, à la non-prise en compte explicite des exigences non-fonctionnelles et à la quasi-absence d'outils open-source supportant ces approches, ont été relevés.

Cette thèse propose une nouvelle approche permettant de dériver des modèles formels abstraits en considérant le modèle de buts en entier (exprimé en SysML/-KAOS). Notre choix s'est porté sur la méthode SysML/KAOS comme méthode GORE essentiellement parce qu'elle fait la distinction explicite entre les buts fonctionnels et non-fonctionnels. Les travaux présentés dans cette thèse répondent en grande partie à nos objectifs initiaux. En fait, ces travaux ont permis d'établir les premières briques vers la construction du pont entre le monde non-formel et celui du formel. Ces briques équilibrent le compromis entre la complexité des méthodes formelles (Event-B ou le B classique) et l'expressivité d'approches semi-formelles (SysML/KAOS). Par ailleurs, la thèse a montré qu'il y a un effet synergétique entre ces méthodes formelles et semi-formelles : (i) SysML/KAOS permet de fournir une aide pour la construction et la structuration des spécifications formelles abstraites grâce surtout à sa dimension graphique (boîtes, flèches, etc.) ; (ii) l'expression formelle (en B ou en Event-B) du modèle de buts fonctionnels SysML/KAOS permet de prouver des propriétés de cohérence locale sur ce modèle.

Synthèse des contributions

Contrairement aux approches existantes qui dérivent les modèles formels en restant au niveau de la spécification, les approches proposées dans cette thèse ont montré qu'il est possible de remonter les méthodes formelles jusqu'à la phase d'analyse des exigences. Cela a permis de dériver les premières spécifications formelles (en B ou en Event-B) ; c'est-à-dire les modèles formels qui sont plus abstraits que ceux dérivés par les approches présentées dans l'étude bibliographique. C'est la raison pour laquelle nos travaux peuvent être considérés comme complémentaires à ces approches.

La thèse a montré également qu'une fois les modèles formels abstraits obtenus, la conception du futur logiciel peut commencer, c'est-à-dire le processus de raffinement formelle B vers l'implémentation peut commencer. Dans ce contexte,

les approches proposées dans cette thèse ont aidé à fixer les éléments formels abstraits concernés par ce processus de raffinement. En effet, ce processus ne concerne que les éléments formels abstraits correspondant aux exigences logicielles (sous la responsabilité des agents logiciels). Les éléments formels abstraits correspondants aux attentes (sous la responsabilité des agents externes) ne seront pas quant à eux implémentés par le futur logiciel puisqu'ils représentent la spécification de l'environnement du système. De plus, nous avons présenté à travers l'étude de cas que l'avantage des éléments formels abstraits est qu'ils décrivent la manière par laquelle nous pouvons éventuellement juger que le programme final (les éléments formels concrets) est adéquat.

D'un autre côté, l'expression formelle (en B ou en Event-B) du modèle de buts fonctionnels SysML/KAOS a permis de lui donner une sémantique précise de même que les traductions existantes des spécifications UML dans des spécifications B donnent une sémantique formelle aux diagrammes de classes ou aux diagrammes de transitions [Mammar & Laleau 2006, Snook & Butler 2006]. Cette étape de formalisation a permis de prouver des propriétés de cohérence locale sur le modèle de buts SysML/KAOS en prouvant sa contrepartie formelle. L'intérêt des deux approches proposées dans cette thèse est qu'elles se basent sur la technique de la preuve de théorèmes. Cela a permis d'éviter le problème de l'explosion du nombré d'états explorés présent avec les techniques de model-checking sur lesquelles se basent les approches existantes [Fuxman *et al.* 2001, Ponsard *et al.* 2007] (présentées dans la section 2.2) pour la vérification des modèles de buts.

