Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option : *Génie Logiciel (G.L)*

Thème

Vérification des architectures systèmes basées sur SysML à l'aide de la méthode B

Réalisé par :

- Fatima Zohra BEMRAH
- Meriem KHERBOUCHE

Présenté le 25 Juin 2018 devant le jury composé de MM.

M. Abdelkrim BENAMAR	Président
Mme. Yassamine SELADJI	Examinatrice
M. Mohamed MESSABIH	Encadrant
M. Christian ATTIOGBE	Co-encadrant

Année universitaire : 2017-2018

REMERCIEMENT

En premier lieu, nous remercions le bon Dieu, tout puissant, de nous avoir donné la force nécessaire pour faire ce modeste travail, et la persévérance pour surmonter toutes les difficultés.

Le travail présenté dans ce mémoire a été réalisé au laboratoire de recherche de l'informatique « LRIT » au niveau de l'université Abou Bakr Belkaid Tlemcen, sous l'encadrement de M. Mohamed MESSABIHI.

Nous adressons toute notre gratitude à notre cher encadrant M. Mohamed MESSABIHI pour sa patience, sa disponibilité, sa gentillesse, son abnégation et ses encouragements, ainsi que son savoir qu'il nous a transmis avec toute modestie par les outils méthodologiques indispensables à la conduite de cette recherche. Ce travail n'aurait pu être achevé et mené à terme que grâce à ses judicieux conseils. Merci infiniment.

Nous remercions aussi notre co-encadrant M. J. Christian ATTIOGBÉ pour sa disponibilité, son soutien à distance, et son encouragement pour continuer la recherche.

Nos vifs remerciements sont également adressés aux membres du jury M. Abdelkerim BENAMAR et Mme. Yassamine SELADJI pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail.

Sans oublier de remercier nos chers parents pour leur amour, leur soutien, leur patience, et leurs prières, nos proches, nos amis, nos collègues pour le travail d'équipe que nous avons abordé au niveau du laboratoire de recherche.

DÉDICACE

Je dédie ce modeste travail à ALLAH Le très Haut, le très Grand, L'Omniscient, l'Omnipotent. Le Tout Puissant, le très miséricordieux d'avoir permis à ce travail d'aboutir à son terme.

À mon père, toi qui m'a donné tant de choses et tu continues à le faire, sans jamais te plaindre. De tous les pères, tu as été le meilleur, tu as su m'entourer d'attention, m'inculquer les valeurs nobles de la vie, m'apprendre le sens du travail, de l'honnêteté et de la responsabilité. Merci d'avoir été toujours là pour moi, un grand soutien tout au long de mes études. Des mots ne pourront jamais exprimer la profondeur de mon respect, ma considération, ma reconnaissance et mon amour éternel. Que Dieu te préserve des malheurs de la vie afin que tu demeures le flambeau illuminant mon chemin.

À ma mère, A la plus douce et la plus merveilleuse de toutes les mamans. A une personne qui m'a tout donné sans compter. Tu n'as pas cessé de me soutenir et de m'encourager, ton amour, ta générosité exemplaire et ta présence constante ont fait de moi ce que je suis aujourd'hui. Tes prières ont été pour moi un grand soutien tout au long de mes études. Puisse Dieu tout puissant te protéger du mal, te procurer longue vie, santé et bonheur afin que je puisse te rendre un minimum de ce que je te dois. Je t'aime maman.

À ma sœur Soumia ; mon frère Mohamed et sa femme Fatima ; Je vous dédie ce travail en témoignage de mon amour et mon attachement. Puisse nos fraternels liens se pérenniser et consolider encore. Je ne pourrais d'aucune manière exprimer ma profonde affection et mon immense gratitude pour tous les sacrifices consentis, votre aide et votre générosité extrêmes ont été pour moi une source de courage, de confiance et de patience.

Je remercie spécialement Feth Allah pour ses conseils, ses remarques pertinentes et son encouragement. J'implore DIEU qu'il t'apporte bonheur, amour et que tes rêves se réalisent.

À M. Houari MAHFOUD pour ses encouragements, à M. Ismail SMAHI et à tous mes enseignants ; à mon amie et binôme Fatima, à mes collègues et tous ceux qui m'ont soutenu de près ou de loin qu'il me soit permis aujourd'hui de vous assurer ma profonde et ma grande reconnaissance.

Meriem KHERBOUCHE

DÉDICACE

Au nom du Dieu clément et miséricordieux louange à ALLAH le tout puissant.

Je dédie ce modeste travail en signe de respect et reconnaissance :

À la mémoire de ma chère grand-mère paternelle et grand-père maternel.

À mon très cher papa qui est toujours présent à mes côtés pour m'encourager, me soutenir, m'orienter, me conseiller et surtout me donner la force et la volonté de poursuivre mes études sans limites.

À ma très chère maman pour son amour, son attention et son affection infinie, pour ses prières et sa bénédiction qui m'ont aidés et facilités tout mon parcours dans mes études,

À mon cher mari Youcef pour son soutien moral et matériel, son aide et ses encouragements quotidiens.

À mes chères petites sœurs Amel et Asma.

À ma chère tante Nadéra pour son soutien.

À ma chère copine et binôme Meriem

À tous ceux qui me sont chers.

Fatima Zohra BEMRAH

SOMMAIRE

1

INTRODUCTION

Les logiciels informatiques sont aujourd'hui omniprésents, allant d'une simple montre à un système aéronautique complexe, d'où leur importance et criticité sur la vie de l'être humain. Malgré la présence de ces logiciels au cœur de nombreux produits de la vie quotidienne, on trouve que les avions ne s'écrasent pas aussi souvent, cela est parce que ces logiciels critiques sont développés en utilisant des méthodes rigoureuses et vérifiés également au début du cycle du développement par des méthodes formelles.

Au cours du développement des logiciels complexes, les activités de vérification sont cruciales étant donné qu'elles permettent de s'assurer de la fiabilité et la qualité du logiciel produit. C'est pourquoi les erreurs d'exigence découvertes tôt dans le processus de développement sont moins coûteuses à corriger que si ces mêmes erreurs sont découvertes tard.

Par conséquent, il est opportun d'étudier les méthodes permettant de détecter les erreurs d'exigences au début du processus de développement. Une des pistes prometteuses pour la réduction de ces coûts de vérification est l'avancement de processus de vérification pour gagner en marge de manœuvre en utilisant des méthodes de vérification formelles. Ces méthodes s'appuient sur des fondements mathématiques et permettent d'effectuer des tâches de vérification à forte valeur ajoutée au cours du développement.

1.1/ CONTEXTE

À tout moment, le système peut être amené a intégrer de nouvelles fonctionnalités. Cependant, le fait de voir et de développer le système comme une unité constitue une barrière à son évolution, où il sera très difficile de préciser les parties du système qui sont altérées par chaque évolution. En outre, la vérification du système après modification sera de plus en plus complexe.

En effet, les inconvénients de cette approche ont changé la manière de concevoir et de développer ces systèmes. C'est ce qui justifie la tendance des nouvelles approches, telles que l'approche CBD (Component-Based Development) qui prend le système comme un ensemble de composants. Développer des systèmes en réutilisant et en adaptant un ensemble de composants constitue le sujet central du développement. Il permet de s'attaquer aux problèmes des anciennes approches, mais il crée également de nouveaux défis et critères qui doivent être pris en compte lors du développement.

Lors de l'assemblage de composants conçus séparément, il existe une forte probabilité

de rencontrer le problème d'incompatibilité entre eux. Ces compatibilités peuvent concerner par exemple le nom des services, ainsi que l'ordre dans lequel le composant demande (ou propose) ses services a son environnement.

Pour résoudre ce problème et faciliter la communication entre les parties prenantes. La communauté d'ingénierie système propose d'utiliser des langages de haut niveau qui adoptent le principe d'utiliser le composant comme unité de développement. Cela apparaît clairement à travers SysML, un langage qui est adopté par l'OMG, il est utilisé pour concevoir des systèmes qui incluent des logiciels et du matériel. Le langage de modélisation du système (SysML), à travers ses diagrammes, favorise le point de vue qui prend le système comme un ensemble de composants.

Dans SysML, nous les appelons «blocs». Un bloc est une unité modulaire de la description du système. Il peut inclure à la fois des caractéristiques structurelles et comportementales, telles que des propriétés et des opérations. SysML propose également de nombreux diagrammes pour représenter le comportement des blocs. Il met également à la disposition des développeurs le schéma d'exigences qui permet de saisir les différentes exigences et d'établir le lien entre elles et entre les blocs responsables de leur satisfaction.

Ce privilège donné à SysML ne signifie pas qu'il prendra la place des méthodes formelles. Mais il les remplace à un niveau de représentation du système, où nous avons besoin d'une spécification de haut niveau du système, pour permettre une meilleure communication entre les parties prenantes du projet CBD. Il faut aussi mentionner que SysML manque de sémantique formelle, ce qui rend très intéressant l'introduction de méthodes formelles.

C'est dans ce contexte que l'intégration des méthodes formelles semble intéressante car elle permet de spécifier formellement des interactions de composants et donc d'assurer la fiabilité des systèmes basés sur les composants en vérifiant la compatibilité des composants.

1.2/ PROBLÉMATIQUE

L'ingénierie des exigences présente toujours plusieurs défis pour les chercheurs et les industriels, et surtout dans les méthodes de développement des logiciels modernes, car elles nécessitent une implication permanente du client et permettent une grande réactivité à ses demandes. C'est pourquoi de nombreux projets de développement de logiciels échouent en raison d'une mauvaise gestion des exigences.

La validation des exigences est l'un des processus les plus cruciaux pour déterminer si les besoins des clients et les attentes d'un système logiciel sont suffisamment correctes, cohérentes et sans ambiguïté. En outre, la capture des exigences implique l'analyse du langage naturel, qui est souvent imprécis, et peut avoir des conséquences négatives dans le succès du projet, donc la validation des exigences doit être effectuée dans un stade plus avancé du processus de développement, afin de corriger l'inexactitude, l'incohérence et les questions ambiguës, en réduisant les coûts et en minimisant les efforts.

1.3/ OBJECTIFS

Nous prenons le langage SysML comme cadre de modélisation de systèmes complexes. SysML est un langage semi-formel ; mais il est adopté par certains industriels pour décrire leur systèmes, à différents niveaux. Notre but est d'apporter des méthodes et des techniques rigoureuses pour les utilisateurs de SysML, en permettant d'analyser des modèles SysML et de pouvoir construire rigoureusement des systèmes opérationnels à partir de tels modèles. Les objectifs de ce projet sont :

- définir d'abord l'ensemble d'éléments liés à l'enrichissement des exigences et des blocs proposées par SysML.
- Spécifier comment ces nouveaux concepts sont intégrés aux concepts d'exigences et des blocs SYSML (intégration au niveau du méta-modèle).
- Donner une sémantique précise à cette extension grâce à la définition des règles de dérivation d'un modèle étendu à une spécification B formelle.
- Exploiter les exigences étendues pour vérifier globalement une composition si elle satisfait bien ses propriétés. La technique de développement par raffinements successifs (à l'instar de la méthode B) rejoint le principe des approches "top-down" et semble prometteuse dans le cadre de la vérification des compositions.
- Implémenter les différents algorithmes de dérivation en tant que plugin.
- Enfin, se servir des outils de vérification (Atelier B) pour expérimenter l'approche sur une étude de cas.

1.4/ ORGANISATION DU MÉMOIRE

Ce manuscrit est structuré en cinq chapitres

- Nous commençons dans un premier temps, par une introduction générale en montrant le contexte et la problématique et enfin les objectifs de notre projet.
- Le second chapitre est consacré à l'état de l'art où nous donnons un aperçu sur l'ingénierie système, puis nous introduisons l'ingénierie des exigences, l'un des processus de l'ingénierie système. Ensuite, nous passons à l'ingénierie dirigée par les modèles et la modélisation des systèmes et nous finissons par la validation des spécifications à l'aide des méthodes formelles et notre choix de SysML et la méthode B.
- Le troisième chapitre aborde les langages et les outils de modélisation que nous avons adoptés dans le cadre de notre projet.
- Le quatrième chapitre présente l'approche que nous avons proposée pour intégrer la vérification formelle dans la modélisation SysML. Une étude de cas est également présentée à titre d'illustration.
- Ce manuscrit s'achève par une conclusion générale et quelques perspectives.

2

ÉTAT DE L'ART

Actuellement, l'étude des produits industriels est abordée selon une approche systémique qui est défini comme un ensemble de composants en interaction, le comportement du système ne résulte pas seulement des comportements individuels de ses composants, mais aussi de la façon dont ils interagissent entre eux. La systémique est une approche complètement transversale, qui possède de nombreux domaines d'utilisation. Son application à la conception et à l'analyse des produits industriels s'appelle l'ingénierie des systèmes. Dans ce cadre, la notion de système s'étend sur plusieurs niveaux : un composant peut très bien être vu comme un système et décomposé à son tour, et on le qualifiera alors de sous-système.

2.1/ LA MODÉLISATION DES SYSTÈMES

La modélisation des systèmes a deux objectifs principaux : simuler leur comportement et communiquer des descriptions. En sciences de l'ingénieur, elle peut s'effectuer selon trois grands points de vue complémentaires :

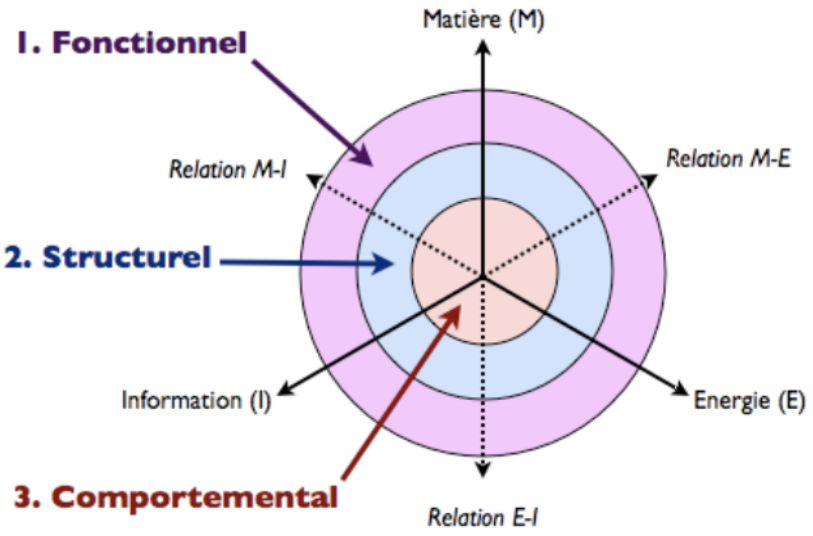

FIGURE 2.1 – Les trois grands points de vue des sciences de l'ingénieur : fonctionnel, structurel et comportemental

1. Le point de vue fonctionnel, qui consiste à décrire les actions effectuées par le système pour répondre à la question « A quoi sert-il ? » ;

2. Le point de vue structurel qui, dans le cadre de la systémique, consiste à décrire les composants du produit et de son environnement ainsi que les relations entre ces composants, pour répondre aux questions « De quoi est-il composé ? » et « Comment est-il organisé ? » ;

3. Le point de vue comportemental, qui consiste à modéliser le produit et son environnement au sein d'une théorie afin de répondre, par la simulation, à la question « Quelles sont ses performances ?».

Une particularité de SysML est de regrouper ces trois familles de représentations au sein d'un unique modèle « multipoints de vue ». L'utilisation d'un modèle unique présente deux avantages notables :

— Cela assure la cohérence des données car les règles de SysML donnent à chaque élément du modèle une définition unique, construite en rassemblant les informations issues de ses différentes représentations, et interdisent à celles-ci de se contredire.

— Cela facilite l'usage intensif de la simulation car un modèle SysML peut regrouper toutes les informations permettant de modéliser le système, de simuler son comportement, et de comparer les résultats aux exigences afin de valider (ou non) des solutions.

— **L'importance des langages semi-formels :** Les langages de modélisation comme UML ou SysML sont très pratiques pour spécifier un système, sont très visuels et faciles à comprendre et agissent comme un langage commun, ils sont fortement recommandés par les normes européennes telles que EN 50128 (applications ferroviaires) ou CEI 61508 (systèmes électroniques de sécurité), pour faciliter les certifications.