Un autre intérêt de nos travaux est qu'ils se basent dès le début sur la distinction entre les exigences fonctionnelles et celles non-fonctionnelles. Ce choix s'explique par le fait que les exigences non-fonctionnelles sont plus difficiles à traiter dans la mesure où leur impact n'est pas généralement localisé sur une partie du système, mais s'étend sur l'ensemble du système, comme le confirme [Aurum & Wohlin 2005]. C'est pourquoi nous nous sommes intéressés tout d'abord au modèle de buts fonctionnels SysML/KAOS en proposant deux approches différentes (l'une pour le B classique et l'autre pour Event-B) qui permettent de dériver des spécifications formelles abstraites. Etant donné les limites de l'approche dédiée au B classique (voir section 3.3.2), nous nous sommes focalisés par la suite sur l'approche dédiée à Event-B afin de la compléter et l'enrichir par les aspects non-fonctionnels. Pour cela, nous avons proposé une approche permettant de présenter les différentes manières de prendre en compte les buts non-fonctionnels et leurs impacts dans les modèles abstraits Event-B déjà obtenus. En effet, les éléments de la spécification Event-B qui sont liés aux buts non-fonctionnels sont étiquetés afin de localiser dans la spécification formelle les impacts des buts non-fonctionnels sur les buts fonctionnels, permettant ainsi de faciliter la gestion de l'évolution des exigences non-fonctionnelles.

Nous avons également défini des liens de correspondance entre le modèle de buts SysML/KAOS et sa contrepartie formelle Event-B. Ces liens, qui permettent de faciliter la traçabilité entre les deux modèles, peuvent être très utiles en pra-

tique pour : (i) vérifier que toutes les exigences fonctionnelles et non-fonctionnelles SysML/KAOS sont représentées dans le modèle Event-B ; (ii) vérifier que la plupart des éléments dans le modèle Event-B ont des correspondants dans le modèle de buts SysML/KAOS.

D'un point de vue outillage, l'approche permettant de générer une architecture de raffinement Event-B à partir d'un modèle de buts fonctionnels SysML/KAOS est outillée dans un plug-in nommé SysKAOS2EventB. A la différence de la majorité des outils décrits dans la littérature et qui se basent sur l'outil propriétaire Objectiver, l'avantage de notre outil est que c'est un logiciel libre construit à partir de logiciels libres disponibles (TOPCASED et RODIN). La liaison de notre plug-in avec RODIN permet de bénéficier de l'ensemble des plug-in RODIN tels que le prouveur, le model-checker ou l'animateur afin de vérifier et valider le modèle Event-B, et par conséquent sa contrepartie semi-formelle SysML/KAOS.

Notons enfin que l'approche dédiée à Event-B a nécessité l'extension de raffinement Event-B afin de permettre l'expression explicite de la notion d'ordonnancement, d'entrelacement et d'exclusivité entre les événements, sans l'ajout des variables de contrôle. L'inconvénient majeur de ces variables de contrôle est qu'elles s'enchevêtrent avec les spécifications Event-B, compliquant ainsi la traçabilité avec les exigences. L'extension du raffinement Event-B proposée dans cette thèse a rendu les spécifications Event-B plus lisibles et peut permettre une économie des obligations de preuve. Ce dernier avantage est à l'étude et doit être normalement confirmé par d'autres études de cas.

Perspectives

Les travaux présentés dans cette thèse ouvrent plusieurs perspectives. Une première perspective consiste en l'application des différentes approches proposées sur des études de cas plus complexes comme celle du contrôle d'accès aux bâtiments qui a été proposée par différents chercheurs comme banc d'essai de diverses méthodes formelles [Ledru *et al.* 2000].

Une autre perspective concerne les éléments formels qui sont jusqu'à maintenant manuellement complétés par le concepteur tels que l'initialisation, les variables et leurs invariants de typage. Une première solution est de les dériver à partir du modèle objets KAOS qui contient tous les concepts utilisés dans la définition des buts du modèle de buts. Comme le modèle objet KAOS est un diagramme de classes UML, l'idée est de réutiliser par exemple les travaux UML-B définis par [Mammar & Laleau 2006, Snook & Butler 2006]. Une seconde solution consiste à définir un modèle objets SysML/KAOS qui introduit les objets d'une manière incrémentale sous la forme d'un diagramme de classes UML également. Ce dernier aura la même structure qu'un modèle de buts fonctionnels en associant à chaque but les objets concernés. A partir d'un tel modèle objets, il sera possible d'extraire les invariants de collage liant les variables dans la mesure où il y aura des relations de raffinement entre les objets SysML/KAOS.

Comme souligné auparavant, les travaux de cette thèse ont aidé à fixer les éléments formels abstraits concernés par le processus de raffinement B vers l'implémentation. Notre objectif maintenant est d'explorer diverses techniques (le raffinement par exemple) permettant d'automatiser le passage entre ces éléments formels abstraits et les phases postérieures de développement formel (les éléments formels concrets). Dans cette logique, nous comptons explorer une éventuelle correspondance avec quelques approches existantes (présentées dans les sections 2.3.1, 2.4.3 et 2.4.4) dans la mesure où leurs modèles formels dérivés sont plus concrets que les nôtres. Par ailleurs, il serait intéressant d'établir la traçabilité entre les spécifications formelles abstraites obtenues à partir du modèle de buts SysML/KAOS et les phases postérieures de développement formel. Pour cela, l'utilisation de SysML/-KAOS comme point d'entrée peut nous permettre de bénéficier des apports de SysML en terme de traçabilité depuis l'analyse des exigences jusqu'à l'implémentation.

L'approche proposée pour la prise en compte des buts non-fonctionnels mérite également un travail d'extension afin d'améliorer et compléter les résultats actuels. Nous comptons surtout étudier la prise en compte en Event-B des conflits qui peuvent intervenir entre plusieurs buts non-fonctionnels et des impacts négatifs des buts de contribution sur les buts fonctionnels.

D'un point de vue méthodologique, l'approche dédiée à Event-B nécessite quelques améliorations afin de considérer l'aspect modulaire puisque les machines Event-B obtenues peuvent êtres volumineuses incluant beaucoup d'événements. L'idée est d'appliquer alors le mécanisme de la décomposition Event-B [Butler 2009, Abrial 2010] qui permet de réduire la complexité de la spécification Event-B à analyser. D'un autre côté, il serait intéressant de définir des catalogues pour stocker les règles de transformation des buts non-fonctionnels (selon leur type) en Event-B en s'inspirant des catalogues définis dans le NFR framework [Chung *et al.* 2000].

Le plug-in SysKAOS2EventB est opérationnel, mais reste encore dans une stade précoce de développement. Il nécessite ainsi quelques améliorations et extensions afin de prendre en compte les buts non-fonctionnels, selon l'approche proposée dans le chapitre 5. De plus, il serait bénéfique que le plug-in SysKAOS2EventB signale sur le modèle de buts SysML/KAOS les éléments à l'origine d'une éventuelle obligation de preuve non déchargée ou d'une éventuelle animation Event-B non réussie. L'idée est que le plug-in SysKAOS2EventB puisse proposer les corrections à effectuer sur ce modèle de buts.

Bibliographie

[Abrial 1996a] Jean-Raymond Abrial. The B-Book : Assigning programs to meanings. Cambridge University Press, 1996.

[Abrial 1996b] Jean-Raymond Abrial. *Extending B without changing it (for developing distributed systems)*. In H. Habrias editor, The First Conference on the B-Method, pages 169–190, November 1996.

[Abrial 2003] Jean-Raymond Abrial. *B : Passé, Présent, Futur*. Technique et Science Informatiques, vol. 22, no. 1, pages 89–118, 2003.

[Abrial 2006] Jean-Raymond Abrial. *Formal methods in industry : Achievements, Problems, Future*. In Leon J. Osterweil, H. Dieter Rombach et Mary Lou Soffa, editeurs, ICSE, pages 761–768. ACM, 2006.

[Abrial 2010] Jean-Raymond Abrial. Modeling in Event-B : System and Software Engineering. Cambridge University Press, 2010.

[Anton 1996] Annie I. Anton. *Goal-Based Requirements Analysis*. In Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96), pages 136–144, Washington, DC, USA, 1996. IEEE Computer Society.

[Anton 1997] Annie I. Anton. *Goal identification and refinement in the specification of software-based information systems*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 1997.

[Aurum & Wohlin 2005] Aybüke Aurum et Claes Wohlin. Engineering and managing software requirements. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Aziz *et al.* 2009] Benjamin Aziz, Alvaro E. Arenas, Juan Bicarregui, Christophe Ponsard et Philippe Massonet. *From Goal-Oriented Requirements to Event-B Specifications*. In First Nasa Formal Method Symposium (NFM 2009), Moffett Field, California, USA, 2009.

[Badeau & Amelot 2005] Frédéric Badeau et Arnaud Amelot. *Using B as a High Level Programming Language in an Industrial Project : Roissy VAL*. In Helen Treharne, Steve King, Martin C. Henson et Steve A. Schneider, editeurs, ZB, volume 3455 of *Lecture Notes in Computer Science*, pages 334–354. Springer, 2005.

[Behm *et al.* 1999] Patrick Behm, Paul Benoit, Alain Faivre et Jean-Marc Meynadier. *Météor : A Successful Application of B in a Large Project*. In Jeannette M. Wing, Jim Woodcock et Jim Davies, editeurs, World Congress on Formal Methods, volume 1708 of *Lecture Notes in Computer Science*, pages 369–387. Springer, 1999.