  Un travail intéressant est celui de , qui suggèrent l'utilisation de profils UML spécifiques aux standards afin de fournir la preuve de la conformité du système. Ces langages peuvent également offrir des possibilités de modéliser eux-mêmes les exigences, en leur permettant d'être au même niveau que la spécification. Une telle traçabilité est possible avec le diagramme de cas d'utilisation d'UML et de SysML, ou avec le diagramme des exigences de SysML.

  Sur le processus montré dans la figure 2.2 , nous pouvons voir qu'une spécification semi-formelle est essentielle pour tous les acteurs : un client peut suivre l'exécution du projet avec des vues compréhensibles ; elle est la base du travail des ingénieurs et n'exige pas la connaissance de concepts formels, cela permet une meilleure compréhension du système et de sa mise en œuvre auprès des autorités de certification. Pour ces raisons, nous aimerons spécifier des systèmes utilisant un langage de modélisation semi-formel pour obtenir une spécification informelle hors des exigences définies (chemin alternatif de la figure 2.2), en gardant à l'esprit qu'une traduction en notations formelles est l'objectif suivant.

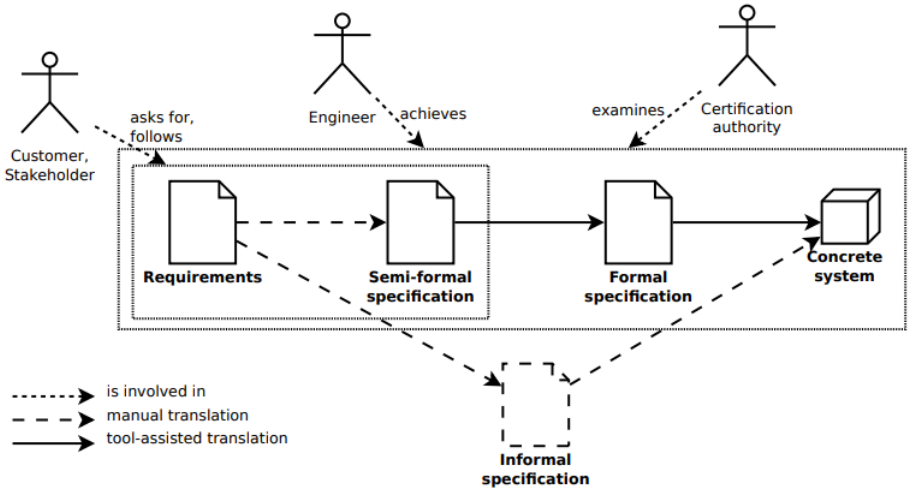

FIGURE 2.2 – Processus de conception de système suggéré en utilisant des méthodes formelles

2.2/ INGÉNIERIE DES SYSTÈMES

Face à la complexité des systèmes d'aujourd'hui, la disposition d'un ensemble d'activités permettant la conception et le développement du système d'une manière structurée devient nécessaire. Cette nécessité s'est traduite par l'apparition d'un nouveau domaine intitulé l'ingénierie système ou ingénierie de systèmes. Cette nouvelle science est le résultat d'un retour d'expérience de la part des grandes entreprises industrielles impliquées dans le développement des systèmes complexes, à savoir Airbus, Thales, Alstom, Altran, EDF, etc. L'Association Française d'Ingénierie Système 1 (AFIS) regroupe ces entreprises parmi d'autres. L'objectif de cette association est de profiter de la synergie des compétences de ces entreprises pour enrichir et améliorer les bonnes pratiques de l'ingénierie système.

L'AFIS définit l'Ingénierie Système comme « une démarche méthodologique générale qui englobe l'ensemble des activités adéquates pour concevoir, faire évoluer et vérifier un système apportant une solution économique et performante aux besoins d'un client tout en satisfaisant l'ensemble des parties prenantes ». Les pratiques de cette démarche sont aujourd'hui répertoriées dans des normes, qui décrivent les pratiques métiers en termes de processus réalisées à l'aide de méthodes supportées par des outils [3].

2.3/ INGÉNIERIE DES EXIGENCES

L'une des disciplines de l'ingénierie système est l'ingénierie des exigences à laquelle se consacre une partie de notre travail. Elle occupe une part importante de l'ingénierie système. Pour mieux appréhender cette démarche, nous devons définir tout d'abord ce qu'est une exigence. Selon [Groupe de Travail Ingénierie Système de l'AFIS, 2009], une exigence prescrit une propriété dont l'obtention est jugée nécessaire. Son énoncé peut

être une fonction, une aptitude, une caractéristique ou une limitation à laquelle doit satisfaire un système, un produit ou un processus ».

Ces exigences peuvent exprimer un besoin en fonctionnalité ou en qualité : Il existe des exigences fonctionnelles et non-fonctionnelles. Les exigences non-fonctionnelles peuvent être exprimées par un ensemble de propriétés, définies en termes de contraintes temporelles, de qualité de service, de sureté de fonctionnement, de sécurité informatique, etc.

La discipline de l'ingénierie des exigences contient deux éléments de base : les exigences et les besoins. Telles qu'elles sont déjà définies, les exigences représentent une vision du système du point de vue des concepteurs. Elles formalisent l'expression des besoins qui représentent une vision du système d'un point de vue utilisateur. Le passage des besoins vers les exigences étant critique, l'enjeu majeur de l'ingénierie des exigences est d'assurer un processus de transformation des besoins exprimés par les clients en exigences systèmes qui seront techniquement exploitables [15]. Concrètement, le découpage du processus d'ingénierie des exigences varie selon les auteurs. Selon [18], ce processus comprend les phases d'extraction, de modélisation, d'analyse de spécification et de validation. D'autres auteurs tels que [5] proposent un découpage en trois phases qui intègre la modélisation dans la phase d'analyse, comparé avec le découpage de Nuseibeh et Easterbrook. Dans notre étude, nous nous intéressons aux phases de modélisation (intégrant la phase d'analyse de spécification) et de validation des exigences et plus précisément l'architecture du système.

2.4/ INGÉNIERIE DIRIGÉE PAR LES MODÈLES

2.4.1/ DU « TOUT EST OBJET » VERS LE « TOUT EST MODÈLE »

Les langages à objets ont vu le jour au début des années 1970 avec le langage SIMULA [4]. Ils sont sortis des laboratoires de recherche au milieu des années 1980, et depuis le début des années 1990 ils ont pris une place prépondérante et sont utilisés à grande échelle dans des domaines de l'informatique.Depuis lors, ces langages orientés objets ainsi que les architectures logicielles et les techniques de conception qui les supportent n'ont pas cessé d'évoluer.

Cette discipline de programmation orientée objet a apporté des réponses à nombreuses questions liées principalement à l'efficacité de la production des logiciels de qualité. Cette discipline met au premier plan la notion d'objet désignant une entité informatique appartenant au monde réel. Il possède deux caractéristiques : un état et un comportement. Il maintient son état via des variables appelées attributs et implémente son comportement à l'aide de méthodes.

Après cette vision basée sur les objets des années 80 et de son principe qui en découle « tout est objet », une nouvelle orientation d'ingénierie, appelée ingénierie dirigée par les modèles (IDM) est apparue avec le principe « tout est modèle ». L'IDM est le résultat de l'évolution du « génie logiciel » dans le but de maitriser la complexité des systèmes informatiques qui ne cessent de s'accroitre. L'IDM place la notion de modèle (plutôt que le code) au centre du cycle du développement et elle propose de modéliser les applications à un haut niveau d'abstraction sans penser à sa technologie cible. Dans ce paradigme, le code source n'est plus considéré comme l'élément central d'un logiciel, mais plutôt comme un élément dérivé des éléments de modélisation.

Le modèle est ainsi défini dans [8] et la définition est traduite dans [12] : Un modèle est une abstraction d'un système, modélise sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé. Il définit aussi l'IDM « comme une forme d'ingénierie générative dans laquelle tout ou partie d'une application est engendrée à partir de modèles ».

L'IDM est considérée dans [2] comme une continuité des travaux précédents de la technologie objet, mais aussi une rupture. En effet, cette technologie des objets est considérée comme un point de départ vers de nouveaux chemins de migrations technologiques telles que l'ingénierie des modèles. D'ailleurs, les solutions proposées par l'IDM sont considérées plutôt complémentaires et non antagonistes avec celles de la technologie des objets du fait que l'IDM ne contredit pas les apports de la programmation par objets.

Toutefois, ce passage peut être considéré aussi comme une rupture du fait des perspectives et des différentes dimensions à considérer de l'une ou de l'autre. Autant dire qu'il est nécessaire de distinguer les relations de base de la technologie des objets, essentiellement héritage et instanciation, de celles de l'ingénierie des modèles, essentiellement représentation et conformité. La figure suivante montre les différentes dimensions permettant le passage de la technologie des objets à l'ingénierie des modèles.

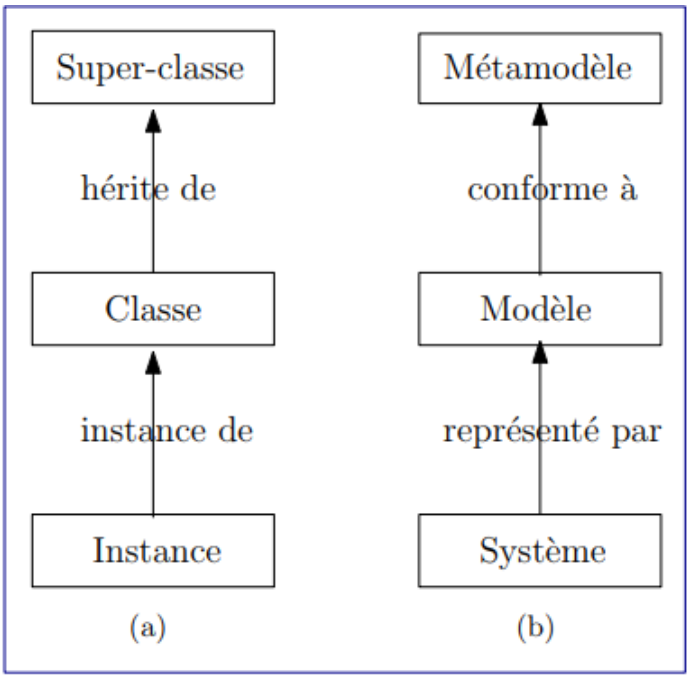

FIGURE 2.3 – Le passage de la technologie des objets à l'ingénierie des modèles

La tendance de la technologie objet des années 80 a mis en avant les relations d'héritage et d'instanciation qui peuvent être exprimées par le fait qu'un objet peut être une instance d'une classe et une classe peut hériter d'une autre classe (Figure 2.3). La nouvelle tendance de l'ingénierie des modèles met en avant le modèle qui représente une vue particulière du système et considère que le modèle doit être écrit conformément au langage de son méta-modèle.

Par conséquent, le passage de la technologie objet à l'ingénierie des modèles est justifié

par l'utilisation de techniques similaires liées à l'abstraction. Le modèle permet l'abstraction de la complexité d'un système en se focalisant sur une préoccupation bien particulière. Cette abstraction fournit non seulement une meilleure maitrise de la complexité des systèmes mais aussi l'augmentation de la productivité et de la réutilisabilité ainsi que l'amélioration de la qualité logicielle.

Un modèle est considéré utile s'il permet d'acquérir une meilleure compréhension du système. Dans un contexte d'ingénierie, un modèle est utile s'il permet de produire les mesures appropriées qui doivent être prises en compte pour atteindre et maintenir l'objectif du système. C'est sur ce principe de base que l'IDM s'appuie fondamentalement pour définir les deux principaux enjeux : la méta-modélisation et la transformation de modèles.

2.4.2/ CONCEPTS DE BASE DE L'IDM

L'objectif majeur de l'IDM est de définir un niveau d'abstraction plus élevé du système et d'automatiser le processus de développement. Il existe trois concepts de base de l'IDM : le modèle, le méta-modèle et le méta-méta-modèle. Le modèle étant déjà défini dans la section précédente, nous nous focalisons sur la définition du méta-modèle, du méta-méta-modèle et de leurs langages.

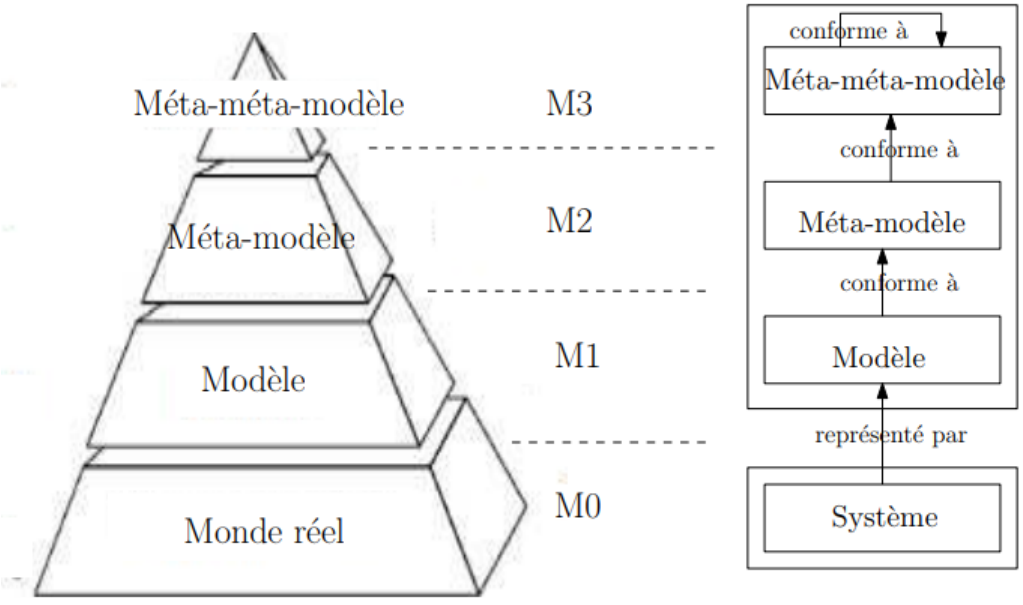

FIGURE 2.4 – Les quatre niveaux de l'architecture de l'IDM

— **Méta-modèle et langage de modélisation :** Pour être en mesure de manipuler des modèles, leur langage doit être spécifié comme un modèle de ces modèles, appelé méta-modèle. D'où, un méta-modèle est un modèle d'un langage de modélisation [22]. La définition de l'OMG 3 (Object Management Group) du méta-modèle est la suivante :
Un modèle est une abstraction d'un système réel tandis qu'un méta-modèle est une abstraction d'ordre supérieur mettant en évidence les concepts utilisés pour définir

le modèle [7]. D'où la relation, un modèle est « conforme à » son méta-modèle. Par analogie aux langages de programmation, un méta-modèle est l'équivalent de la syntaxe d'un langage. La différence est qu'un modèle se concentre sur la syntaxe abstraite alors qu'un programme se concentre sur la syntaxe concrète [12]. En ce qui concerne la sémantique, elle constitue le vocabulaire choisi pour nommer les concepts fondamentaux du domaine défini par la syntaxe abstraite. Bien que les deux concepts de langage et méta modèle soient proches et ils sont souvent confondus dans la littérature, ils sont néanmoins différents [2]. Selon le rapport de synthèse de l'AS CNRS 4 sur le MDA [2] : un langage est un système abstrait alors qu'un méta-modèle est une définition explicite de ce langage. En d'autres termes, le langage joue le rôle de système, tandis que le méta-modèle joue le rôle de modèle de ce langage.

— **Méta-méta-modèle et langage de méta-modélisation :** A l'instar du méta-modèle qui définit le langage de modélisation et qui interprète un modèle, le méta-méta-modèle dispose d'une description du langage dans lequel est écrit le méta-modèle. C'est un modèle pour les méta-modèles [16].

  Dans le but de limiter le nombre de niveaux d'abstraction et de contourner la possibilité de définir un méta-méta méta-modèle et ainsi de suite, ces langages de méta-modélisation sont dotés d'une propriété de méta circularité. C'est la capacité de se décrire lui-même. Le caractère réflexif du méta-méta-modèle est illustré par le niveau M3 dans la figure 2.4 qui représente les quatre niveaux de l'architecture de l'IDM. Cette architecture est également introduite par l'OMG. Le niveau M0 représente le système réel. Au niveau M1, le modèle représente le système. Ce modèle est conforme à son méta-modèle au niveau M2. De même, ce méta-modèle est conforme à son méta-méta-modèle au niveau M3.