[Bendisposto *et al.* 2008] Jens Bendisposto, Michael Leuschel, O. Ligot et Mireille Samia. *La validation de modèles Event-B avec le plug-in ProB pour RODIN*. Technique et Science Informatiques, vol. 27, no. 8, pages 1065–1084, 2008.

[Bézivin *et al.* 2003] Jean Bézivin, Grégoire Dupé, Frédéric Jouault, Gilles Pitette et Jamal Eddine Rougui. *First experiments with the ATL model transformation language : Transforming XSLT into XQuery*. In 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, Anaheim, California, USA, 2003.

[Bézivin *et al.* 2004] Jean Bézivin, Frédéric Jouault, Peter Rosenthal et Patrick Valduriez. *Modeling in the Large and Modeling in the Small*. In Uwe Aßmann, Mehmet Aksit et Arend Rensink, editeurs, MDAFA, volume 3599 of *Lecture Notes in Computer Science*, pages 33–46. Springer, 2004.

[Bézivin 2004] Jean Bézivin. *Sur les principes de base de l'ingénierie des modèles*. L'OBJET, vol. 10, no. 4, pages 145–157, 2004.

[Bicarregui *et al.* 2008] Juan Bicarregui, Alvaro Arenas, Benjamin Aziz, Philippe Massonet et Christophe Ponsard. *Towards Modelling Obligations in Event-B*. In Egon Börger, Michael J. Butler, Jonathan P. Bowen et Paul Boca, editeurs, ABZ, volume 5238 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2008.

[Bjørner & Jones 1978] Dines Bjørner et Cliff B. Jones, editeurs. The vienna development method : The meta-language, volume 61 of *Lecture Notes in Computer Science*. Springer, 1978.

[Bjørner *et al.* 1996] Nikolaj Bjørner, Anca Browne, Eddie Chang, Michael Colón, Arjun Kapur, Zohar Manna, Henny Sipma et Tomás E. Uribe. *STeP : Deductive-Algorithmic Verification of Reactive and Real-Time Systems*. In Proceedings of the 8th International Conference on Computer Aided Verification, CAV '96, pages 415–418, London, UK, 1996. Springer-Verlag.

[Blanc 2005] Xavier Blanc. Mda en action : Ingénierie logicielle guidée par les modèles. Paris, Eyrolles, 2005.

[Boehm & Papaccio 1988] B. W. Boehm et P. N. Papaccio. *Understanding and Controlling Software Costs*. IEEE Trans. Softw. Eng., vol. 14, pages 1462–1477, October 1988.

[Boehm 1988] Barry W. Boehm. *A Spiral Model of Software Development and Enhancement*. Computer, vol. 21, pages 61–72, May 1988.

[Brandozzi & Perry 2001] Manuel Brandozzi et Dewayne E Perry. *Transforming Goal Oriented requirements specifications into Architectural Prescriptions*. In Workshop From Software Requirements to Architectures (STRAW'01) at International Conference on Software Engineering (ICSE), pages 54–61, Toronto, Canada, 2001.

[Burns & Hayes 2010] Alan Burns et Ian J. Hayes. *A Timeband Framework for Modelling Real-Time Systems*. Real-Time Syst., vol. 45, pages 106–142, June 2010.

[Butler 2009] Michael Butler. *Decomposition Structures for Event-B*. In Michael Leuschel et Heike Wehrheim, editeurs, IFM, volume 5423 of *Lecture Notes in Computer Science*, pages 20–38. Springer, 2009.

[Cabral & Sampaio 2008] Gustavo Cabral et Augusto Sampaio. *Formal Specification Generation from Requirement Documents*. Electron. Notes Theor. Comput. Sci., vol. 195, pages 171–188, January 2008.

[Castro *et al.* 2001] Jaelson Castro, Manuel Kolp et John Mylopoulos. *A Requirements-Driven Development Methodology*. In Klaus R. Dittrich, Andreas Geppert et Moira C. Norrie, editeurs, CAiSE, volume 2068 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 2001.

[Castro *et al.* 2002] Jaelson Castro, Manuel Kolp et John Mylopoulos. *Towards Requirements-Driven Information Systems Engineering : The Tropos Project* . Information Systems, vol. 27, pages 365–389, 2002.

[Chung *et al.* 2000] Lawrence Chung, Brian A. Nixon, Eric Yu et John Mylopoulos. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

[Cimatti *et al.* 2000] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia et Marco Roveri. *NUSMV : A New Symbolic Model Checker*. STTT, vol. 2, no. 4, pages 410–425, 2000.

[Clarke & Emerson 1981] Edmund M. Clarke et E. Allen Emerson. *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*. In Dexter Kozen, editeur, Logic of Programs, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[Cysneiros & Kushniruk 2003] Luiz Marcio Cysneiros et André Kushniruk. *Bringing Usability to the Early Stages of Software Development*. In RE, pages 359–360. IEEE Computer Society, 2003.

[Dardenne *et al.* 1993] Anne Dardenne, Axel van Lamsweerde et Stephen Fickas. *Goal-directed requirements acquisition*. Sci. Comput. Program., vol. 20, pages 3–50, April 1993.

[Darimont & van Lamsweerde 1996] Robert Darimont et Axel van Lamsweerde. *Formal refinement patterns for goal-driven requirements elaboration*. In Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering, SIGSOFT '96, pages 179–190, New York, NY, USA, 1996. ACM.

[de Sousa & Russo 2010] Thiago C. de Sousa et Aryldo G. Russo. *Starting B Specifications from Use Cases*. In Frappier et al. [Frappier *et al.* 2010], page 411.

[Dehbonei & Mejia 1994] Babak Dehbonei et Fernando Mejia. *Formal Methods in the Railways Signalling Industry*. In Maurice Naftalin, B. Tim Denvir et Miquel Bertran, editeurs, FME, volume 873 of *Lecture Notes in Computer Science*, pages 26–34. Springer, 1994.

[Devroey 2010] Xavier Devroey. Building a bridge between Goal-Oriented Requirements with KAOS and Event-B System Specifications. Master's thesis, Facultés Universitaires Notre-Dame de la Paix Faculté d'Informatique, Namur, Belgium, 2010.

[Diaw *et al.* 2010] Samba Diaw, Rédouane Lbath et Bernard Coulette. *État de l'art sur le développement logiciel basé sur les transformations de modèles*. Technique et Science Informatiques, vol. 29, no. 4-5, pages 505–536, 2010.

[Ebert 1997] Christof Ebert. *Dealing with Nonfunctional in Large Software System*. Annals of Software Engineering, vol. 3, no. 1, pages 367–395, 1997.

[El-Maddah & Maibaum 2003] Islam El-Maddah et Tom Maibaum. *Goal-Oriented Requirements Analysis for Process Control Systems Design*. In Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design, MEMOCODE '03, pages 45–46. IEEE Computer Society, 2003.

[Finkelstein & Dowell 1996] A. Finkelstein et J. Dowell. *A comedy of errors : the London Ambulance Service case study*. In Proceedings of the 8th International Workshop on Software Specification and Design, IWSSD '96, pages 2–4, Washington, DC, USA, 1996. IEEE Computer Society.

[Fitzgerald *et al.* 2005] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat et Marcel Verhoef. Validated Designs For Object-oriented Systems. Springer-Verlag TELOS, Santa Clara, CA, USA, 2005.

[Frappier *et al.* 2010] Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau et Steve Reeves, editeurs. Abstract state machines, alloy, b and z, second international conference, abz 2010, orford, qc, canada, february 22-25, 2010. proceedings, volume 5977 of *Lecture Notes in Computer Science*. Springer, 2010.

[Friedenthal *et al.* 2008] Sanford Friedenthal, Alan Moore et Rick Steiner. A Practical Guide to SysML : The Systems Modeling Language. Morgan Kaufmann Publishers Inc., 2008.

[Fuxman *et al.* 2001] Ariel Fuxman, John Mylopoulos, Marco Pistore et Paolo Traverso. *Model Checking Early Requirements Specifications in Tropos*. In RE, pages 174–181. IEEE Computer Society, 2001.

[Glinz 2007] Martin Glinz. *On Non-Functional Requirements*. In RE, pages 21–26. IEEE, 2007.

[Gnaho & Semmak 2011] Christophe Gnaho et Farida Semmak. *Une extension SysML pour l'ingénierie des exigences non fonctionnelles orientée but*. Ingénierie des Systèmes d'Information, vol. 16, no. 1, pages 9–32, 2011.

[Goknil *et al.* 2011] Arda Goknil, Ivan Kurtev, Klaas Berg et Jan-Willem Veldhuis. *Semantics of trace relations in requirements models for consistency checking and inferencing*. Softw. Syst. Model., vol. 10, pages 31–54, February 2011.