  L'OMG a proposé un langage standardisé MOF (Meta Object Facility) pour la définition et la spécification des méta-modèles OMG. Le but de modélisation à travers MOF est de décrire les modèles dans ces différentes couches de la figure 2.4 en utilisant des abstractions de modélisation communes. Cela permet un accès homogène aux modèles dans les différents niveaux (ou différentes couches) grâce à la réflexion. En outre, cela permet de standardiser l'accès aux outils à travers une API (Application Programming Interface) commune et de sérialiser les modèles grâces au standard XMI (XML Metadata Interchange) [22]. Le MOF a contribué de manière significative aux principes fondamentaux de l'architecture dirigée par les modèles MDA de l'OMG.

2.4.3/ L'APPROCHE MDA

L'architecture dirigée par les modèles ou MDA (en anglais Model Driven Architecture) est une approche de l'IDM proposée par l'OMG en 2000 (Figure 2.5). Cette approche vise la promulgation des bonnes pratiques de modélisation en exploitant les avantages de considérer le modèle au centre du développement logiciel. Elle se base sur la séparation des préoccupations entre la logique métier des systèmes informatiques et les plates-formes utilisées. Cette séparation est conçue, dans le but d'éviter de mêler les spécifications fonctionnelles d'un système avec des spécifications techniques de la phase de mise en production sur une plate-forme donnée.

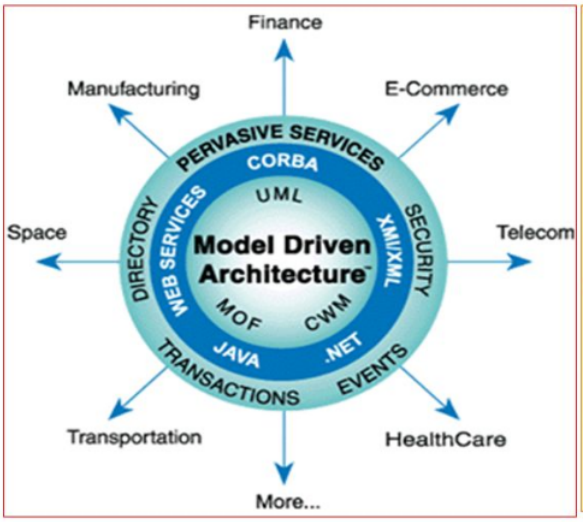

FIGURE 2.5 – MDA par l'OMG

Plusieurs formalismes standards de modélisation sont définis par MDA, notamment UML (Unified Modeling Language), MOF et XMI, afin de promouvoir les qualités intrinsèques des modèles telles que pérennité, productivité et prise en compte des plateformes d'exécution [9]. Le MDA place le modèle au centre du cycle du développement. En effet, en s'appuyant sur le standard UML, le principe clé du MDA consiste en l'utilisation des modèles séparément aux différentes phases de développement d'une application : de la phase d'extraction des exigences jusqu'au codage (Figure 2.6). A cet effet, le MDA préconise l'élaboration des modèles suivants [12] :

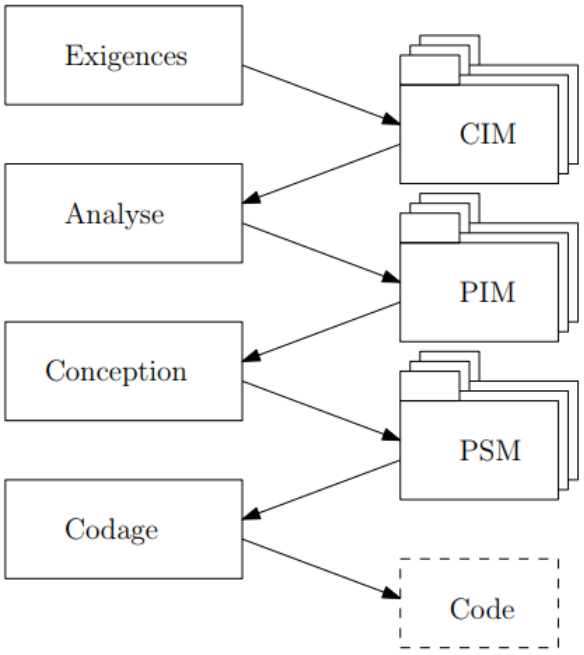

FIGURE 2.6 – Le cycle du développement d'une application en MDA

— **CIM (Computation Independant Model) :** qui est un modèle d'exigences indépendant de l'information.

— **PIM (Platform Independant Model) :** qui est un modèle d'analyse et de conception indépendant de tout détail technique de la plate-forme.

— **PSM (Platform Specific Model) :** qui est un modèle de code, combinant les spécifications du PIM avec les détails propres de la plate-forme.

— **PDM (Platform Description Model) :** qui est un modèle de description de la plate-forme.

Le cycle du développement d'un logiciel selon l'approche MDA prend la forme de « Y » (Figure 2.7). Ce cycle, propre au MDD (Model Driven Development), met l'accent sur les différents niveaux des modèles présentés ci-dessus. Le PSM représente la plate-forme de mise en œuvre permettant de générer un modèle spécifique à partir de PIM. Il peut être raffiné jusqu''a l'obtention d'une implémentation exécutable. Le passage de PIM à PSM nécessite des mécanismes de transformations qui garantissent la pérennité des modèles et leur productivité.

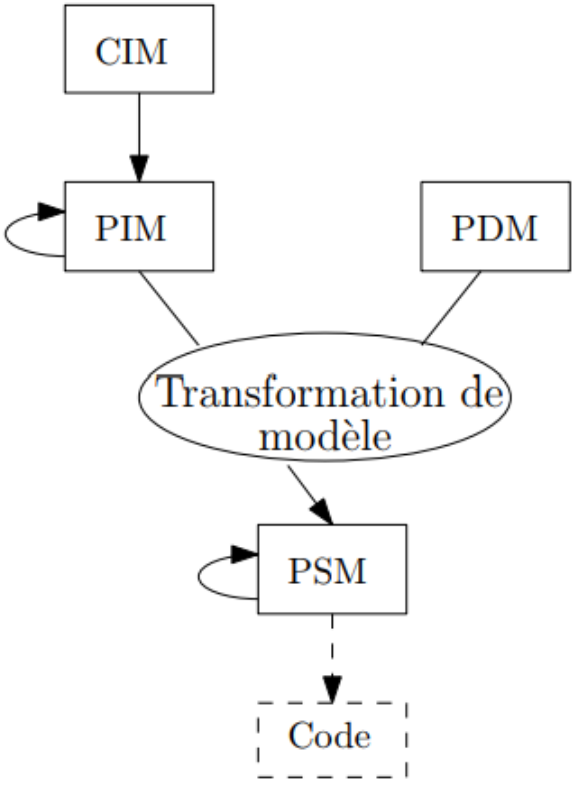

FIGURE 2.7 – processus de développement en Y

2.4.4/ LA TRANSFORMATION DE MODÈLES

Afin de rendre les modèles utilisables et opérationnels, la transformation de modèles est l'une des problématiques clés de l'IDM. Elle se situe au centre de l'approche MDA [12].

Le processus de transformation de modèles en MDA repose sur la génération d'un ou plusieurs modèles cibles à partir d'un ou plusieurs modèles sources. Cette transformation s'effectue par l'intermédiaire de règles qui décrivent la correspondance entre une ou plusieurs constructions du modèle source et une ou plusieurs constructions du modèle cible. Les modèles source et cible de la transformation doivent être conformes à leurs méta-modèles source et cible. Ceci est réalisé par un moteur de transformation (voir figure 2.8).

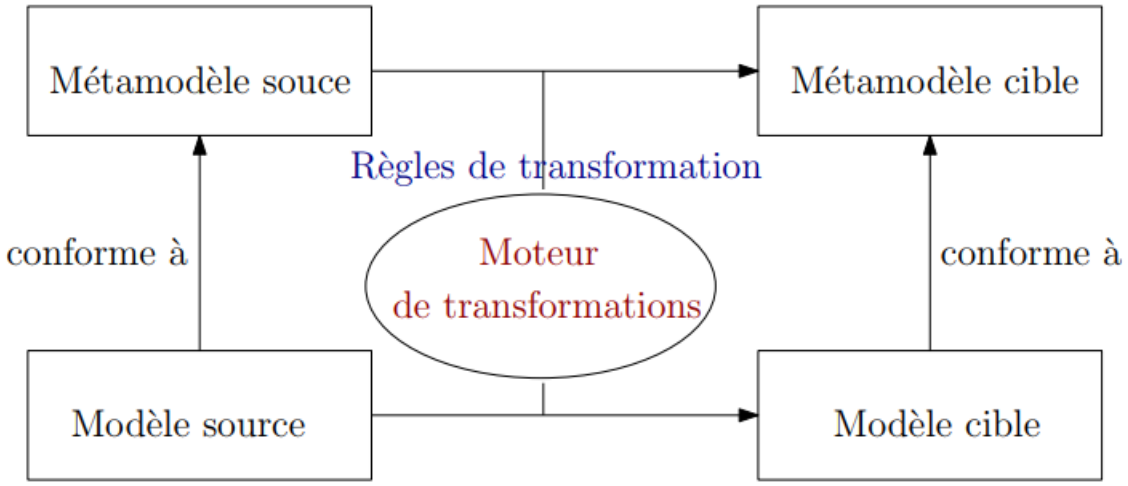

FIGURE 2.8 – Principes de transformation

Selon la nature des méta-modèles source et cible et le changement du niveau d'abstraction, nous pouvons distinguer quatre types de transformations à savoir les transformations endogènes et exogènes et les transformations horizontales et verticales :

— Une transformation est endogène si les modèles source et cible ont le même méta-modèle.
— Une transformation est exogène si les modèles source et cible sont conformes à des méta-modèles différents.
— Une transformation est verticale si le niveau d'abstraction change comme par exemple dans le cas de passage de PIM à PSM et inversement.
— Une transformation est horizontale si le niveau d'abstraction ne change pas comme par exemple dans le cas de transformation de PIM à PIM ou de PSM à PSM.

La figure 2.9, issue de [14] , donne des exemples de la combinaison des différents types de transformations. La restructuration, la normalisation et l'intégration de patrons sont des transformations endogènes horizontales ; tandis que la migration de logiciel et la fusion de modèles sont des transformations exogènes horizontales. Dans le cas de passage de PIM vers PSM pour le raffinement, la transformation est considérée endogène verticale, alors que la génération de code et la rétro-conception sont des exemples de transformation exogène verticale.

Une autre répartition des approches de transformation de modèles peut être prise en compte, celle qui se scinde en approche de modèle-à-modèle (M2M) et approche de

modèle-à-texte (M2T). L'approche M2M permet de transformer des modèles sources, conformes à ses méta-modèles, à d'autres modèles cibles, conformes à d'autres (ou les mêmes) méta-modèles. L'approche M2T est généralement utilisée pour la génération de code. Cette approche peut être considérée comme un cas particulier de l'approche M2M, du moment où on peut fournir un méta-modèle du langage de programmation cible [13].

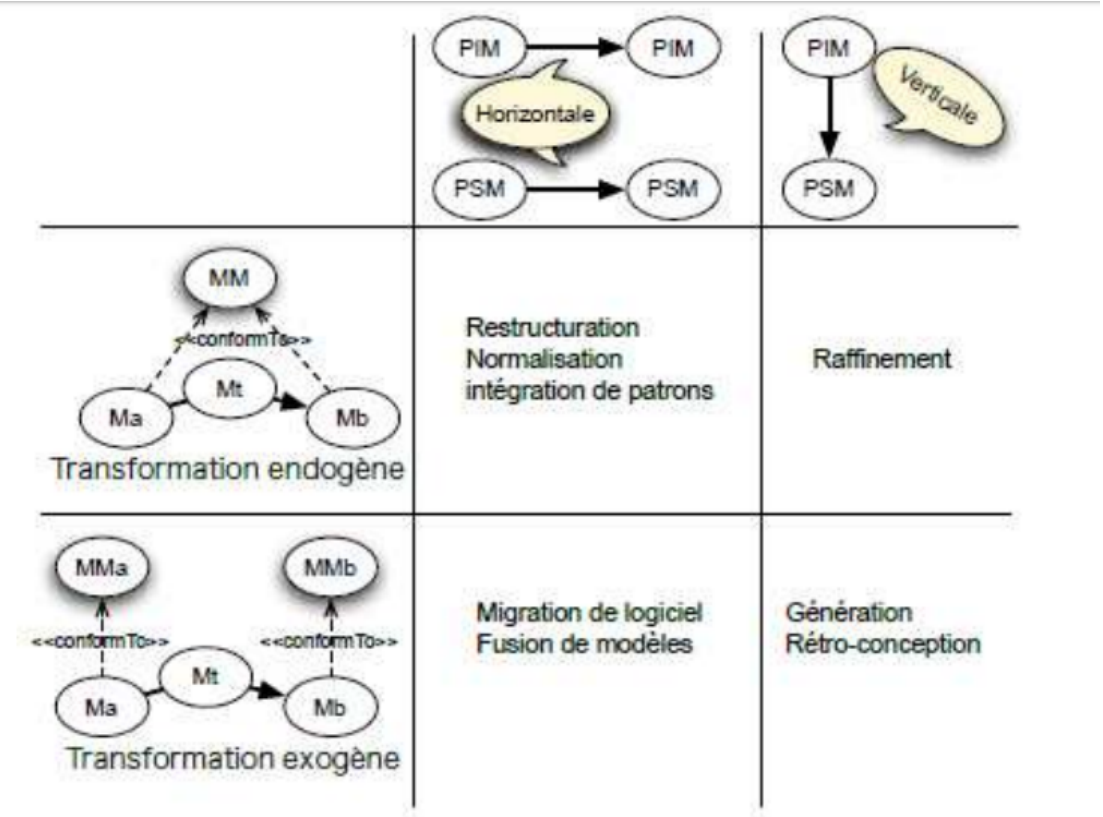

FIGURE 2.9 – Taxonomie de transformation de modèles

2.4.5/ IDM DANS L'INDUSTRIE DES SYSTÈMES CRITIQUES

Les techniques issues de l'IDM font partie intégrante de la panoplie des approches à la disposition des industriels et influencent leurs pratiques en génie logiciel. Nous citons, dans cette section, quelques travaux de recherche qui exploitent l'IDM et les adaptent selon leurs besoins industriels afin de profiter des bonnes pratiques de développement logiciel d'envergure et de la synergie entre les travaux issus des différents domaines.

Dans le secteur aéronautique, les approches de MDA et MDT (Model Driven Testing) ont été intégrées et adaptées dans le processus du modèle en V pour le système de contrôle de trafic aérien [1]. Dans le secteur automobile, le projet VETESS 7 s'inscrit dans le cadre de l'IDM pour la mise en œuvre automatique d'une série de tests de vérification concernant les composants des systèmes embarqués lors de la conception d'un véhicule. Quant au domaine médical, [19] montre l'intérêt de l'utilisation de l'IDM en termes de flexibilité et de modularité pour le développement d'une plate-forme d'échographie en temps réel. Les systèmes nucléaires, comme les autres systèmes critiques, bénéficient

de l'intégration de l'IDM.

Les travaux de thèse de Sannier [21], dans le cadre du projet CONNEXION 8, utilise l'IDM pour la formalisation du domaine par la définition d'un méta-modèle permettant une capitalisation et une vue globale à haut niveau d'un référentiel d'exigences.

2.5/ VALIDATION DES SPÉCIFICATIONS À L'AIDE DES MÉTHODES FORMELLES

La vérification des systèmes critiques tels que les systèmes ferroviaires est très importante. Afin de vérifier les propriétés pour s'assurer que le système fonctionne correctement et ne présente pas de risques pour les êtres humains, il faut d'abord le décrire en utilisant des notations formelles avec une sémantique précise. Les formalismes existants sont de plusieurs formes : langages de programmation exécutables comme LUSTER, graphes comme les structures de Kripke, machines abstraites comme la méthode B, etc. Nous considérons une spécification formelle comme la dernière étape avant un système concret.

En effet, une sémantique précise facilite la traduction directe dans les langages d'implémentation, comme la technique d'affinement de la méthode B. Malheureusement, de telles méthodes nécessitent la connaissance et la compréhension des définitions mathématiques compliquées, ce qui les rend peu appropriées dans des projets industriels en grandeur réelle.