[Hassan 2009] Riham Hassan. *Formal Analysis and Design for Engineering Security (FADES)*. PhD thesis, Virginia Polytechnic Institute and State University, Virginia, USA, 2009.

[Hoare 1969] C. A. R. Hoare. *An axiomatic basis for computer programming*. Commun. ACM, vol. 12, pages 576–580, October 1969.

[Iliasov 2009] Alexei Iliasov. *On Event-B and Control Flow*. Rapport technique, DEPLOY Project, 2009.

[Jackson 2001] Michael Jackson. Problem frames : analyzing and structuring software development problems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Jastram *et al.* 2009] Michael Jastram, Michael Leuschele, Jens Bendisposto et Aryldo G. Russo. *Mapping Requirements to B models*. Rapport technique 104, DEPLOY project, 2009.

[Jastram *et al.* 2010] Michael Jastram, Stefan Hallerstede, Michael Leuschel et Aryldo G. Russo. *An Approach of Requirements Tracing in Formal Refinement*. In Gary T. Leavens, Peter W. O'Hearn et Sriram K. Rajamani, editeurs, VSTTE, volume 6217 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2010.

[Jouault & Kurtev 2006] Frédéric Jouault et Ivan Kurtev. *On the architectural alignment of ATL and QVT*. In Proceedings of the 2006 ACM symposium on Applied computing, SAC '06, pages 1188–1195, New York, NY, USA, 2006. ACM.

[Kaiya *et al.* 2002] Haruhiko Kaiya, Hisayuki Horai et Motoshi Saeki. *AGORA : Attributed Goal-Oriented Requirements Analysis Method*. In 10th Anniversary IEEE Joint International Conference on Requirements Engineering, pages 13–22. IEEE Computer Society, 2002.

[Kassab *et al.* 2007] Mohamad Kassab, M. Daneva et Olga Ormandjieva. *Scope Management of Non-Functional Requirements*. In EUROMICRO '07 : Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pages 409–417. IEEE Computer Society, 2007.

[Koymans 1992] Ron Koymans. Specifying message passing and time-critical systems with temporal logic. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[Krishna *et al.* 2004] Aneesh Krishna, Aditya K. Ghose et Sergiy A. Vilkomir. *Co-Evolution of Complementary Formal and Informal Requirements*. In IWPSE, pages 159–164. IEEE Computer Society, 2004.

[Laleau *et al.* 2010] Régine Laleau, Farida Semmak, Abderrahman Matoussi, Dorian Petit, Ahmed Hammad et Bruno Tatibouët. *A First Attempt to Combine SysML Requirements Diagrams and B*. ISSE, vol. 6, no. 1-2, pages 47–54, 2010.

[Ledru *et al.* 2000] Yves Ledru, Gerard Padiou et Jacques Jaray. *Etude de cas : système de contrôle d'accès*. Rapport technique, Dans Journées AFADL'2000, http ://vasco.imag.fr/afadl2000/EtudeDeCas/, 2000.

[Letier 2001] Emmanuel Letier. *Reasoning About Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Université catholique de Louvain, Faculté des Sciences appliquées, Louvain-la-Neuve, Belgium, 2001.

[Leuschel & Butler 2003] Michael Leuschel et Michael J. Butler. *ProB : A Model Checker for B*. In Keijiro Araki, Stefania Gnesi et Dino Mandrioli, editeurs, FME, volume 2805 of *Lecture Notes in Computer Science*, pages 855–874. Springer, 2003.

[Liu & Yu 2001] Lin Liu et Eric Yu. *From Requirements to Architectural Design : Using Goals and Scenarios*. In First International Workshop From Software Requirements to Architectures (STRAW 01), Toronto, Canada, May 2001.

[Lynch & Vaandrager 1995] Nancy A. Lynch et Frits W. Vaandrager. *Forward and Backward Simulations : I. Untimed Systems*. Information and Computation, vol. 121, no. 2, pages 214–233, 1995.

[Mammar & Laleau 2006] Amel Mammar et Régine Laleau. *A Formal Approach Based on UML and B for the Specification and Development of Database Applications*. Automated Software Engg., vol. 13, pages 497–528, October 2006.

[Manna & Pnueli 1984] Zohar Manna et Amir Pnueli. *Adequate proof principles for invariance and liveness properties of concurrent programs*. Sci. Comput. Program., vol. 4, pages 257–289, December 1984.