Cet inconvénient est notre principale motivation pour l'utilisation d'une spécification semi-formelle comme passerelle entre des exigences informelles et une spécification formelle.

Il existe au moins deux manières différentes d'effectuer une vérification sur ces descriptions. D'une part, la vérification du modèle est un moyen de vérifier les modèles en explorant leur espace d'état et en testant les propriétés des états énumérés. La principale faille réside dans le problème d'explosion d'état qui apparaît lorsqu'on considère des systèmes réels. D'un autre côté, les techniques de preuve de théorèmes - base de la méthode B - consistent à des démonstrations mathématiques de propriétés. Ils ne souffrent pas de problèmes d'explosion d'état, et des outils existent pour les exécuter automatiquement. Cependant, lors de la modélisation du système, il est difficile de trouver les protections correctes, les invariants et les preuves correspondantes, de sorte que les propriétés correctes soient vérifiées.

2.6/ POURQUOI SYSML ET LA MÉTHODE B ?

Nous nous concentrons sur un problème précis dans cette étude : être capables de traduire les exigences en spécifications d'un système complet utilisant SysML, et être capable de traduire ce modèle SysML en un modèle de méthode B afin que la mise en œuvre de logiciels sûrs soit possible. Au lieu de considérer des approches utilisant des outils / langages similaires, nous nous concentrons sur les méthodes SysML et B susmentionnées pour les raisons suivantes :

SysML est l'un des langages semi-formels les plus polyvalents disponibles. Premièrement, il prend en charge la modélisation d'un large éventail de systèmes et

de sous-systèmes : matériel, logiciel, informations, processus, etc. En revanche, UML se concentre uniquement sur l'ingénierie logicielle.

Deuxièmement, le diagramme des exigences permet d'exprimer les exigences au même niveau que l'ensemble de la spécification du système, ce qui offre des perspectives de traçabilité très intéressantes, et le diagramme de définition de bloc pour la modélisation de la structure du système [10].

De nombreuses façons d'utiliser SysML avec des méthodes formelles ont été étudiées : [6] suggèrent l'utilisation de réseaux de Petri pour définir le comportement des diagrammes SysML, afin de vérifier les propriétés à l'aide du vérificateur de modèle introduisent un langage graphique basé sur des diagrammes paramétriques SysML, qui peuvent être lus par le vérificateur de modèle UPPAAL.

Ces méthodes peuvent vérifier les propriétés qui ne peuvent pas être vérifiées avec la méthode B, telles que les propriétés dépendant du temps. Néanmoins, la méthode B est basée sur des techniques de preuve de théorème et possède une syntaxe similaire aux langages de programmation lorsqu'elle atteint la phase de mise en œuvre.

En termes de préoccupations industrielles, ces outils semblent clairement appropriés. D'une part, SysML peut spécifier une partie importante des systèmes. D'un autre côté, la méthode B a été utilisée avec succès dans l'industrie des systèmes critiques. Nous citons le travail de [17] qui consiste à combiner des méthodes d'ingénierie des exigences avec des méthodes formelles. son idée est d'étendre le langage SysML avec des concepts de méthodes d'ingénierie existantes par une définition des extensions de SysML avec des concepts du modèle de but de la méthode KAOS et donner des règles pour dériver une spécification B formelle de ce modèle de but.

Le travail de [9] , qui a comme objectif la détection des problèmes critiques le plus tôt possible. Il a proposé d'identifier un sous-ensemble vérifiable de SysML qui soit utilisable par les ingénieurs système, tout en se prêtant à une transformation automatique vers des outils de vérification formels. En considérant la méthode B dans le but de prouver des propriétés de sécurité exprimées en utilisant des invariants sur les états. Cette approche consiste en un alignement des concepts SysML, OCL(Object Constraint Language) et Alf(Action Language for Foundational UML) avec un sous-ensemble identifié de la méthode B, en utilisant des similarités sémantiques entre les langages. Il a défini un SysML restreint étendu par un profil léger et une transformation vers la méthode B à des fins de V et V du modèle SysML.

2.7/ BILAN GLOBAL

Nous avons abordé, dans notre étude de l'art, l'ingénierie des exigences faisant partie de l'ingénierie système. Nous nous focalisons sur les phases de modélisation et des exigences. Afin de profiter des concepts de modélisation, de méta-modélisation et de transformation de modèles, l'ingénierie dirigée par les modèles vient compléter l'ingénierie des exigences. En effet, les approches basées sur les modèles sont vues comme des méthodologies à forte valeur ajoutées permettant de définir un niveau d'abstraction plus élève du système et d'automatiser le processus de développement. Nous avons présenté les concepts de base de l'IDM pour l'industrie des systèmes critiques d'une façon générale.

Dans notre travail de recherche, nous nous basons sur la vérification des exigences liées à l'architecture des systèmes complexes. Dans l'optique de mener à bien la phase de spécification et les activités de validation et de vérification, nous optons pour une approche de couplage de notations semi-formelles et formelles SysML/B afin de tirer profit de chacune d'entre elles.

3

LANGAGES ET OUTILS DE MODÉLISATION UTILISÉS

3.1/ SYSML

Une extension du langage de UML normalisée par l'OMG. C'est l'un des principaux langages de modélisation de systèmes embarqués. Il focalise essentiellement sur l'architecture logicielle et matérielle et n'aborde pas les aspects liés à l'exécution ni aux aspects temporels [11].

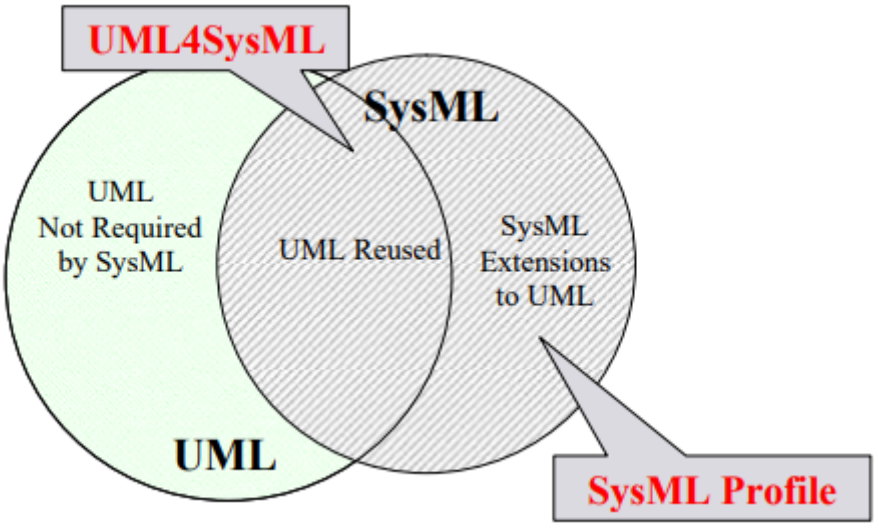

FIGURE 3.1 – Relation entre UML et SysML

L'ingénierie des systèmes est intéressée par les différents côtés des systèmes complexes, que ce soit le logiciel ou les côtés matériels. Cependant, le développement d'un système, qui se caractérise par un ordre de complexité, n'est toujours pas évident s'il n'existe pas d'outils appropriés qui assistent et guident son développement. C'est à cause de ce besoin que le langage SysML a vu le jour comme un langage de communication entre les différents membres des équipes de développement. Il a permis d'unifier les

principes de modélisation visuelle en utilisant un petit ensemble de diagrammes, ce qui le rend facile à apprendre et à utiliser.

L'introduction de SysML dans ce domaine a non seulement permis de simplifier la modélisation et la communication, mais aussi d'offrir au développement communautaire une base solide pour répondre aux exigences des systèmes depuis les premières étapes du développement à travers un processus piloté par un modèle.

3.1.1/ PRINCIPES DE SYSML

Le développement de SysML a été principalement guidé par ces principes :

— **Parcimonie :** Les bases SysML sont sur une partie d'UML. Cette partie est considérée comme le sous-ensemble minimal de diagrammes UML qui permet de satisfaire les exigences de la communauté d'ingénierie système. Les autres éléments nécessaires ont été ajoutés en fonction des nouveaux besoins exprimés par le domaine de l'ingénierie des systèmes.

— **Réutilisation :** SysML réutilise les concepts d'UML. Cependant, les exigences supplémentaires ont été satisfaites en ajoutant de nouveaux concepts. Cette extension des concepts SysML a toujours été guidée par le principe de la parcimonie.

— **Stratification :** Ce principe est utilisé pour organiser le profil SysML de deux manières. La première repose sur le fait que SysML est défini comme un profil UML strict. Par conséquent, tous les paquets SysML sont considérés comme une couche d'extension du méta-modèle UML. Cependant, la seconde manière concerne uniquement les constructions SysML, où elles sont organisées en deux niveaux de conformité, Basic et Advanced, ce qui constitue une superposition supplémentaire.

— **Extensibilité :** SysML prend en charge les mêmes mécanismes d'extension fournis par UML (méta classes, stéréotypes, bibliothèques de modèles), ce qui permet d'étendre le langage à des domaines d'ingénierie système spécifiques tels que l'automobile, l'aérospatiale, la fabrication et les communications.

— **Interopérabilité :** SysML est aligné sur la sémantique de la norme d'échange de données ISO pour prendre en charge l'interopérabilité entre les outils d'ingénierie. Il hérite de l'échange XMI d'UML ce qui rend possible l'utilisation des fichiers de modèles générés d'un outil par plusieurs autres outils.

3.1.2/ LES DIAGRAMMES SYSML

SysML a neuf diagrammes, où quatre diagrammes (paquet, cas d'utilisation, séquence et diagrammes de machine d'état) sont directement copiés à partir d'UML 2.0, trois diagrammes (activité, définition de bloc et diagrammes de blocs internes) sont copiés avec quelques modifications. Les deux derniers diagrammes sont considérés comme nouveaux (diagrammes paramétriques et d'exigences). Une autre taxonomie décompose les diagrammes SysML sur trois sous-ensembles (voir Figure 3.2). Il différencie les diagrammes structurels, comportementaux et d'interrogation.

Dans ce qui suit, nous donnons la description de chaque sous-ensemble, avec une définition de chaque diagramme. Nous nous concentrons davantage sur les diagrammes que nous utiliserons dans notre travail.

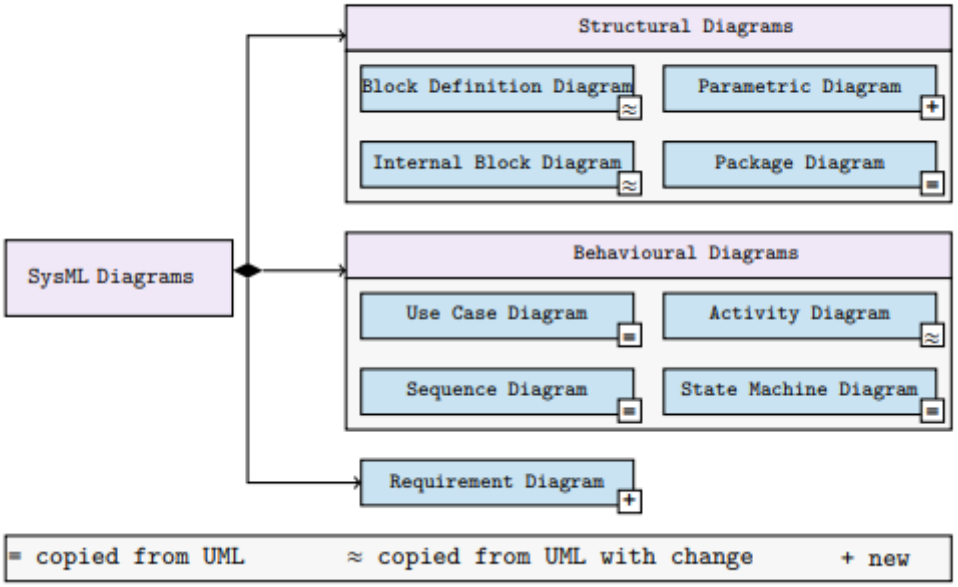

FIGURE 3.2 – Les diagrammes SysML

SysML utilise certains diagrammes UML et propose aussi des extensions. Une des extensions proposées concerne la prise en compte des exigences. Plusieurs concepts sont proposés pour la représentation des exigences (fonctionnelles ou non-fonctionnelles) et la traçabilité avec d'autres éléments de spécification. Malgré ses apports, l'ensemble des concepts proposés dans SysML n'est pas aussi riche que dans les autres méthodes d'ingénierie des exigences : les exigences sont spécifiées de manière informelle sous forme de texte ce qui entraîne des ambiguïtés .

Sur les 9 diagrammes SysML, nous n'utilisons ici que le diagramme d'exigences et le diagramme de définition de bloc (modélisation de structure).

3.1.2.1/ DIAGRAMME DE DÉFINITION DE BLOC

Le diagramme de définition de bloc (BDD), tel qu'il est capturé dans la figure 3.3, est un diagramme copié à partir d'UML avec quelques modifications. Il repose sur le diagramme de classes UML, avec l'exclusion de certaines capacités, telles que certaines formes d'associations spécialisées. Sur d'autre part, il a modifié certains concepts tels que la notion de classe a été remplacée par la notion de bloc. Il a également ajouté de nouveaux concepts tels que les ports de blocs. Il peut se référer à une partie matérielle ainsi qu'à une partie logicielle, il peut aussi représenter une personne qui utilise ou interagit avec ce système.

Chaque bloc a un nom, un ensemble de valeurs, des propriétés, des blocs référencés, des parties, des opérations qui ont des précoditions et des postcoditions qui sont liées a des exigences dans notre cas.

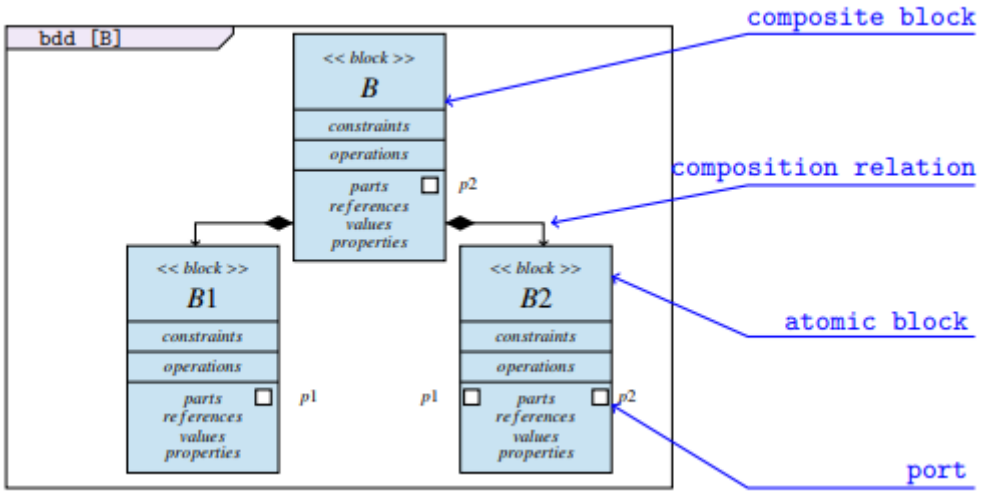

FIGURE 3.3 – diagramme de définition de blocs

3.1.2.2/ DIAGRAMME DES EXIGENCES

Le diagramme d'exigences (RD) est un nouveau diagramme dans SysML. Il spécifie les exigences système attendues par les utilisateurs. Ce diagramme offre un moyen de modéliser les exigences fonctionnelles et non-fonctionnelles et les différents liens entre elles.

Il est possible de représenter la hiérarchie existante entre les exigences en utilisant les relations de composition et de dérivation («deriveReqt»). Il est également possible de lier un bloc à une exigence en utilisant la relation de satisfaction («satisfy»), ou d'associer une exigence à un cas de test en utilisant une relation de vérification («verify»). Chaque exigence est définie par son nom, sa description et son propre identifiant unique.