[Matoussi & Laleau 2008] Abderrahman Matoussi et Régine Laleau. *A Survey of Non-Functional Requirements in Software Development Process*. Rapport technique TR-LACL-2008-7, LACL, University of Paris-Est, 2008.

[Matoussi & Laleau 2011a] Abderrahman Matoussi et Régin Laleau. *Un outil de construction de spécifications abstraites Event-B dirigée par les buts*. Ingénierie des Systèmes d'Information, vol. 16, no. 5, pages 143–166, 2011.

[Matoussi & Laleau 2011b] Abderrahman Matoussi et Régine Laleau. *Une première approche de traçabilité entre modèles d'exigences non-fonctionnelles et spécifications abstraites Event-B*. In Actes du XXIXème Congrès INFORSID, pages 301–316, Lille, France, Mai 2011.

[Matoussi & Petit 2010] Abderrahman Matoussi et Dorian Petit. *Improving Traceability between KAOS Requirements Models and B Specifications*. In Frappier et al. [Frappier *et al.* 2010], pages 401–402.

[Matoussi *et al.* 2010a] Abderrahman Matoussi, Frédéric Gervais et Régine Laleau. *An Event-B formalization of KAOS goal refinement patterns*. Rapport technique TR-LACL-2010-1, LACL, University of Paris-Est, *http ://lacl.univ-paris12.fr/Rapports/TR/TR-LACL-2010-1.pdf*, 2010.

[Matoussi *et al.* 2010b] Abderrahman Matoussi, Frédéric Gervais et Régine Laleau. *Specification of a Localization Component Driven by a Goal-Based Approach : Some Lessons We Learned*. In Jim Davies, Leila Silva et Adenilso da Silva Simão, editeurs, SBMF, volume 6527 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2010.

[Matoussi *et al.* 2011] Abderrahman Matoussi, Frédéric Gervais et Régine Laleau. *A Goal-Based Approach to Guide the Design of an Abstract Event-B Specification*. In Isabelle Perseil, Karin Breitman et Roy Sterritt, editeurs, ICECCS, pages 139–148. IEEE Computer Society, 2011.

[Muller *et al.* 2005] Pierre-Alain Muller, Franck Fleurey et Jean-Marc Jézéquel. *Weaving Executability into Object-Oriented Meta-languages*. In Lionel C. Briand et Clay Williams, editeurs, MoDELS, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2005.

[Nakagawa *et al.* 2007] Hiroyuki Nakagawa, Kenji Taguchi et Shinichi Honiden. *Formal specification generator for KAOS : model transformation approach to generate formal specifications from KAOS requirements models*. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ASE '07, pages 531–532. ACM, 2007.

[Neto *et al.* 2000] Jaime De Melo Sabat Neto, Julio Cesar Sampaio Leite et Luiz Marcio Cysneiros. *Non-Functional Requirements for Object-Oriented Modeling*. In WER, pages 109–125, 2000.

[Nuseibeh & Easterbrook 2000] Bashar Nuseibeh et Steve Easterbrook. *Requirements engineering : a roadmap*. In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 35–46. ACM, 2000.

[Petit *et al.* 2008] Dorian Petit, Thierry Delot et Vincent Poirriez. *Chapter 3 : Localization Component : Relevant Properties and First Algorithms*. Rapport technique, Livrable L1.1 TACOS project : Model for the Land Transport Domain, *http ://tacos.loria.fr/drupal/ ?q=system/files/Livrable1.1-4Fev08.pdf*, 2008.

[Pnueli 1977] Amir Pnueli. *The temporal logic of programs*. In Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[Ponsard & Devroey 2011] Christophe Ponsard et Xavier Devroey. *Generating High-Level Event-B System Models from KAOS Requirements Models*. In Actes du XXIXème Congrès INFORSID, pages 317–332, Lille, France, Mai 2011.

[Ponsard & Dieul 2006] Christophe Ponsard et Emmanuel Dieul. *From Requirements Models to Formal Specifications in B*. In REMO2V'2006, Luxembourg, 2006.

[Ponsard *et al.* 2007] C. Ponsard, P. Massonet, J. F. Molderez, A. Rifaut, A. Van Lamsweerde et H. Tran Van. *Early verification and validation of mission critical systems*. Form. Methods Syst. Des., vol. 30, pages 233–247, June 2007.