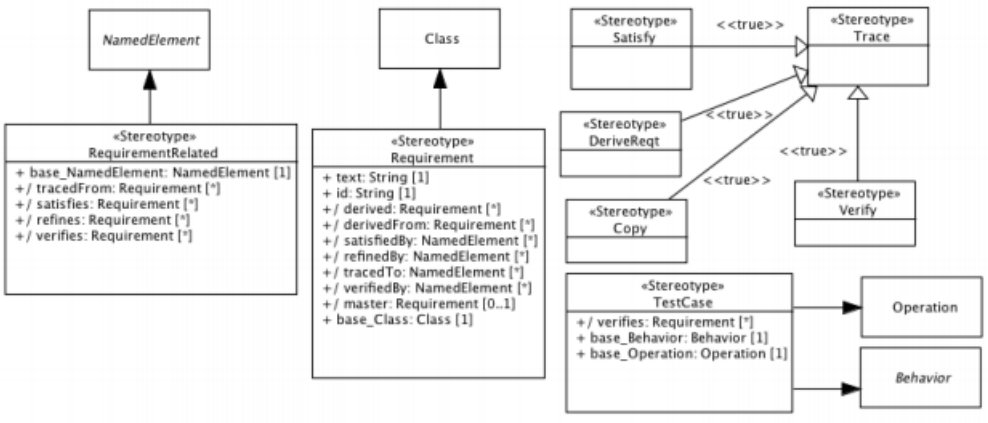

FIGURE 3.4 – Métamodéle des exigences SysML de Papyrus

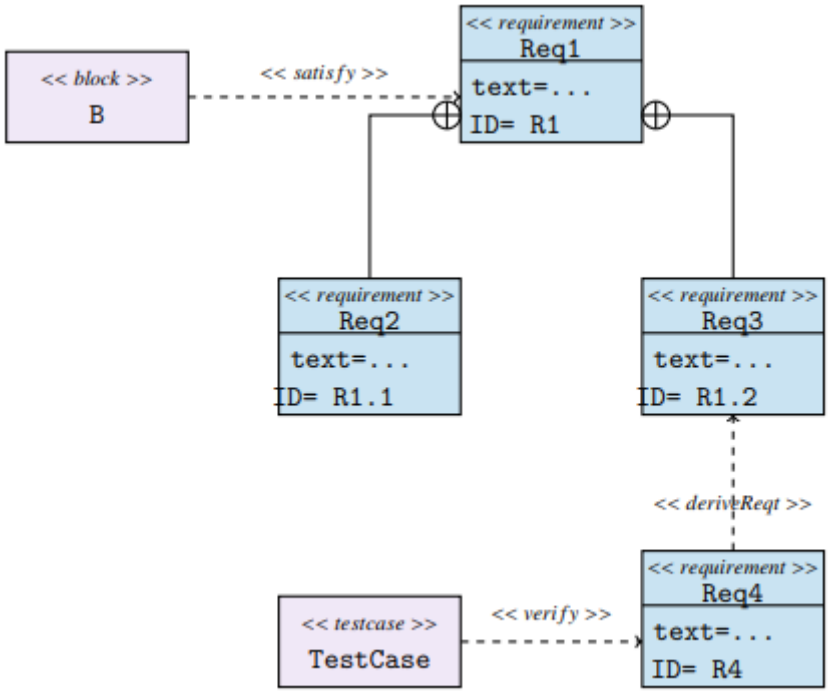

FIGURE 3.5 – les élements basiques du le diagramme d'exigences

3.2/ MÉTHODE B

La méthode B est une méthode qui couvre toutes les phases de développement formel. En effet, un cycle de développement classique se compose généralement de phases de spécification, de conception, d'implémentation et de test appuyées éventuellement par des activités de vérification et de validation. La méthode B apporte des modifications sur ce cycle par la formalisation d'un modèle abstrait aboutissant à la production de code exécutable par une suite de raffinements prouvés (voir figure 3.6). Le passage d'une phase à une autre dans un processus de développement en B se fait par raffinement et implique la démonstration d'un ensemble d'obligations de preuve dans le but de validation.

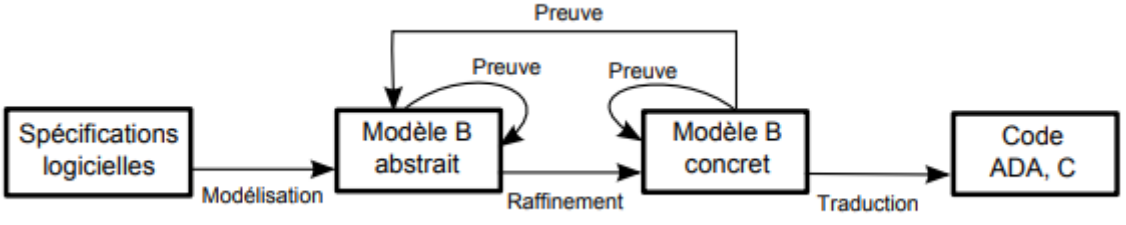

FIGURE 3.6 – Développement logiciel avec la méthode B

La méthode B a été aussi utilisée dans d'autres objectifs tels que la certification de propriétés ou de protocoles. Dans ce qui suit, nous présentons l'historique industriel de la méthode B et nous définissons ses fondements conceptuels et ses notations. Ensuite, nous poursuivons par la notion du raffinement et de la modularité en B. Nous achevons cette vue générale par la présentation de quelques outils de B utilisés pour réaliser notre travail.

3.2.1/ HISTORIQUE

Plusieurs projets de métros automatiques sans conducteur, réalisés par Siemens, ont été développés sans méthode formelle tels que le VAL à Lille et MAGGALY (Métro à Grand Gabarit de l'Agglomération LYonnaise).

Suite à ces projets, des méthodes formelles ont été introduites pour réaliser le SACEM (Systèmes d'Aide à la Conduite, à l'Exploitation et à la Maintenance), un automatisme ferroviaire destiné à la RATP. Au milieu des années 80, J.-R. Abrial a conçu la méthode B, en s'inspirant des méthodes formelles VDM et Z. Comme VDM et Z, la méthode B fait partie des méthodes formelles orientées modèle. Elle a été utilisée industriellement avec le projet Météor, la ligne 14 du métro parisien entièrement automatique sans conducteur mise en service en 1998.

Ce projet a été développé par Matra Transport Internationa (maintenant Siemens) pour la RATP. Le pilotage de ce système a nécessité l'utilisation des logiciels de sécurité développés formellement avec la méthode B dont la preuve a permis de supprimer les tests unitaires et a donné un résultat remarquable.

Comparé au projet SACEM, l'utilisation de la méthode B dans ce projet s'est effectuée plus en amont d'es la phase de développement des logiciels de sécurité. En effet, comme la preuve de programme valide le code par rapport à la conception détaillée, il devrait de même pour la conception détaillée par rapport à la spécification. Dès lors, la méthode B a été généralisée à l'ensemble des logiciels pour les équipements SACEM.

3.2.2/ FONDEMENTS ET NOTATIONS

Trois notations de base constituent les piliers du langage de modélisation de la méthode B, à savoir : la notation mathématique, la notation des machines abstraites et la notation des substitutions généralisées.

3.2.2.1/ NOTATIONS MATHÉMATIQUES

Les fondements mathématiques de la méthode B sont issus de la logique du premier ordre et de la théorie des ensembles. Le langage B est un langage de données relevant de la théorie des ensembles permettant de modéliser les données et les propriétés des données. La méthode repose, toutefois, sur une théorie des ensembles typée. En effet, les éléments des ensembles en langage B ont tous la même structure fondamentale. D'ailleurs, un ensemble ne peut pas contenir, par exemple, des entiers et des couples d'entiers à la fois. Cette notion de typage repose sur la notion d'ensemble et la propriété de monotonie de l'inclusion.

3.2.2.2/ LES MACHINES ABSTRAITES

La machine abstraite est la notion de base de la méthode B. Cette notion est similaire à des notions de programmation connues sous le nom d'un module, d'une classe ou d'un type de donnée abstrait. De façon générale, la machine abstraite se scinde en trois parties :

```
MACHINE
  PARTIE ENTÊTE
    Nom de la machine abstraite
    Paramètres de la machine
    Contraintes sur les paramètres
  PARTIE STATIQUE
    Déclaration d'ensembles
    Déclaration de constantes
    Propriétés des constantes
    Déclaration des variables (état)
    Invariant (caractérisation de l'état)
  PARTIE DYNAMIQUE
    Initialisation de l'état
    Opérations sur l'état
END
```

FIGURE 3.7 – Structure d'une machine abstraite B

— La partie entête sert à identifier la machine abstraite : le nom de la machine dans la clause **MACHINE**, ses éventuels paramètres et la description de ses paramètres à l'aide de prédicats de typage dans la clause **CONSTRAINTS**.

— La partie statique contient la déclaration de la liste des ensembles abstraits et la définition des ensembles énumérés dans la clause **SETS**, ainsi que la déclaration des constantes dans la clause **CONSTANTS** et des variables dans la clause **VARIABLES**. Ces déclarations sont complétées par la définition du type et des propriétés de ces constantes et de ces variables respectivement dans les clauses **PROPERTIES** et **INVARIANT**. Les propriétés de la clause **INVARIANT** doivent être toujours satisfaites par l'état de la machine.

— La partie dynamique définit l'évolution de l'état de la machine à travers une initialisation dans la clause **INITIALISATION** et les opérations de la machine dans la clause **OPERATIONS**.

  La clause **OPERATIONS** permet de déclarer les services offerts par une machine et de spécifier leur comportement en utilisant la notion de substituions généralisées. Les opérations décrites permettent, en effet, de modifier les données de la machine tout en préservant les propriétés des variables de la clause **INVARIANT**.

3.2.2.3/ NOTION DE SUBSTITUTION GÉNÉRALISÉE

La notation de substitutions généralisées a pour but de modéliser un ensemble de services qui définit le mécanisme d'évolution des données de la machine. Les substitutions sont des notations mathématiques permettant de modéliser la transformation de prédicats. En effet, elles produisent, à partir des prédicats qui définissent les données avant l'activation d'un service, les prédicats qui d´définissent les données après l'accomplissement de ce service. Les substitutions généralisées sont des extensions de la substitution élémentaire.

Les principales substitutions généralisées sont :

— La substitution avec Pré-condition, notée PRE P THEN S END, qui d´définit les conditions d'utilisation d'une autre substitution ;

— La substitution avec garde, notée SELECT P THEN S END, qui d´définit le contexte d'application d'une autre substitution ;

— La substitution de choix non-déterministe borné, notée CHOICE S1 OR S2 END, qui d´définit deux substitutions possibles ;

— La substitution de choix non-déterministe non-borné, notée ANY X WHERE P THEN S END, qui d´définit un nombre indéterminé de substitutions possibles, une pour chaque valeur des variables X qui satisfait au prédicat P ;

— La composition parallèle de substitutions, notée S1 —— S2, qui effectue simultanément deux substitutions ;

— La substitution identité, notée SKIP, qui laisse inchangés les prédicats ;

— La substitution conditionnelle d´déterministe, notée IF C THEN S1 ELSE S2 END ;

— La composition séquentielle de substitutions, notée S1 ; S2 ;

— et, finalement la substitution d'itération, notée WHILE C DO S INVARIANT I VARIANT V END ;

3.2.2.4/ LE RAFFINEMENT

La méthode B propose une construction par approximations successives, appelée raffinement. La notion de raffinement est fondamentale en B. D'une part, elle permet d'introduire les détails du problème, qui n'étaient pas pris en compte auparavant, dans le processus de développement formel. Par conséquent, la traduction du problème initial est effectuée d'une façon progressive et non pas en une seule fois.

D'autre part, le raffinement est également utilisé pour raffiner les machines abstraites afin d'obtenir un code exécutable. Le passage de la spécification abstraite vers le code nécessite des preuves de raffinement dans le but de s'assurer que le code final d'une machine satisfait la spécification initiale. Le dernier raffinement est désigné par implantation.

Sur le plan pratique, le raffinement d'une machine abstraite est mené par :

1. la suppression des éléments non-exécutables du pseudo-code des opérations de la machine abstraite (les Pré-conditions et les choix (choice))

2. l'introduction des structures de contrôle classique de la programmation (le séquencement et les boucles (loop)), et

3. la transformation des structures de données mathématiques (ensembles, relations, fonctions, séquences et arbres) en des structures programmables (des variables simples, des tableaux ou des fichiers). Les opérations de la machine abstraite restent les mêmes dans le raffinement, mais leurs pseudocodes seront modifiés selon le premier et le deuxième point.

3.2.2.5/ LA MODULARITÉ EN B

En partant du principe que la réalisation d'un système complexe ne peut s'accomplir en une seule fois, la méthode B propose une construction par plusieurs niveaux de raffinement et une décomposition en modules. La décomposition en modules est une caractéristique importante de la méthode B car une architecture modulaire atténue la complexité perçue pour un système. Un module B permet de modéliser un sous-système faisant partie d'un projet B. Les modules sont constitués par des composants B. Un composant peut-être une machine abstraite, un raffinement ou une implantation. En fonction des trois propriétés suivantes du module :

1. il comprend toujours une machine abstraite,
2. il peut posséder une implantation et éventuellement des raffinements et
3. il peut posséder un code associé, nous pouvons distinguer trois sortes de modules

— des modules développés par raffinements successifs d'une machine abstraite qui satisfont les trois propriétés en ayant un code associé par traduction,

— des modules de base qui ne satisfont pas la deuxième propriété mais satisfont la troisième en ayant un code associé manuellement, et

— des modules abstraits qui ne satisfont ni la deuxième ni la troisième propriété. Afin d'assurer la cohérence mathématique de ces composants (modules) et la correction d'un raffinement vis-à-vis de ses versions plus abstraites, les obligations de preuve sont ainsi générées par la méthode. La sous-section suivante décrit les obligations de preuve prescrites par la méthode.

3.2.2.6/ LES OBLIGATIONS DE PREUVE

Une obligation de preuve est une formule mathématique a démontrer afin d'assurer qu'un composant B est correct. La théorie B indique quelles sont les obligations de preuve a démontrer pour assurer la correction d'un composant B donné. Dans cette optique, les obligations de preuve sont une aide au processus de vérification.

Les obligations de preuve sont générées pour les machines abstraites, les raffinements et les implantations. Nous citons, dans ce qui suit, les obligations de preuve qui concernent la correction de l'initialisation et la correction des opérations pour les machines abstraites et les raffinements.

Les obligations de preuve relatives à une machine abstraite concernent, entre autres :

— La correction de l'initialisation : l'initialisation doit établir l'invariant de la machine (l'invariant de la machine doit être vrai après application de la substitution de l'initialisation) ;

— La correction des opérations : les opérations doivent préserver l'invariant (l'invariant de la machine doit être vrai après application de la substitution de l'opération, sachant que l'invariant était vrai avant) ; les opérations doivent établir leur post condition. Dans le cas du raffinement, la correction de l'initialisation et des opérations fait intervenir l'initialisation/l'opération abstraite afin de montrer la pertinence du raffinement. Les obligations de preuve relatives à un raffinement concernent, entre autres :

— La correction de l'initialisation : l'initialisation doit établir l'invariant du raffinement (propriétés des nouvelles variables et invariant de liaison) sans contredire l'initialisation spécifiée ;

— La correction des opérations : les opérations doivent préserver l'invariant du raffinement (propriétés des nouvelles variables et invariant de liaison) sans contredire l'opération abstraite spécifiée.

3.3/ OUTILS

3.3.1/ PAPYRUS POUR LA MODÉLISATION

Papyrus est un ensemble de plugin eclipse appartenant au projet de modélisation Eclipse. Il vise à offrir un environnement intégré dédié aux utilisateurs. Ainsi, Papyrus est un nouvel environnement pour éditer toutes sortes de modèles EMF (Eclipse Modeling Framework), et particulièrement pour supporter UML et ses profils tels que SysML, MARTE, etc. Papyrus propose des éditeurs de diagrammes de langages de modélisation EMF. Il offre également la possibilité de relier ses éditeurs et autres outils. Sa génération de fichiers XMI, permet l'utilisation de ses modèles avec d'autres applications et outils. Papyrus est un outil graphique mais aussi textuel. Ainsi, il est possible de modifier les modèles en utilisant des éditeurs textuels qui offrent l'aide pour éditer le contenu du modèle. Papyrus a notamment été utilisé avec succès dans des projets industriels et constitue la base de plusieurs outils de modélisation industrielle. développé initialement au sein du CEA-List en France et issu du projetPolarSys de la fondation Eclipse. L'outil supporte les métalangages UML 2.5 et SysML 1.4, et autorise la création de langages de modélisation spécifiques à une industrie ou à un domaine applicatif.

3.3.2/ ACCELEO POUR LA TRANSFORMATION DES MODÈLES

Acceleo est un langage basé sur modèle pour générer du code. Ce langage permet la génération de modèle à texte en permettant à l'utilisateur d'écrire des modèles configurables pour chaque élément d'un modèle d'entrée. Acceleo est une implémentation du langage MOF modèle-texte défini par l'OMG. Acceleo est également un langage de transformation mais la cible est du texte au lieu d'un autre modèle. En définissant des modèles pour chaque composant du méta-modèle d'entrée, Acceleo génère un ensemble de fichiers conformes à la grammaire cible.

Acceleo offre une transformation beaucoup plus simple qu'ATL. Il ne peut pas ajouter de comportement s'il n'est pas spécifié par le modèle d'entrée. Cela ne peut fonctionner qu'avec les concepts de la grammaire cible. Le modèle d'entrée doit être écrit dans la même langue que le langage textuel cible, il doit donc être conforme à un métamodèle qui

représente le langage textuel cible. Dans certains cas, si le modèle d'entrée se conforme à un langage proche du langage textuel cible, une adaptation doit être effectuée. Prenez par exemple un générateur de code UML vers Java. UML n'est pas censé être une version graphique de Java. Il a l'intention d'être plus générique. En raison de son expressivité générique, certains éléments UML ne peuvent pas être traduits directement en Java. Les classes d'association UML n'ont par exemple pas d'équivalent côté Java. Ils peuvent être traduits en d'autres concepts java qui imitent la fonctionnalité attendue des classes d'association UML, comme une classe complète et deux associations.

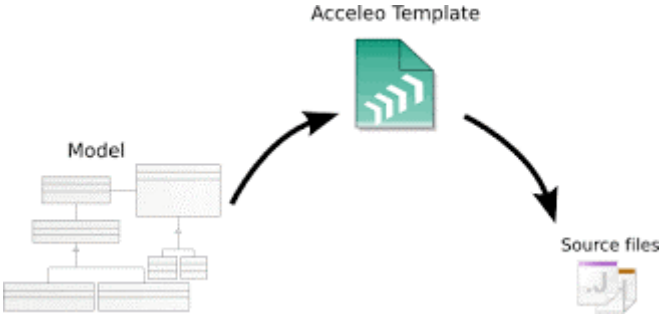

FIGURE 3.8 – Fonctionement d'Acceleo

3.3.3/ ATELIER B POUR LA VÉRIFICATION

L'outil de développement relatif à la méthode B utilisé est l'Atelier B. Il couvre toutes les étapes de développement d'un logiciel, à savoir la spécification (grâce à un modèle abstrait), les étapes de raffinement pour obtenir un modèle concret ainsi que la génération de code. Toutes les obligations de preuve pour vérifier la correction sont générées automatiquement. L'Atelier B offre un prouveur automatique, ainsi qu'un environnement de preuve interactif utilisé dans le cas où le prouveur automatique échoue.

L'Atelier B est un outil, développé par ClearSy, qui implante la méthode B. Il permet notamment de spécifier un modèle, de le raffiner, de générer les obligations de preuve (et de les démontrer) puis de générer du code exécutable conforme au modèle. Les ingénieurs de Siemens IC-MOL l'utilisent pour développer les applications sécuritaires des métros automatiques.

4

VERS UNE APPROCHE COMBINANT SYSML ET B POUR LA VÉRIFICATION DES SYSTÈMES

L'ingénierie des systèmes est un domaine d'ingénierie interdisciplinaire qui permet d'avoir une vision globale d'un problème et de sa solution. Le modèle de domaine opérationnel est un élément central de toute approche fondée sur un modèle et décrit le système dans le contexte de son environnement. Cela inclut les humains destinés à fonctionner et à interagir avec le système, les objets externes susceptibles d'affecter le système et les éléments environnementaux susceptibles d'avoir un impact sur le système. Le modèle du domaine opérationnel est un point de départ utile pour avoir une vue d'ensemble d'un système et de son fonctionnement.

Papyrus fournit une gamme de fonctionnalités qui aident l'ingénieur à construire un modèle de domaine opérationnel, y compris la définition de bloc SysML standard et les diagrammes de blocs. Les éléments peuvent également être hyper-liés, permettant au spectateur d'utiliser un diagramme comme une rampe de lancement pour des modèles et des diagrammes plus détaillés. Le modèle de domaine opérationnel SysML définit l'environnement d'exploitation du système, qui décrit les conditions d'exploitation dans lesquelles le système doit fonctionner.

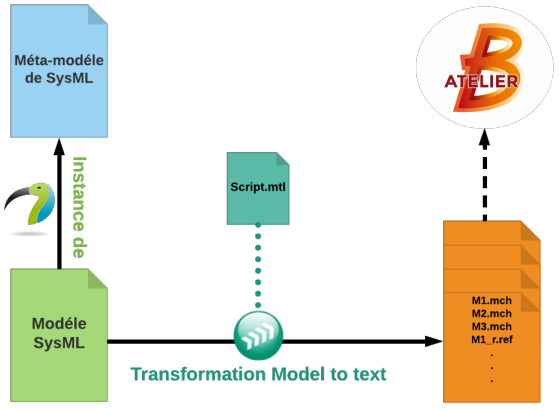

FIGURE 4.1 – Processus de réalisation du projet B vérifiée à partir du modèle SysML

L'objectif de proposer et d'introduire SysML dans le domaine de l'ingénierie système est de mettre à la disposition des concepteurs un langage permettant de reprendre les milliers de lignes de documents (documents textuels utilisés pour décrire les exigences, l'architecture et le comportement du système) en utilisant ensemble de modèles conviviaux et graphiques. Ainsi, SysML a été créé pour être un langage de haut niveau qui offre aux concepteurs une flexibilité de conception suffisante.

Cependant, cette flexibilité et cet aspect non-formel de SysML ont contribué à l'émergence de certaines ambiguïtés. Ces ambiguïtés peuvent être résumées dans les différentes interprétations de la sémantique de ses concepts et diagrammes (par exemple, il n'y a pas d'indication au niveau des diagrammes SysML).

Le processus de développement logiciel d'un système critique en génie logiciel se base généralement sur des documents informels qui soit les cahiers de charge contenant des exigences en langage naturel. Et comme le langage naturel n'est pas précis pour la formalisation et la gestion des exigences par la suite, il est nécessaire alors d'interpréter et formaliser ces spécifications mathématiquement afin de les valider et les vérifier au début du cycle du développement.

La phase d'extraction des exigences a pour vocation de spécifier d'une manière claire, précise et non- ambiguë les caractéristiques du système à développer. Ce constat illustre la nécessité de passer des notations informelles aux notations semi-formelles ou encore formelles en tirant profit des avantages de chacune d'elles. Notre travail est de développer un plugin qui fait une transformation d'un modèle d'entrée SysML en machines et raffinements B vérifiables, tout en modélisant les exigences liées à l'architecture du système et transformer ce modèle en texte dans un template Acceleo pour générer les machines et les raffinements en langage B.

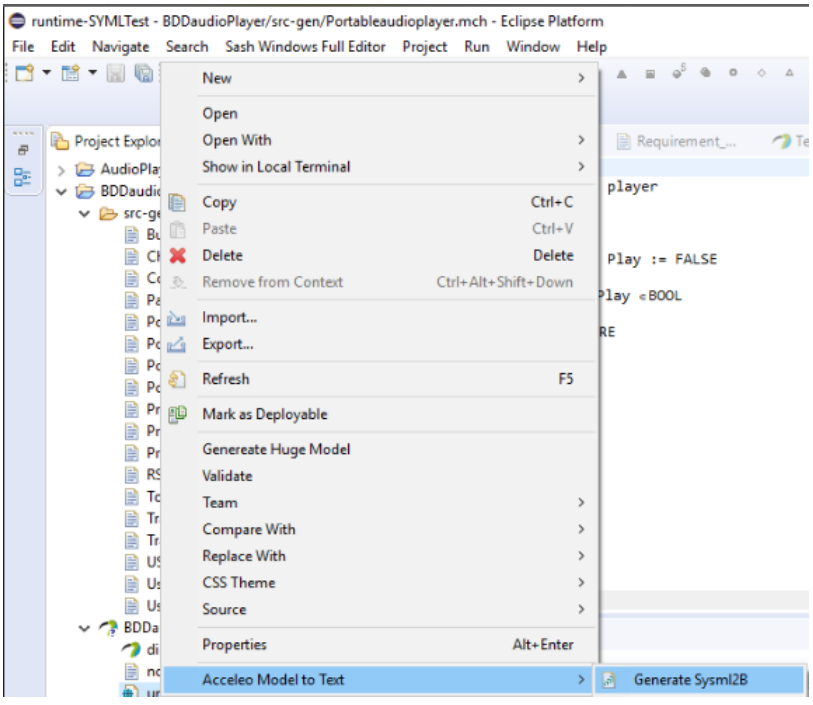

FIGURE 4.2 – Plugin SysML2B

Afin de rendre la sémantique des diagrammes SysML que nous utiliserons plus précise et plus claire, nous proposons une approche constructive dans laquelle des modèles formels abstraits exprimés en B comme les machines et les raffinements qui sont construits progressivement à partir du modèle formalisé contenant des exigences et des blocs ainsi que des relations entre ces deux derniers. Nous complétons les machines et les raffinements obtenus en les enrichissant avec l'invariant les post-conditions et les pré-conditions retirées des exigences. Pour cela, nous étendons SysML en utilisant le mécanisme de profilage. La figure 4.2 montre le plugin que nous avons développé dans le contexte de notre travail de recherche.

La figure suivante décrit les étapes de la réalisation de notre plugin « SysML2B » par un diagramme d'état montrant les différentes phases de l'extraction des données à leur vérification formelle.

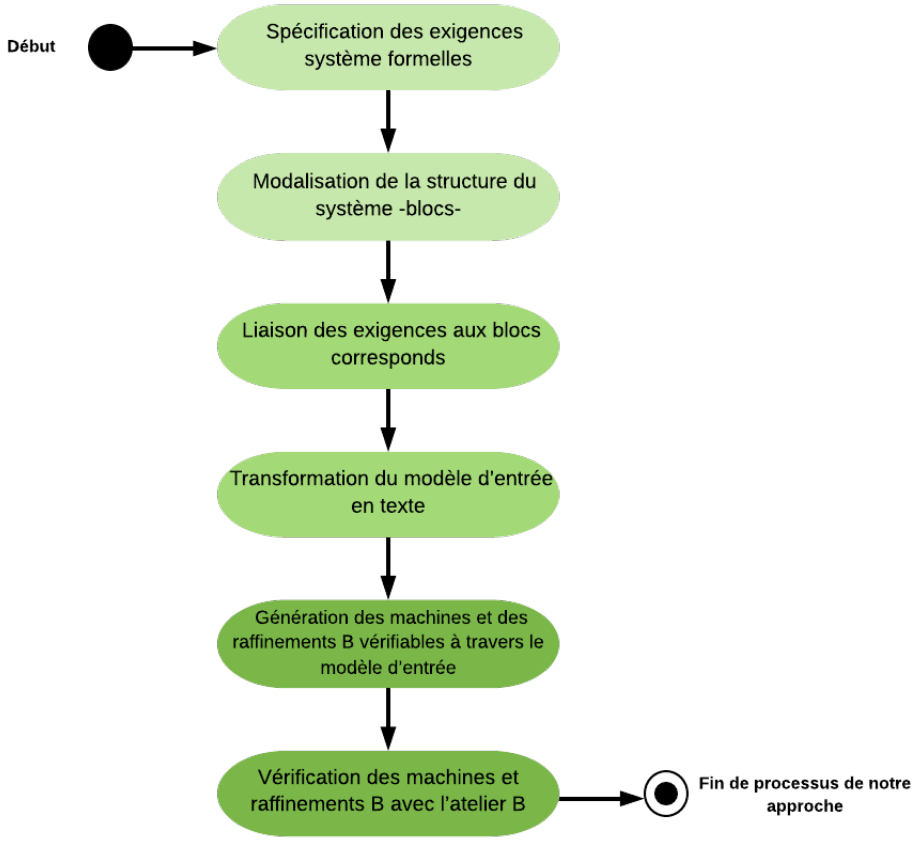

FIGURE 4.3 – Les étapes de réalisation du plugin « SysML2B »

4.1/ EXTRACTION DE DONNÉES À PARTIR D'UN MODÈLE SYSML

La démarche de notre approche commence par une analyse des besoins à partir du cahier de charges qui engendre les spécifications fonctionnelles. Ces spécifications seront l'entrée pour la phase de conception qui consiste à façonner et formaliser le système

en répondant à l'ensemble des exigences du système. Cette phase permet une formalisation/modélisation semi-formelle en SysML et formelle en B. Le modèle semi-formel qui répond aux spécifications fonctionnelles de départ sera enrichi par les exigences des spécifications formelles du modèle formel B. Cependant, nous ajoutons à l'aide de la notion de profilage un nouveau champ à l'exigence déjà définie en SysML (Bclause) pour exprimer les propriétés en langage B.

Dans ce contexte , nous avons créé des profils UML combinant les opérateurs du langage B entre exigences afin d'exprimer les propriétés formelles au niveau des exigences, la figure 4.4 montre notre premier profil contenant les opérateurs logiques en B.

Nous avons proposé de décomposer une exigence en deux types : une exigence élémentaire et une composite qui aussi contient des exigences élémentaires ou composites. Nous avons aussi défini les opérations B comme des relations logiques entre exigences, des relations de contenance qui sont décomposées en relations de composition parallèle ou séquentielle et de choix inclusif ou exclusif, et de relations de dérivation qui est l'opérateur logique : implique.

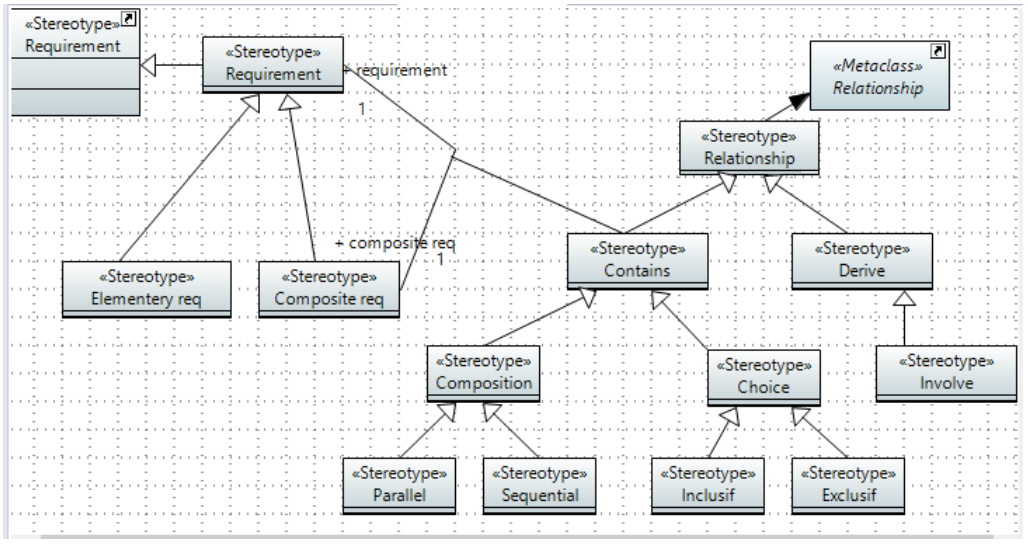

FIGURE 4.4 – Première version du profil SysML

Mais nous avons abandonné ce profil car il n'était riche au point d'englober tout un système qui contient des blocs et des exigences. Dans une autre part, afin de d'améliorer l'expression formelle, nous avons ajouté le stéréotype **CONSTRAINT** aux propriétés des exigences pour exprimer l'expression en B entièrement dans la contrainte, nous avons étendu le stéréotype **CONSTRAINT** en un **Bconstraint** et exprimer la propriété B à l'intérieur. En papyrus, les contraintes sont exprimées en OCL (Object Constraint Language), et il est possible de changer le langage d'expression en langage naturel. Donc nous avons pensé d'écrire les propriétés B mais il est impossible d'exprimer la relation exigence – bloc directement avec les propriétés B.