[Rashid & Chitchyan 2008] Awais Rashid et Ruzanna Chitchyan. *Aspect-oriented requirements engineering : a roadmap*. In Proceedings of the 13th international workshop on Early Aspects, EA '08, pages 35–41, New York, NY, USA, 2008. ACM.

[Robinson & Pawlowski 1998] William N. Robinson et Suzanne D. Pawlowski. *Surfacing Root Requirements Interactions from Inquiry Cycle Requirements Documents*. In ICRE, pages 82–89. IEEE Computer Society, 1998.

[Rolland *et al.* 1998] Colette Rolland, Carine Souveyet et Camille Ben Achour. *Guiding Goal Modeling Using Scenarios*. IEEE Trans. Softw. Eng., vol. 24, pages 1055–1071, December 1998.

[Roscoe *et al.* 1997] A. W. Roscoe, C. A. R. Hoare et Richard Bird. The theory and practice of concurrency. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[Sangiorgi 1996] Davide Sangiorgi. *Locality and interleaving semantics in calculi for mobile processes*. Theor. Comput. Sci., vol. 155, pages 39–83, February 1996.

[Sekerinski 1998] Emil Sekerinski. *Graphical Design of Reactive Systems*. In Didier Bert, editeur, B, volume 1393 of *Lecture Notes in Computer Science*, pages 182–197. Springer, 1998.

[Servat 2007] Thierry Servat. *BRAMA : A New Graphic Animation Tool for B Models*. In Jacques Julliand et Olga Kouchnarenko, editeurs, B, volume 4355 of *Lecture Notes in Computer Science*, pages 274–276. Springer, 2007.

[Snook & Butler 2006] Colin Snook et Michael Butler. *UML-B : Formal modeling and design aided by UML*. ACM Trans. Softw. Eng. Methodol., vol. 15, pages 92–122, January 2006.

[Snook & Butler 2008] Colin Snook et Michael Butler. *UML-B and Event-B : an integration of languages and tools*. In The IASTED International Conference on Software Engineering – SE'08, February 2008.

[Snook *et al.* 2010] Colin Snook, Fabian Fritz et Alexei Illisaov. *An EMF Framework for Event-B*. In Workshop on Tool Building in Formal Methods - ABZ Conference, Orford, QC, Canada, February 2010.

[Spivey 1988] J. M. Spivey. Understanding z : a specification language and its formal semantics. Cambridge University Press, New York, NY, USA, 1988.

[Steinberg *et al.* 2009] Dave Steinberg, Frank Budinsky, Marcelo Paternostro et Ed Merks. EMF : Eclipse Modeling Framework. Addison-Wesley, 2. édition, 2009.

[Sutcliffe 2003] Alistair Sutcliffe. *Scenario-Based Requirements Engineering*. In Proceedings of the 11th IEEE International Conference on Requirements Engineering, pages 320–, Washington, DC, USA, 2003. IEEE Computer Society.

[Tatibouët *et al.* 2003] Bruno Tatibouët, Antoine Requet, Jean-Christophe Voisinet et Ahmed Hammad. *Java Card Code Generation from B Specifications*. In Jin Song Dong et Jim Woodcock, editeurs, ICFEM, volume 2885 of *Lecture Notes in Computer Science*, pages 306–318. Springer, 2003.

[Tonu 2006] Subrina Anjum Tonu. Incorporating Non-Functional Requirements with UML Models. Master's thesis, University of Waterloo, Ontario, Canada, 2006.

[van Lamsweerde & Letier 2002] Axel van Lamsweerde et Emmanuel Letier. *From Object Orientation to Goal Orientation : A Paradigm Shift for Requirements Engineering*. In Martin Wirsing, Alexander Knapp et Simonetta Balsamo, editeurs, RISSEF, volume 2941 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2002.

[van Lamsweerde 2009] Axel van Lamsweerde. Requirements Engineering : From System Goals to UML Models to Software Specifications. Wiley, 2009.

[Yu 1995] Eric Yu. *Modelling Strategic Relationships for Process Reengineering*. Ph.D. Thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[Yu 1997] Eric Yu. *Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering*. In Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, RE '97, pages 226–235, Washington, DC, USA, 1997. IEEE Computer Society.

[Yue 1987] K. Yue. *What Does It Mean to Say that a Specification is Complete ?* In Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design, 1987.

[Zave 1997] Pamela Zave. *Classification of Research Efforts in Requirements Engineering*. ACM Comput. Surv., vol. 29, no. 4, pages 315–321, 1997.

Table des figures