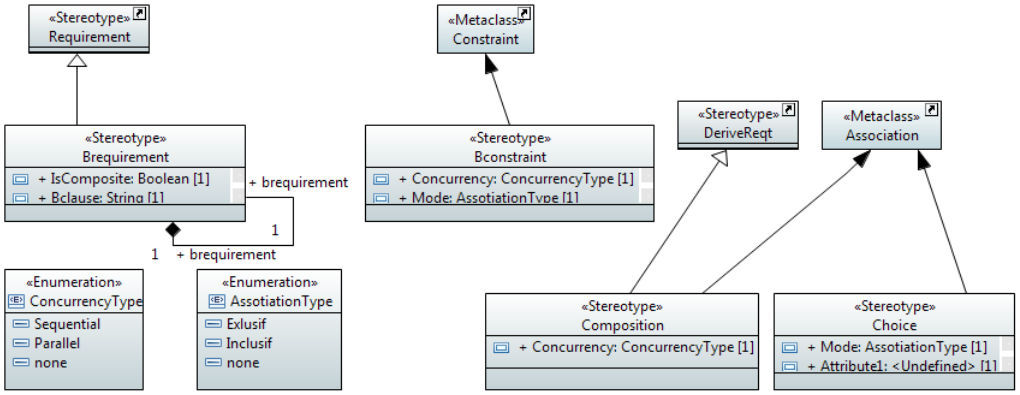

FIGURE 4.5 – Deuxième version du profil SysML

C'est pourquoi nous travaillons directement sur l'exigence par une simple extension des champs d'une exigence, la figure 4.5 montre notre choix final à notre besoin de modéliser les exigences liées à l'architecture du système modélisé.

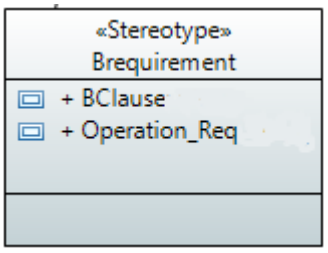

FIGURE 4.6 – stéréotype du Brequirement

4.1.1/ DIAGRAMME DES EXIGENCES

Nous avons créé un nouveau stéréotype Brequirement qui nous permet de prendre une Clause B en considération dans le modèle SysML, cette exigence peut être prise comme un invariant, une Pré-condition ou une Post-condition.

Dans le cas où l'exigence est Pré-condition ou Post-condition on précise l'opération du bloc à laquelle est liée cette exigence.

4.1.2/ DIAGRAMME DE DÉFINITION BLOCS

Un bloc SysML contient des variables que nous utiliserons après dans la machine B associée au bloc, des opérations qui ont des liens (Pré-condition ou Post-condition) avec les exigences et la relation Satisfy qui se transforme en invariant en B.

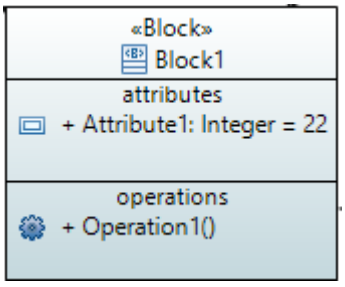

FIGURE 4.7 – Bloc SysmL

4.1.3/ RELATIONS ENTRE EXIGENCES ET BLOCS

— **Satisfy :** un ou plusieurs éléments du modèle (blocs) permettent de satisfaire une exigence dans ce cas l'exigence est prise comme invariant de la machine associée au bloc.

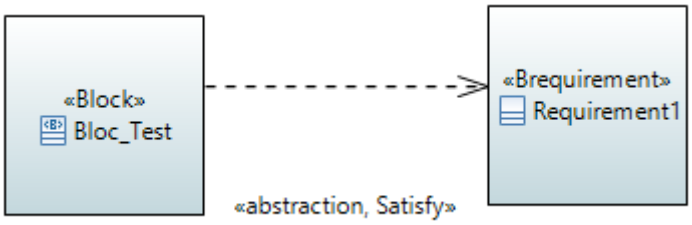

FIGURE 4.8 – Relation de satisfaction entre blocs et exigences

— **Pré-condition :** nous avons ajouté cette relation qui permet de lier une exigence à une opération de bloc pour préciser que cette exigence est une Pré-condition de l'opération mentionnée dans la propriété OperationReq de l'exigence sélectionnée comme opération de ce bloc et qui doit être toujours vraie avant l'exécution d'une opération.

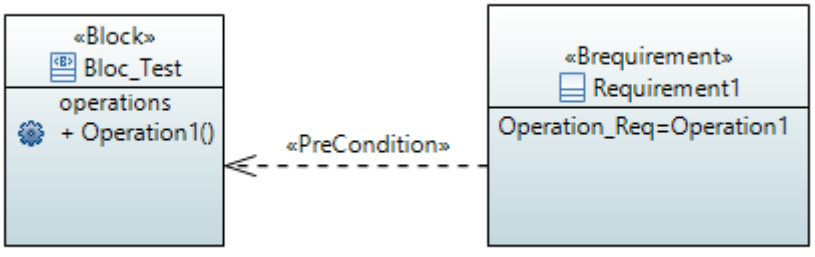

FIGURE 4.9 – Relation de pré-condition entre blocs et exigences

— **Post-condition :** cette relation permettre de lier une exigence à une opération

de bloc pour préciser que cette exigence est une Post-condition de l'opération mentionné dans la propriété OperationReq de l'exigence sélectionnée comme opération de ce bloc et qui doit être toujours vraie après l'exécution d'une opération.

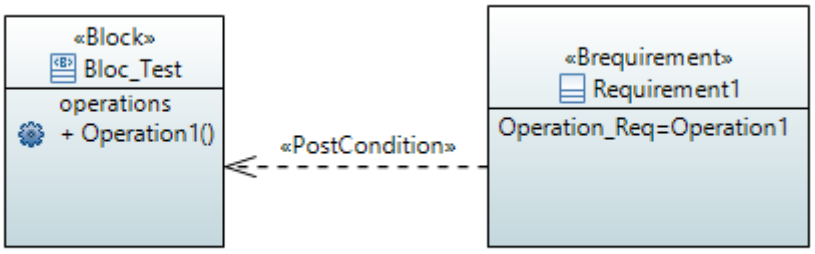

FIGURE 4.10 – Relation de post-condition entre blocs et exigences

4.2/ TRANSFORMATION DES DONNÉES

Cette phase consiste à transformer notre modèle d'entrée en code source en utilisant Acceleo dans un Template contenant toutes les données reliées au modèle, dans notre cas le Template contient le méta modèle importé qui est SysML et le modèle transformé, la figure 4.11 montre le modèle d'entrée vers Acceleo.

FIGURE 4.11 – Traitement de données du modèle dans un template Acceleo

Une fois le template est créé, nous allons l'adapter pour répondre à nos besoins de spécifications, dans notre cas, la création des machines B à partir des blocs ainsi que des raffinements à partir des relations de composition , les deux tableaux suivants montrent les similutudes sémantiques entre les deux modèles ainsi que quellques règles de passages de SysML vers B [10].

Citation de ClearSy « Manuel du Langage B »	Citation de la spécification « OMG SysML»
"Un développement complet en B correspond à un projet B. Un projet permet formellement un système de n'importe quel type. [. . .] Un projet B fait référence à un ensemble complet d'instances de module B. Les composants de ces instances de module sont connectés par des liens. "	"Les blocs fournissent une capacité polyvalente pour modéliser des systèmes en tant qu'arbres de composants modulaires. Les types spécifiques de composants, les types de connexions entre eux, et la façon dont ces éléments se combinent pour définir le système global peuvent tous être sélectionnés en fonction des objectifs d'un modèle de système particulier "
"Un module B modélise un sous-système; il fait partie d'un projet B. "	"Un bloc est une unité modulaire qui décrit la structure d'un système ou d'un élément. "

FIGURE 4.12 – Similitudes sémantiques entre les projets de méthode B et les blocs SysML pour modéliser la structure

Les concepts en B	Les concepts en SysML
Projet	Ensemble de blocs qui ont des relations avec des exigences
Lien Imports	Part Property
Type basique	Value Type
Donnée basique	Valeur de la propriété
Constante « isReadOnly »	Meta-attribute
Ennumerated set	Enumeration
Operation	Operation (avec comportement)
Component Parametre	Valeur par defaut

FIGURE 4.13 – Liste des similitudes sémantiques que nous considérons

4.2.1/ CRÉATION DES MACHINES

Quelques ajouts à notre sous-ensemble de SysML : Même avec un sous-ensemble de la méthode B et un sous-ensemble de SysML qui sont sémantiquement similaires, nous ne sommes pas encore complètement satisfaits pour deux raisons. Premièrement, nous n'avons pas trouvé de contreparties SysML pour certains concepts essentiels de

la méthode B : la notion de module principal d'un projet, la possibilité de définir des éléments abstraits, et la prise en compte des invariants définis dans la mise en œuvre d'un module. Deuxièmement, nous voulons être en mesure de concevoir des systèmes réactifs d'une manière pratique.

Pour surmonter la première raison, nous utilisons le mécanisme de profil UML pour définir une extension très simple au langage SysML. Tout d'abord, nous ajoutons la propriété BClause au nouveau stéréotype Brequirement. Puis, nous avons défini nos règles SysML personnalisées écrites en langage naturel qui limitent l'utilisation de SysML.

4.2.2/ CRÉATION DES RAFFINEMENTS

Nous avons ajouté la règle règle suivante : *Si un bloc est composé de plusieurs blocs : on génère un raffinement du bloc principale composé en incluant les machines associées au blocs composants avec la clause* ***"INCLUDES"***

La clause INCLUDES permet de regrouper dans un raffinement les variables, les constantes, les ensembles avec leurs propriétés provenant d'autres machines abstraites. La clause INCLUDES permet donc de décomposer une machine abstraite complexe en plusieurs machines abstraites, tout en facilitant le travail de preuve associé.

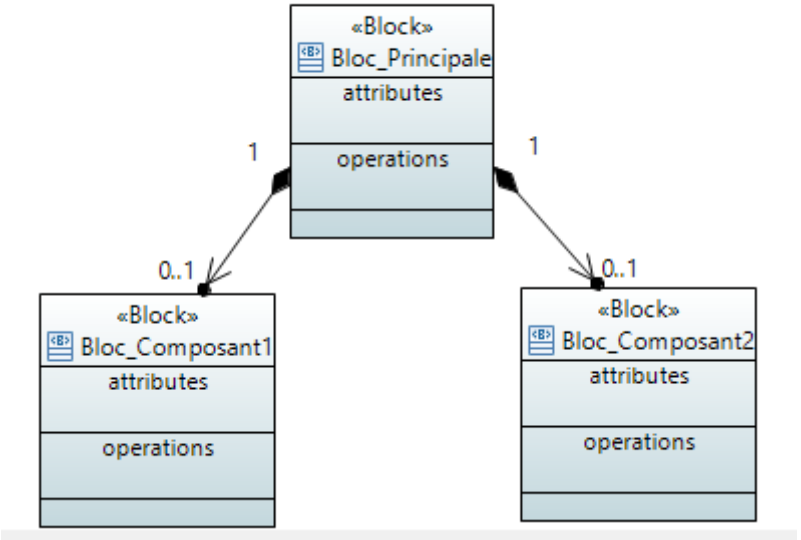

FIGURE 4.14 – Relation de composition entre blocs pour créer les raffinement

En effet, la preuve d'un composant et de ses machines "incluses" est globalement plus simple que la preuve du composant équivalent non décomposé. L'interprétation d'une inclusion est simple : les variables et constantes de la machine incluse deviennent des entités de la machine inclue, et les opérations incluses deviennent des "morceaux de spécification" utilisables.

Portableaudioplayer_i.ref*

```
1  REFINEMENT Portableaudioplayer_i
2  REFINES Portableaudioplayer
3  INCLUDES PowerSubsystem
4- VARIABLES
5  Pause, Play
6- INITIALISATION
7  Play,Pause := FALSE, TRUE
8- INVARIANT
9  Pause ∈ BOOL ∧  Play ∈ BOOL
10- OPERATIONS
11- PlayList = PRE
12  Pause = TRUE
13- THEN
14  Pause := TRUE
15  END
16  END
```

FIGURE 4.15 – Exemple de raffinement généré

4.3/ VÉRIFICATION DES MACHINES ET RAFFINEMENTS AVEC ATELIER B

Dès que les machines et les raffinements sont créés dans des fichiers.mch seront prêts à être vérifiés par l'atelier B qui va concrètement valider les propriétés exprimées dans les exigences et reliées avec des blocs du système. Le résultat est un ensemble de machines et de raffinements vérifiables. La figure montre le résultat de la vérification où on remarque qu'il y a des obligations de preuve qui ont été vérifiées (”OK”).

Composant	Typage vérifié	OPs générées	Obligations de Preuve	Prouvé	Non-prouvé	B0 Vérifié
Portableaudioplayer	-	-	-	-	-	-
Portableaudioplayer_i	-	-	-	-	-	-
PowerSubsystem	OK	OK	0	0	0	OK

FIGURE 4.16 – machines et raffinements vérifiés ”OK” dans ateliers B

4.4/ EXPÉRIMENTATION

Au cours de ce document, nous illustrerons les étapes de notre processus en présentant les différents diagrammes d'un exemple de projet. Notre exemple (développé par Sam Mancarella) un système matériel/logiciel que la plupart des gens connaissent - un lecteur audio-. Nous choisissons le diagramme de définition de bloc SysML qui représente tous les blocs du système avec les liens entre ces blocs pour montrer la structure de notre système. C'est un exemple de modèle dans le domaine opérationnel, le lecteur audio portable est une composition de divers sous-systèmes prêts à l'emploi réutilisables qui exécutent des tâches spécifiques : la conception comprend des sous-systèmes d'alimentation, de lecture et de traitement audio et d'interfaçage avec d'autres dispositifs et l'interface utilisateur [20] (Figure 4.17).

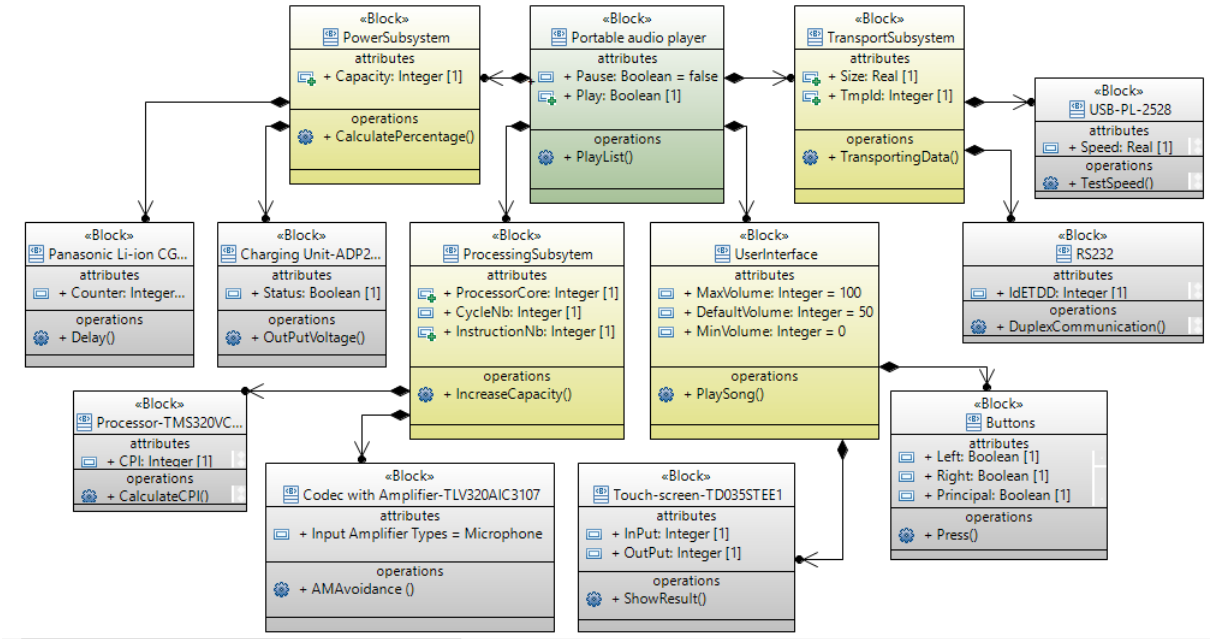

FIGURE 4.17 – Diagramme de définition de blocs du AudioPlayer

4.4.1/ LIAISON DES EXIGENCES AUX BLOCS

Nous lions ce diagramme avec des exigences de telle façon qu'une exigence soit satisfaite par un ou plusieurs blocs et que les exigences peuvent former des Pré-conditions ou des post conditions des opérations des blocs comme c'est montré sur la figure 4.18.

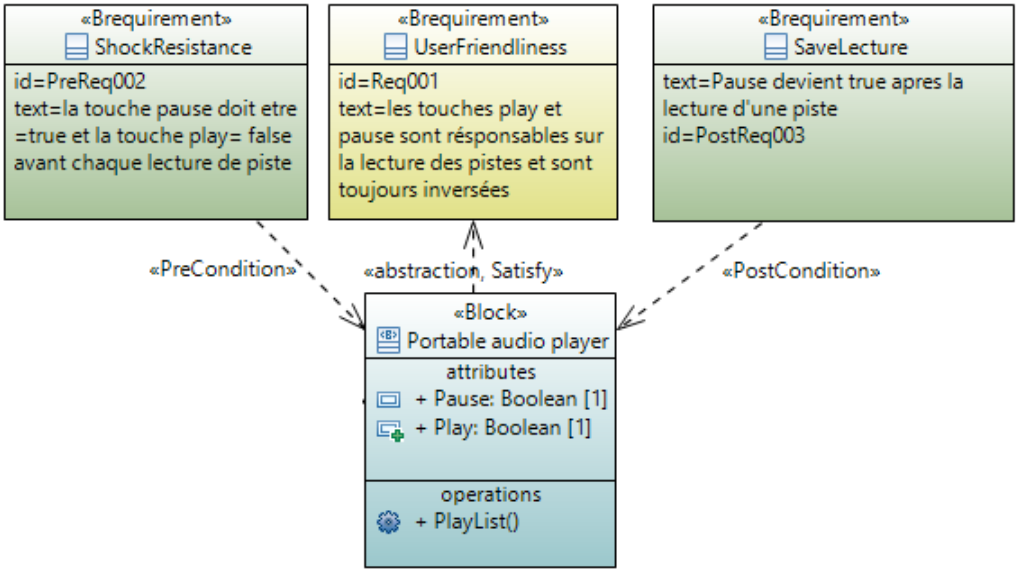

FIGURE 4.18 – liaison du bloc AudioPlayer avec les exigences

Les exigences sont définies avec un texte non formel et une clause en langage B, cette clause contient des propriétés mathématiques vérifiables avec l'atelier B.

4.4.2/ GÉNÉRATION DES MACHINES ET DES RAFFINEMENTS B

La figure 4.19 montre la machine B générée Portableaudioplayer.mch qui correspond au bloc audioPlayer modélisé dans le diagramme de définition de blocs , la machine contient toutes les méthodes et tous les attributs du bloc, et c'est pareil pour tous les blocs du diagramme.

Portableaudioplayer.mch

```
1 - MACHINE
2   Portableaudioplayer
3 - VARIABLES
4   Pause, Play
5 - INITIALISATION
6   Play,Pause := FALSE, TRUE
7 - INVARIANT
8   Pause ∈ BOOL ∧  Play ∈ BOOL
9 - OPERATIONS
10 - PlayList = PRE
11   Pause = TRUE
12 - THEN
13   Pause := TRUE
14   END
15   END
16
```

FIGURE 4.19 – Machine B générée a partir du bloc portableaudioplayer

La figure 4.20 illustre un sous ensemble de diagramme de définition de blocs en se basant sur la relation de composition entre blocs. Cette relation génère un raffinement portableaudioplayer.ref à partir de ces blocs reliés.

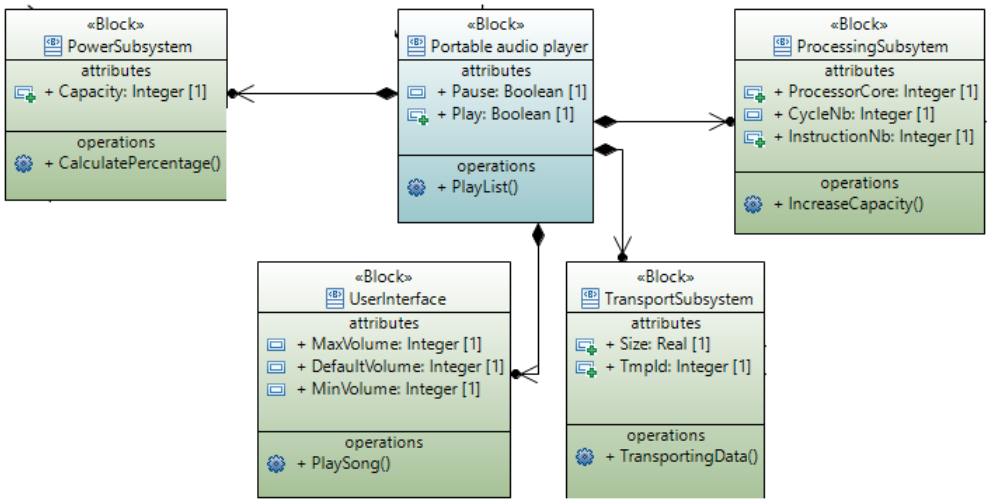

FIGURE 4.20 – Machine B générée a partir du bloc portableaudioplayer

Ci-dessous un aperçu sur le dossier du projet B généré qui contient toutes les machines et raffinements générés associés à un modèle d'entrée SysML.

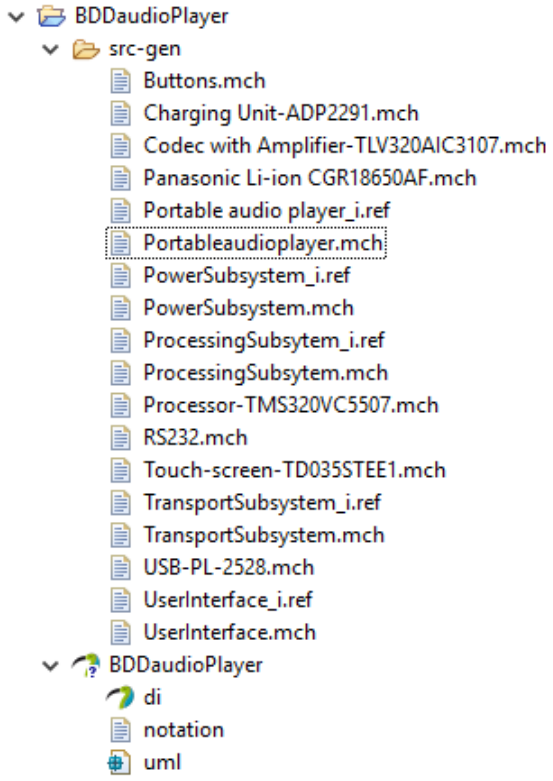

FIGURE 4.21 – Projet B avec toutes les machines et les raffinements

Notre plugin a une particularité de transformer les blocs et les relations rapidement et créer plusieurs fichiers avec une simple clique sur SysML2B avec une transparence totale des étapes intermédiaires pour l'utilisateur.

5

CONCLUSION GÉNÉRALE

5.1/ CONCLUSION

Le développement basé sur les composants se concentre sur la décomposition du système en composants fonctionnels individuels qui représentent des interfaces de communication bien définies.

Le but principal de ce projet est la proposition d'une approche pour faciliter la combinaison du semi-formelle au formelle pour construire des systèmes complexes où leur structure et leur comportement sont modélisés à l'aide de diagrammes SysML (diagramme de définition de blocs et le diagramme des exigences). Lors de l'assemblage de ces blocs, nous devons prendre en considération la dépendance des exigences que nous voulons satisfaire en réutilisant et en composant ces blocs.

Dans notre projet, nous avons examiné comment fournir une validation et une vérification précoce pour le langage SysML en utilisant la méthode B supportée par un outil existant. Nos objectifs étaient de considérer des modèles conçus au niveau de la mise en œuvre et d'utiliser cette méthode formelle de manière évolutive. Notre contribution repose sur l'alignement de SysML avec la méthode B en trois étapes :

— La première est la définition d'un sous-ensemble de la méthode B qui met l'accent sur une bonne décomposition.

— La seconde étape consiste à rechercher des similarités sémantiques entre SysML et le sous-ensemble précédemment identifié de la méthode B, afin de définir un sous-ensemble du langage SysML proche du sous-ensemble de B. Notre sous-ensemble retenu repose sur le diagramme de définition de bloc. Nous avons dû définir un profil SysML afin de combler un manque de certains concepts présents dans B et d'ajouter quelques relations entre exigences et blocs.

— La troisième étape repose sur des similitudes trouvées pour définir une transformation avec un écart sémantique aussi peu que possible. Nous l'avons implémenté en utilisant Acceleo. Nous avons appliqué cette transformation de SysML à B sur un exemple '. . . ' et avons réussi à prouver toutes les propriétés de sécurité.

5.2/ PERSPECTIVES

Nous désirons proposer dans un premier temps d'étendre notre étude et aller plus loin dans les concepts du langage B et manipuler les implémentations B en plus des machines abstraite et des raffinements pour terminer le cycle de développement entier.

Nous souhaiterions exploiter d'autres diagrammes SysML comme IBD (Internal Block Diagram)et Parametric Diagram afin de gérer les interactions à l'intérieur des composants du système.

Quelques améliorations dans SysML au niveau des contraintes exprimées en OCL nous aideront à améliorer notre plugin. Nous aimerions proposer à la communauté SysML/ Papyrus de pouvoir exprimer des contraintes dorectement en langage B en plus de java, OCL et du langage naturel.

Notre travail est orienté plus particulièrement vers l'étude des aspects structurels des systèmes, nous aimerions compléter cette étude en abordant les aspects fonctionnels et comportementaux.

LISTE DES ACRONYMES

UML Unified Modeling Language

SysML System Modeling Language

IE Ingénierie des Exigences

SE System Engineering

CBD Component Based Development

IDM Inginierie Dirigée par les Modèles

MDE Model Driven Engineering

OCL Object Constraint Langage

M2T Model to Text

M2M Model to Model

ALF Action Langage Foundational

MDE Model Driven Engineering

BDD Block Definition Diagram

RD Requirement Diagram

AFS Association Française d'Ingénierie Système

TCTL Timed Computation Tree Logic

BDD Binary Decision Diagram

DBM Difference Bound Matrices

MDA Model Driven Architecture

MDT Model Driven Testing

IDM Ingénierie Dirigée par les Modèles

API Application Programming Interface

XMI XML Metadata Interchange

MOF Meta Object Facility

EMF Eclipse Modeling Framework

SE System Engineering

TABLE DES FIGURES

BIBLIOGRAPHIE

[1] Engineering Air Traffic Control Systems with a Model-Driven Approach.

[2] Sur_les_principes_de_base_de_lingenierie_des_mode.

[3] Découvrir et comprendre l'Ingénierie Système Ouvrage collectif AFIS préparé par le Groupe de Travail Ingénierie Système. 2009.

[4] X 2004. MyCopy From Object-Orientation to Formal Methods. *Lecture Notes in Computer Science*.

[5] Henric Andersson, Erik Herzog, Gert Johansson, and Olof Johansson. Experience from introducing Unified Modeling Language/Systems Modeling Language at Saab Aerosystems. *Systems Engineering*, 2010.

[6] Ermeson Andrade, Paulo Maciel, Gustavo Callou, Eduardo Tavares, and Bruno Nogueira. Mapping sysml state machine diagram to time petri net for analysis and verification of embedded real-time systems with energy constraints, 11 2008.

[7] Olivier Barais. Séparation des préoccupations en phase de méta-modélisation. 2010.

[8] Jean Bézivin. On the unification power of models. *Software and Systems Modeling*, 2005.

[9] Erwan Bousse, Julien Deantoni, Benoit Baudry, and Benoit Combemale. Execution Trace Management to Support Dynamic V&V for Executable DSMLs.

[10] Erwan Bousse, David Mentré, Benoît Combemale, Benoît Baudry, and Takaya Katsuragi. Aligning SysML with the B method to provide V&V for systems engineering. *Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation - MoDeVVa '12*, (October) :11–16, 2012.

[11] Jean-michel Bruel and Sébastien Gérard. SysML par la pratique avec Papyrus. 2017.

[12] Benoît Combemale. Ingénierie Dirigée par les Modèles (IDM) État de l'art. 2008.

[13] Krzysztof Czarnecki and Simon Helsen. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture Classification of Model Transformation Approaches.

[14] S Ephane Galland. ethodologie et outils pour la simulation multiagent dans des univers virtuels.

[15] Par E Jacqueline Kona, Paul Sabatier, and Jean-Pierre Bourey. APPROCHE SYS EME POUR LA CONCEPTION D'UNE ETHODOLOGIE POUR L'ELICITATION COLLABORATIVE DES EXIGENCES.

[16] Sébastien Lahaye. Contributions à l'étude des systèmes à événements discrets à partir de modèles définis sur des semi-anneaux idempotents. 2011.

[17] Régine Laleau, Farida Semmak, Abderrahman Matoussi, Dorian Petit, Ahmed Hammad, and Bruno Tatibouet. A first attempt to combine SysML requirements diagrams and B. *Innovations in Systems and Software Engineering*, 6(1) :47–54, 2010.

[18] Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering : A Roadmap.

[19] Haroldo José Onisto, Tiago De, Moraes Machado, Ramon Cravo Fernandes, Johannes Dantas De Medeiros, Iliézer Tamagno, Tiago César Dezotti, and José Eduardo Bertuzzo. Model-driven Engineering Applied to the Development of Embedded Software for B-mode Ultrasound Imaging Systems – A Case Study.

[20] Doug Rosenberg with Sam Mancarella. Embedded Systems Development using SysML.

[21] Nicolas Sannier and Benoit Baudry. Toward multilevel textual requirements traceability using model-driven engineering and information retrieval. In *Proceedings of the Model-Driven Requirements Engineering workshop (MoDRE'12) at RE'12*, pages –, Chicago, USA, September 2012.

[22] Jonathan Sprinkle, Bernhard Rumpe, Hans Vangheluwe, and Gabor Karsai. State of the Art and Research Challenges.

Document réalisé avec LaTeX et :
le style LaTeX pour Thèse de Doctorat créé par S. Galland — http://www.multiagent.fr/ThesisStyle
la collection de paquets `tex-upmethodology` — http://www.arakhne.org/tex-upmethodology/

Résumé

L'ingénierie des exigences présente de nombreux défis pour les chercheurs car les systèmes logiciels deviennent de plus en plus complexes et leur mise en œuvre demande des approches de modélisation plus rigoureuses.
Notre but est de Combiner des méthodes d'ingénierie des exigences avec des méthodes formelles de spécification et de vérification. L'idée est de proposer une extension du modèle SYSML par des concepts existants dans le domaine de l'ingénierie des exigences puis d'implémenter les règles de transformation de cette extension SYSML vers des spécifications B formelles pour vérifier l'architecture du système.

Mots clés : *vérification, modélisation, méthode B, SysML, diagramme de bloc de définition, diagramme d'exigences*

Abstract

Requirements engineering presents many challenges for researchers as software systems become more complex and their implementation requires more rigorous modeling approaches. Our goal is to combine requirements engineering methods with formal specification and verification methods. The idea is to propose an extension of the SYSML model by existing concepts in the field of requirements engineering and then to implement the transformation rules of this SYSML extension to formal B specifications to verify the architecture of the system.

Keywords: *verification, modeling, method B, SysML, Block Definition Diagram, Requirement Diagram*

ملخص

يجد الباحثون العديد من التحديات في هندسة المتطلبات، خاصة وأن أنظمة البرمجيات اصبحت معقدة للغاية مما يجعل تطبيقها يتطلب مقاربات لنماذج اكثر صرامة. لذلك هدفنا من هذه الدراسة هو الجمع بين متطلبات الأساليب الهندسية مع المواصفات الرسمية وطرق التحقق، و تتمثل الفكرة في اقتراح امتداد لنموذج SysML من خلال المفاهيم الموجودة في مجال المتطلبات الهندسية ثم تطبيق قواعد تحويل ملحق SysML الى مواصفات B الرسمية للتحقق من بنية النظام.